Dedication
For Jocelyn and Dashiel, the two most dynamic forces in my life.
Special Thanks to:

Tara, my soul mate and best critic.

Dad and Nancy who made me who I am.

Uncle Johnny, for his unwavering support.

Pat and Red, my two biggest fans.

Nancy P., who kept the project going.

Bob, Darren, and Liz, who dotted my i's and made sure that everything made sense.

Thomas, who was always there when I needed help.

Heather, who gave me a chance when I needed it most.

Judy, Boyd, Dr. G and teachers everywhere who care. Keep up the good work.

Charles Dodgson (aka Lewis Carroll), for writing Alice’s Adventures in Wonderland.

John Tenniel & Arthur Rackham, for their incredible illustrations of Alice’s Adventures in Wonderland.


BBC 6 Music, The Craig Charles Funk and Soul Show, Rasputina, Electric Six, Cake, Client, Jonathan Coulton, Cracker, Nine Inch Nails, Bitter:Sweet, Metric, Captain Sensible, HIDE, Origa, Richard Hawley, the Pogues, New Model Army, Cocteau Twins, Dead Can Dance, the Sisters of Mercy, the Smiths, Mojo Nixon, Bauhaus, Lady Tron, David Bowie, Bad Religion, The Black Belles, T. Rex, Bad Religion, Dr. Rubberfunk, Smoove and Turell, Dury, The Kinks, This Mortal Coil, Rancid, Monty Python, the Dead Milkmen, New Order, Regina Spektor, The Sex Pistols, Beethoven, Bach, Brahms, Handel, Mozart, Liszt, Vivaldi, Holst, Synergy, and Garrison Keillor (for The Writer’s Almanac) whose noise helped keep me from going insane while writing this book.
# Contents at a Glance

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction</td>
<td></td>
<td>xiii</td>
</tr>
<tr>
<td><strong>Chapter 1</strong></td>
<td>Understanding CSS3</td>
<td>1</td>
</tr>
<tr>
<td><strong>Chapter 2</strong></td>
<td>HTML5 Primer</td>
<td>17</td>
</tr>
<tr>
<td><strong>Chapter 3</strong></td>
<td>CSS Basics</td>
<td>35</td>
</tr>
<tr>
<td><strong>Chapter 4</strong></td>
<td>Selective Styling</td>
<td>69</td>
</tr>
<tr>
<td><strong>Chapter 5</strong></td>
<td>Font Properties</td>
<td>123</td>
</tr>
<tr>
<td><strong>Chapter 6</strong></td>
<td>Text Properties</td>
<td>157</td>
</tr>
<tr>
<td><strong>Chapter 7</strong></td>
<td>Color and Background Properties</td>
<td>183</td>
</tr>
<tr>
<td><strong>Chapter 8</strong></td>
<td>List and Table Properties</td>
<td>219</td>
</tr>
<tr>
<td><strong>Chapter 9</strong></td>
<td>User Interface and Generated Content Properties</td>
<td>235</td>
</tr>
<tr>
<td><strong>Chapter 10</strong></td>
<td>Box Properties</td>
<td>247</td>
</tr>
<tr>
<td><strong>Chapter 11</strong></td>
<td>Visual Formatting Properties</td>
<td>283</td>
</tr>
<tr>
<td><strong>Chapter 12</strong></td>
<td>Transformation and Transition Properties</td>
<td>307</td>
</tr>
<tr>
<td><strong>Chapter 13</strong></td>
<td>Essential Design and Interface Techniques</td>
<td>327</td>
</tr>
<tr>
<td><strong>Chapter 14</strong></td>
<td>Responsive Web Design</td>
<td>349</td>
</tr>
<tr>
<td><strong>Chapter 15</strong></td>
<td>CSS Best Practices</td>
<td>375</td>
</tr>
<tr>
<td><strong>Appendix A</strong></td>
<td>CSS Quick Reference</td>
<td>409</td>
</tr>
<tr>
<td><strong>Appendix B</strong></td>
<td>HTML and UTF Character Encoding</td>
<td>423</td>
</tr>
<tr>
<td>Index</td>
<td></td>
<td>429</td>
</tr>
</tbody>
</table>
# Table of Contents

<table>
<thead>
<tr>
<th>Chapter 1</th>
<th>Understanding CSS3</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>What Is a Style?</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>What Are Cascading Style Sheets?</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>The Evolution of CSS</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>CSS and HTML</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>Types of CSS Rules</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>The Parts of a CSS Rule</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>CSS Browser Extensions</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>New in CSS3</td>
<td>14</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Chapter 2</th>
<th>HTML5 Primer</th>
<th>17</th>
</tr>
</thead>
<tbody>
<tr>
<td>What Is HTML?</td>
<td>18</td>
<td></td>
</tr>
<tr>
<td>Types of HTML Elements</td>
<td>21</td>
<td></td>
</tr>
<tr>
<td>The Evolution of HTML5</td>
<td>24</td>
<td></td>
</tr>
<tr>
<td>What’s New in HTML5?</td>
<td>27</td>
<td></td>
</tr>
<tr>
<td>How Does HTML5 Structure Work?</td>
<td>28</td>
<td></td>
</tr>
<tr>
<td>Using HTML5 Structure Now</td>
<td>29</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Chapter 3</th>
<th>CSS Basics</th>
<th>35</th>
</tr>
</thead>
<tbody>
<tr>
<td>The Basic CSS Selectors</td>
<td>36</td>
<td></td>
</tr>
<tr>
<td>Inline: Adding Styles to an HTML Tag</td>
<td>37</td>
<td></td>
</tr>
<tr>
<td>Embedded: Adding Styles to a Web Page</td>
<td>40</td>
<td></td>
</tr>
<tr>
<td>External: Adding Styles to a Web Site</td>
<td>43</td>
<td></td>
</tr>
<tr>
<td>(Re)Defining HTML Tags</td>
<td>50</td>
<td></td>
</tr>
<tr>
<td>Defining Reusable Classes</td>
<td>53</td>
<td></td>
</tr>
<tr>
<td>Defining Unique IDs</td>
<td>57</td>
<td></td>
</tr>
<tr>
<td>Defining Universal Styles</td>
<td>61</td>
<td></td>
</tr>
<tr>
<td>Grouping: Defining Elements That Are Using the Same Styles</td>
<td>64</td>
<td></td>
</tr>
<tr>
<td>Adding Comments to CSS</td>
<td>67</td>
<td></td>
</tr>
<tr>
<td>Chapter</td>
<td>Title</td>
<td>Pages</td>
</tr>
<tr>
<td>---------</td>
<td>-------</td>
<td>-------</td>
</tr>
<tr>
<td><strong>Chapter 7</strong></td>
<td>Color and Background Properties</td>
<td>183</td>
</tr>
<tr>
<td>Getting Started</td>
<td>184</td>
<td></td>
</tr>
<tr>
<td>Choosing Color Values</td>
<td>185</td>
<td></td>
</tr>
<tr>
<td>✷ Creating Color Gradients</td>
<td>191</td>
<td></td>
</tr>
<tr>
<td>Choosing Your Color Palette</td>
<td>196</td>
<td></td>
</tr>
<tr>
<td>Setting Text Color</td>
<td>202</td>
<td></td>
</tr>
<tr>
<td>Setting a Background Color</td>
<td>204</td>
<td></td>
</tr>
<tr>
<td>✷ Setting Background Images</td>
<td>205</td>
<td></td>
</tr>
<tr>
<td>Using Background Shorthand to Add Multiple Background Images and Gradients</td>
<td>212</td>
<td></td>
</tr>
<tr>
<td>Putting It All Together</td>
<td>217</td>
<td></td>
</tr>
</tbody>
</table>

| Chapter 8 | List and Table Properties | 219 |
| Getting Started | 220 |
| Setting the Bullet Style | 223 |
| Creating Your Own Bullets | 224 |
| Setting Bullet Positions | 225 |
| Setting Multiple List Styles | 226 |
| Setting the Table Layout | 228 |
| Setting the Space Between Table Cells | 229 |
| Collapsing Borders Between Table Cells | 230 |
| Dealing with Empty Table Cells | 232 |
| Setting the Position of a Table Caption | 233 |
| Putting It All Together | 234 |

| Chapter 9 | User Interface and Generated Content Properties | 235 |
| Getting Started | 236 |
| Changing the Mouse Pointer Appearance | 238 |
| Adding Content Using CSS | 240 |
| Teaching the Browser to Count | 242 |
| Specifying the Quote Style | 244 |
| Putting It All Together | 246 |

| Chapter 10 | Box Properties | 247 |
| Understanding an Element’s Box | 250 |
| Displaying an Element | 252 |
| Setting the Width and Height of an Element | 255 |
Table of Contents

**Controlling Overflowing Content** . . . 259
Floating Elements in the Window . . . 262
Setting an Element’s Margins . . . 265
Setting an Element’s Outline . . . 268
Setting an Element’s Border . . . 269
★ Rounding Border Corners . . . 272
★ Setting a Border Image . . . 274
Setting an Element’s Padding . . . 276
★ Creating a Multi-Column Text Layout . . . 278
Coming Soon! . . . 280
Putting It All Together . . . 281

**Chapter 11** Visual Formatting Properties . . . 283
Getting Started . . . 284
Understanding the Window and Document . . . 286
Setting the Positioning Type . . . 288
Setting an Element’s Position . . . 292
Stacking Objects in 3D . . . 294
Setting the Visibility of an Element . . . 296
Clipping an Element’s Visible Area . . . 298
★ Setting an Element’s Opacity . . . 300
★ Setting an Element’s Shadows . . . 302
Putting It All Together . . . 305

**Chapter 12** Transformation and Transition Properties . . . 307
Getting Started . . . 308
★ Transforming an Element . . . 311
★ Adding Transitions Between Element States . . . 320
Putting It All Together . . . 325

**Chapter 13** Essential Design and Interface Techniques . . . 327
Getting Started . . . 328
Creating Multicolumn Layouts with Float . . . 330
Fixing the Float . . . 334
Styling Links vs. Navigation . . . 339
Using CSS Sprites . . . 342
Creating a CSS Drop-Down Menu . . . 345
Putting It All Together . . . 347
This page intentionally left blank
These days, everyone is a Web designer. Whether you are adding a comment to a Facebook page, creating your own blog, or building a Fortune 50 Web site, you are involved in Web design.

As the Web expands, everyone from PTA presidents to presidents of multinational corporations is using this medium to get messages out to the world because the Web is the most effective way to communicate your message to the people around you and around the world.

Knowing how to design for the Web isn’t always about designing complete Web sites. Many people are creating simple Web pages for auction sites, their own photo albums, or their blogs. So, whether you are planning to redesign your corporate Web site or place your kid’s graduation pictures online, learning Cascading Style Sheets (CSS) is your next step into the larger world of Web design.

What Is This Book About?

HTML is how Web pages are structured. CSS is how Web pages are designed. This book deals primarily with how to use CSS to add a visual layer to the HTML structure of your Web pages.

CSS is a style sheet language; that is, it is not a programming language. Instead, it’s code that tells a device (usually a Web browser) how the content in a file should be displayed. CSS is meant to be easily understood by anyone, not just “computer people.” Its syntax is straightforward, basically consisting of rules that tell an element on the screen how it should appear.

This book also includes the most recent additions to the CSS language, commonly referred to as CSS3 (or CSS Level 3). CSS3 builds on and extends the previous version of CSS. For the time being, it’s important to understand what is new in CSS3 because some browsers (most notably Internet Explorer) have incomplete support or no support for these new features.
CSS Visual QuickStart Guide has three parts:

- **CSS Introduction and Syntax (Chapters 1–4)**—This section lays the foundation you require to understand how to assemble basic style sheets and apply them to a Web page. It also gives you a crash course in HTML5.

- **CSS Properties (Chapters 5–12)**—This section contains all the styles and values that can be applied to the elements that make up your Web pages.

- **Working with CSS (Chapters 13–15)**—This section gives advice and explains best practices for creating Web pages and Web sites using CSS.

**Who is this book for?**

To understand this book, you need to be familiar with HTML (Hypertext Markup Language). You don’t have to be an expert, but you should know the difference between a `<p>` element and a `<bx>` tag. That said, the more knowledge of HTML you bring to this book, the more you’ll get out of it.

Chapter 2 deals briefly with HTML5, bringing you up to date on the latest changes. If you are already familiar with HTML, this chapter has everything you will need to get going.

**What tools do you need for this book?**

The great thing about CSS is that, like HTML, it doesn’t require any special or expensive software. Its code is just text, and you can edit it with programs as simple as TextEdit (Mac OS) or Notepad (Windows).
Why Standards (Still) Matter

The idea of a standard way to communicate over the Internet was the principle behind the creation of the World Wide Web: You should be able to transmit information to any computer anywhere in the world and display it in the way the author intended. In the beginning, only one form of HTML existed, and everyone on the Web used it. This situation didn’t present any real problem because almost everyone used Mosaic, the first popular graphics-based browser, and Mosaic was the standard. That, as they say, was then.

Along came Netscape Navigator and the first HTML extensions were born. These extensions worked only in Netscape, however, and anyone who didn’t use that browser was out of luck. Although the Netscape extensions defied the standards of the World Wide Web Consortium (W3C), most of them—or at least some version of them—eventually became part of those very standards. According to some people, the Web has gone downhill ever since.

The Web is a very public form of discourse, the likes of which has not existed since people lived in villages and sat around the campfire telling stories every night. The problem is that without standards, not everyone in the global village can make it to the Web campfire. You can use as many bleeding-edge techniques as you like. You can include Flash, JavaScript, QuickTime video, Ajax, HTML5, or CSS3, but if only a fraction of browsers can see your work, you’re keeping a lot of fellow villagers out in the cold.

When coding for this book, I spent 35 to 45 percent of my time trying to get the code to run as smoothly as possible in Internet Explorer, Firefox (and related Mozilla browsers), Opera, Safari, and Chrome. This timeframe holds true for most of my Web projects; much of the coding time is spent on cross-browser inconsistencies. If the browsers stuck to the standards, this time would be reduced to almost nothing. Your safest bet as a designer, then, is to know the standards of the Web, try to use them as much as possible, and demand that the browser manufacturers use them as well.
Values and Units Used in This Book

Throughout this book, you’ll need to enter various values to define properties. These values take various forms, depending on the needs of the property. Some values are straightforward—a number is a number—but others have special units associated with them.

Values in angle brackets (< >) represent one type of value (Table i.1) that you will need to choose, such as <length> (a length value like 12px) or <color> (a color value). Words that appear in code font are literal values and should be typed exactly as shown, such as normal, italic, or bold.

Length values

Length values come in two varieties:

- **Relative values**, which vary depending on the computer being used (Table i.2).
- **Absolute values**, which remain constant regardless of the hardware and software being used (Table i.3).

I generally recommend using ems to describe font sizes for the greatest stability between operating systems and browsers.

### Table i.1 Value Types

<table>
<thead>
<tr>
<th>Value Type</th>
<th>What It Is</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;number&gt;</td>
<td>A number</td>
<td>1, 2, 3</td>
</tr>
<tr>
<td>&lt;length&gt;</td>
<td>A measurement of distance or size</td>
<td>1in</td>
</tr>
<tr>
<td>&lt;color&gt;</td>
<td>A chromatic expression</td>
<td>red</td>
</tr>
<tr>
<td>&lt;percentage&gt;</td>
<td>A proportion</td>
<td>35%</td>
</tr>
<tr>
<td>&lt;URL&gt;</td>
<td>The absolute or relative path to a file on the Internet</td>
<td><a href="http://www.mySite.net/images/01.jpg">http://www.mySite.net/images/01.jpg</a></td>
</tr>
</tbody>
</table>

### Table i.2 Relative Length Values

<table>
<thead>
<tr>
<th>Unit</th>
<th>Name</th>
<th>What It Is</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>em</td>
<td>Em</td>
<td>Relative to the current font size (similar to percentage)</td>
<td>3em</td>
</tr>
<tr>
<td>ex</td>
<td>x-height</td>
<td>Relative to the height of lowercase letters in the font</td>
<td>5ex</td>
</tr>
<tr>
<td>px</td>
<td>Pixel</td>
<td>Relative to the monitor’s resolution</td>
<td>125px</td>
</tr>
</tbody>
</table>

### Table i.3 Absolute Length Values

<table>
<thead>
<tr>
<th>Unit</th>
<th>Name</th>
<th>What It Is</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>pt</td>
<td>Point</td>
<td>72pt = 1inch</td>
<td>12pt</td>
</tr>
<tr>
<td>pc</td>
<td>Picas</td>
<td>1pc = 12pt</td>
<td>3pc</td>
</tr>
<tr>
<td>mm</td>
<td>Millimeters</td>
<td>1mm = .24pc</td>
<td>25mm</td>
</tr>
<tr>
<td>cm</td>
<td>Centimeters</td>
<td>1cm = 10mm</td>
<td>5.1cm</td>
</tr>
<tr>
<td>in</td>
<td>Inches</td>
<td>1in = 2.54cm</td>
<td>8.25in</td>
</tr>
</tbody>
</table>
Color values

You can describe color on the screen in a variety of ways, but most of these descriptions are just different ways of telling the computer how much red, green, and blue are in a particular color.

Chapter 7 provides an extensive explanation of color values.

Percentages

Many of the properties in this book have a percentage as their values. The behavior of each percentage value depends on the property in use.

URLs

A Uniform Resource Locator (URL) is the unique address of something on the Web. This resource could be an HTML document, a graphic, a CSS file, a JavaScript file, a sound or video file, a CGI script, or any of a variety of other file types. URLs can be local—describing the location of the resource relative to the current document—or global—describing the absolute location of the resource on the Web and beginning with http://.
Reading This Book

For the most part, the text, tables, figures, code, and examples should be self-explanatory. But you need to know a few things in advance to understand this book.

CSS value tables

Each section that explains a CSS property includes a quick-reference table of the values that the property can use, as well as the browsers compatible with those values. Table i.4 lists the browser icons and abbreviations used in this book.

<table>
<thead>
<tr>
<th>Icon</th>
<th>Abbreviation</th>
<th>Browser</th>
</tr>
</thead>
<tbody>
<tr>
<td>🌐</td>
<td>IE</td>
<td>Microsoft Internet Explorer</td>
</tr>
<tr>
<td>🌐</td>
<td>FF</td>
<td>Mozilla Firefox</td>
</tr>
<tr>
<td>🌐</td>
<td>Op</td>
<td>Opera</td>
</tr>
<tr>
<td>🌐</td>
<td>Sa</td>
<td>Apple Safari</td>
</tr>
<tr>
<td>🌐</td>
<td>Ch</td>
<td>Google Chrome</td>
</tr>
</tbody>
</table>

Text-Overflow Values

<table>
<thead>
<tr>
<th>Value</th>
<th>🌐</th>
<th>🌐</th>
<th>🌐</th>
<th>🌐</th>
<th>🌐</th>
</tr>
</thead>
<tbody>
<tr>
<td>clip</td>
<td><img src="image" alt="circle" /></td>
<td><img src="image" alt="9" /></td>
<td><img src="image" alt="●" /></td>
<td><img src="image" alt="●" /></td>
<td><img src="image" alt="●" /></td>
</tr>
<tr>
<td>ellipsis</td>
<td><img src="image" alt="circle" /></td>
<td><img src="image" alt="●" /></td>
<td><img src="image" alt="●" /></td>
<td><img src="image" alt="●" /></td>
<td><img src="image" alt="●" /></td>
</tr>
<tr>
<td>inherit</td>
<td><img src="image" alt="circle" /></td>
<td><img src="image" alt="●" /></td>
<td><img src="image" alt="●" /></td>
<td><img src="image" alt="●" /></td>
<td><img src="image" alt="●" /></td>
</tr>
</tbody>
</table>

A circle indicates browser support. A diamond indicates support with browser extension (-moz-, -webkit-, -o-, or -e-). A blank entry indicates no support. If a number is added, indicates support is recent since that version.

The property tables show you the values available with a property, the earliest browser version in which the value is available, and with which version of CSS the value was introduced.
The Code

For clarity and precision, this book uses several layout techniques to help you see the difference between the text of the book and the code.

Code looks like this:

```html
<p {...}</style>

All code in this book is presented in lowercase. In addition, quotes in the code always appear as straight quotes (" or '), not curly quotes (‘ or ’). There is a good reason for this distinction. Curly quotes (also called smart quotes) will cause the code to fail.

When you type a line of code, the computer can run the line as long as needed; but in this book, lines of code have to be broken to make them fit on the page. When that happens, you’ll see a gray arrow ➝, indicating that the line of code is continued from above, like this:

```
.title { font: bold 28pt/26pt times, serif; color: #FFF; background color: #000; background-image: url(bg_title.gif); }
```

A numbered step often includes a line of code in red from the main code block:

```html
p { color: red; }
```

This is a reference to help you pinpoint where that step applies in the larger code block that accompanies the task. This code will be highlighted in red in the code listing to help you more easily identify it.
Web Site for This Book

I hope you'll be using a lot of the code from this book in your Web pages, and you are free to use any code in this book without asking my permission (although a mention of the book is always appreciated). However, be careful—retying information can lead to errors. Some books include a CD-ROM containing all the code from the book, and you can copy it from that disc. But guess who pays for that CD? You do. And CDs aren't cheap.

But if you bought this book, you already have access to the largest resource of knowledge that ever existed: the Web. And that's exactly where you can find the code from this book.

My support site for this Visual QuickStart Guide is at www.jasonspeaking.com/css3vqs.

This site includes all the code you see in the book, as well as quick-reference charts. You can download the code and any important updates and corrections from this site.
Selective Styling

It’s not enough to style a Web page element. The art of CSS—and thus the art of Web design—is the ability to style elements based on their context. You must consider where an element is in the document; which elements surround it; its attributes, content, and dynamic state; and even the platform displaying the element (screen, handheld device, TV, and so on).

Selective styling is the closest that CSS gets to traditional computer programming, allowing you to style elements if they meet certain criteria. This level of styling can get increasingly complex, so it’s important, at least in this chapter, to start out as simply as possible and build a firm foundation of understanding.

In This Chapter

- The Element Family Tree 70
- Defining Styles Based on Context 71
- Working with Pseudo-Classes 82
- Working with Pseudo-Elements 96
- Defining Styles Based on Tag Attributes 100
- Querying the Media 104
- Inheriting Properties from a Parent 115
- Making a Declaration !important 117
- Determining the Cascade Order 119
The Element Family Tree

When a tag is surrounded by another tag—one inside another—the tags are nested.

In a nested set, the outer element in this example (\texttt{<h2>}) is called the parent, and the inner element (\texttt{<strong>}) is the child. The child tag and any children of that child tag are the parents’ descendents. Two tags in the same parent are called siblings, and two tags immediately next to each other are adjacent siblings.

- **Parent elements** contain other elements (children). Child elements will often inherit styles from a parent element.
- **Descendent elements** are any elements within another element.
- **Child elements** are first-generation descendents in relation to the parent. Second generation and higher elements are sometimes referred to as grandchildren.
- **Adjacent or preceding sibling elements** are child elements of the same generation that are immediately next to each other in the HTML code.

In Chapter 3, you learned ways to specify the styles of an individual element regardless of where it is placed in the HTML code. However, CSS also lets you specify the element’s style depending on its context. Using contextual selectors, you can specify styles based on a tag’s relationship to other tags, classes, or IDs on the page.
Selective Styling

Defining Styles Based on Context

Contextual styles allow you to specify how a particular element should appear based on its parents and siblings. For example, you may want an emphasis tag to appear one way when it’s in the main header of the page and differently when it appears in the sub-header. You may want still another appearance in a paragraph of text. These combinatorial selectors (Table 4.1) are among the most used and useful in CSS.

Styling descendents

You can style individual descendent elements depending on their parent selector or selectors in a space-separated list. The last selector will receive the style if and only if it is the descendent of the preceding selectors A.

When you want to indicate that the exact selector does not matter at any given level, you can use the universal selector (*) described in Chapter 3 B.

<table>
<thead>
<tr>
<th>Format</th>
<th>Selector Name</th>
<th>Elements Are Styled If…</th>
</tr>
</thead>
<tbody>
<tr>
<td>a b c</td>
<td>Descendent</td>
<td>c descendent of b descendent of a</td>
</tr>
<tr>
<td>a * b</td>
<td>Universal</td>
<td>b within a regardless of b’s parents</td>
</tr>
<tr>
<td>a &gt; b</td>
<td>Direct child</td>
<td>b direct child of a</td>
</tr>
<tr>
<td>a + b</td>
<td>Adjacent sibling</td>
<td>sibling b immediately after a</td>
</tr>
<tr>
<td>a ~ b</td>
<td>General sibling</td>
<td>sibling b anywhere after a</td>
</tr>
</tbody>
</table>
To style descendent elements:

1. Set up a list of descendent selectors.
   Type the HTML selector of the parent tag, followed by a space, and then the final child or another parent (Code 4.1).
   ```html
   article.chaptext p strong em
   → {...}
   ```
   You can type as many HTML selectors as you want for as many parents as the nested tag will have, but the last selector in the list is the one that receives all the styles in the rule.

2. Styles will be used only if the pattern is matched.
   ```html
   <article class="chaptext"><p>
   → <strong><em>...</em></strong>
   → </p></article>
   ```
   The style will be applied if and only if the final selector occurs as a descendent nested within the previous selectors. So, in this example, the emphasis tag (em) is styled only if it is in a paragraph (strong) that is within a paragraph tag (p), that is in an article tag using the class chaptext (article.chaptext).
   The emphasis tag would not be styled by the code in Step 1 in the following case, because it is not in a strong tag:
   ```html
   <article class="chaptext"><p>
   → <em>...</em></p></article>
   ```
   And emphasis will not be styled by the code in Step 1 in the following case because the article tag does not have the chaptext class:
   ```html
   <article><p><strong><em>...</em></strong>
   → </p></article>
   ```
   It is important to note, though, that although the selectors do not style the emphasis tag in these last two cases, it does not mean that styles from other declarations will not do so.
The style is set for the emphasis tag if its parents are the `h1` tag and the `article` tag using the `copy` class.

```html
<DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<title>Alice’s Adventures in Wonderland</title>
<style type="text/css" media="all">
article.chaptertext p strong em {
  color: red;
  font-weight: normal;
  font-size: 2em;
  font-style: normal;
}
</style>
</head>
<body>
<article class="chaptertext">
<p><strong>Alice was beginning to get <em>very tired</em> of sitting by her sister on the bank, and of having nothing to do: once or twice she had peeped into the book her sister was reading, but it <em>had no pictures or conversations in it</em>, and <em>what is the use of a book, </em>thought Alice<em>?</em>, <q>without pictures or conversations?</q></p>
</article>
</body>
</html>
```

The results of Code 4.1. The only text that meets the selective criteria is in red, which is only the emphasis tag in the `h1`, in this example.
To style descendents universally:

1. Set up a list of descendent selectors including a universal selector. Type the HTML selector of the parent tag, followed by a space, and then an asterisk (*) or other selectors (Code 4.2).
   
   \[
   \text{article.chaptertext p * em \{...\}}
   \]

2. Styles will be used only if the pattern is matched. Generally, the universal selector is used at the end of a list of selectors so that the style is applied explicitly to all of a parent’s direct descendents (children). However, the styles will not be directly applied to those children’s descendents.

In this example, the style is applied to the emphasis tag inside any parent tag (such as \texttt{strong}) in a paragraph, such as:

\[
\langle \text{article class="chaptertext"} \rangle \langle p \rangle \text{\rightarrow } \langle \text{strong} \rangle \langle \text{em} \rangle ... \langle /\text{em} \rangle \langle /\text{strong} \rangle \text{\rightarrow } \langle /p \rangle \langle /\text{article} \rangle
\]

Or:

\[
\langle \text{article class="chaptertext"} \rangle \langle p \rangle \text{\rightarrow } \langle q \rangle \langle \text{em} \rangle ... \langle /\text{em} \rangle \langle /q \rangle \langle /p \rangle \langle /\text{article} \rangle
\]

However, an emphasis tag that is \texttt{not} in another tag in the paragraph will \texttt{not} be styled.

\[
\langle \text{article class="chaptertext"} \rangle \langle p \rangle \text{\rightarrow } \langle \text{em} \rangle ... \langle /\text{em} \rangle \langle /p \rangle \langle /\text{article} \rangle
\]

\textbf{TIP} Like grouped selectors, contextual selectors can include class selectors (dependent or independent), ID selectors in the list, and HTML selectors.
The results of Code 4.2. The text in red matches the selective criteria with the universal selector. In this case, all emphasis tags match.
Styling only the children

If you want to style only a parent’s child elements (not a grandchild descendent), you must specify the parent selector and child selector, separated by a right angle bracket (>)

To define child selectors:

1. Set up a list of direct child selectors. Type the selector for the parent element (HTML, class, or ID), followed by a right angle bracket (>) and the child selector (HTML, class, or ID).

   \[ \text{article.chaptertext} \, > \, \text{p} \, > \, \text{em} \, \{ \ldots \} \]

   You can repeat this as many times as you want with the final selector being the target to which you apply the styles (Code 4.3). You can have one space between the selector and the greater-than sign or no spaces.

2. Styles are used only if the pattern is matched.

   \[
   \begin{align*}
   & <\text{article class="chaptertext"}> <p> \rightarrow <em>... </em> </p> <p> </p> <p> \end{align*}
   \]

   The styles from Step 1 are applied if and only if the final selector is an immediate child element nested in the preceding element. Placing the tag within any other HTML tags will disrupt the pattern. In this example, the emphasis tag (em) is styled only if it is in a paragraph (p) within an article (article).

   However, any emphasis tag that is in another tag will not be styled:

   \[
   \begin{align*}
   & <\text{article class="chaptertext"}> <p> \rightarrow <q> <em>... </em> </q> <p> \end{align*}
   \]
The style is applied to the emphasis tag only if it is a child of a paragraph that is in turn the child of an article tag using the copy class.

```
<article class="chaptertext">
  <p><strong>Alice was beginning to get very tired</strong> of sitting by her sister on the bank, and of having nothing to do: <strong>once or twice</strong> she had peeped into the book her sister was reading, but it <em>had no pictures or conversations in it</em>, and <em>what is the use of a book</em>?, thought Alice, "without pictures or conversations?"
</p>
</article>
```

The results of Code 4.3. The text in red matches the direct child criteria. In this case the emphasis tags match within the paragraphs but not within the headers.
Styling siblings

Siblings are elements that have the same parent. You can style a sibling that is immediately adjacent to another or occurs anywhere after that sibling.

To define adjacent sibling selectors:

1. Set up a list of adjacent sibling selectors. Type the selector for the first element (HTML, class, or ID), a plus sign (+), and then the selector (HTML, class, or ID) for the adjacent element to which you want the style applied (Code 4.4).

   \[
   \text{strong} + \text{em} \{\ldots\}
   \]

2. Styles will be used only if the pattern is matched.

   \[
   <\text{strong}>\ldots</\text{strong}>\ldots<\text{em}>\ldots</\text{em}>
   \]

   The styles will be applied to any sibling that occurs immediately after the preceding selector with no other selectors in the way. Placing any element between them (even a break tag) will disrupt the pattern. The following pattern will not work:

   \[
   <\text{strong}>\ldots</\text{strong}>\ldots<\text{q}>\ldots</\text{q}>
   \rightarrow <\text{em}>\ldots</\text{em}>
   \]
Code 4.4 The style is applied to the emphasis tag only if it is in a paragraph that is immediately after another paragraph.

```html
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<title>Alice's Adventures in Wonderland</title>
<style type="text/css" media="all">
strong + em {
  color: red;
  background: silver;
  font-size: 2em;
}
</style>
</head>
<body>
<article class="chaptertext">
<p><strong>Alice was beginning to get very tired</strong> of sitting by her sister on the bank, and of having nothing to do: <strong>once or twice</strong> she had peeped into the book her sister was reading, but it <em>had no pictures or conversations in it</em>, and <em>what is the use of a book</em>, thought Alice, "without pictures or conversations?"  
</p>
</article>
</body>
</html>
```

The results of Code 4.4. The text in red matches the adjacent sibling criteria—the emphasis tags within the second and third paragraphs in this case—but does not match the fourth paragraph because a block quote is in the way.
To define general sibling selectors:

1. Set up a list of general sibling selectors. Type the selector for the first sibling element (HTML, class, or ID), a tilde sign (~), and then another selector (HTML, class, or ID) (Code 4.5).

   \texttt{strong ~ em {...}}

   You can repeat this as many times as necessary, but the last selector in the list is the one you are targeting to be styled.

2. Styles are used only if the pattern is matched.

   \texttt{<strong>...</strong>...<em>...</em> → ...<q>...<q>...<em>...<em>}

   The styles are applied to any siblings that occur after the first sibling selector, not just the first one. Unlike the adjacent sibling, this is true even when other types of tags are located in between. In the case above, this includes both the second and third \texttt{strong} tags.

\textbf{TIP} Although the universal selector shown in this section is used with the combinatory selectors, it can be used with any selector type. Table 4.2 shows how you can apply it.

<table>
<thead>
<tr>
<th>Format</th>
<th>Elements Are Styled If...</th>
</tr>
</thead>
<tbody>
<tr>
<td>a * b</td>
<td>b within a regardless of b's parents</td>
</tr>
<tr>
<td>a &gt; * &gt; b</td>
<td>b is the direct child of any element that is the direct child of a</td>
</tr>
<tr>
<td>a + * + b</td>
<td>sibling b immediately after any element that is immediately after a</td>
</tr>
<tr>
<td>:hover</td>
<td>mouse pointer over any element</td>
</tr>
<tr>
<td>:disabled</td>
<td>any element that is disabled</td>
</tr>
<tr>
<td>:first-child</td>
<td>first child of any element</td>
</tr>
<tr>
<td>:lang()</td>
<td>any element using specified language code</td>
</tr>
<tr>
<td>:not(s)</td>
<td>any element that is not the using indicated selectors</td>
</tr>
<tr>
<td>:first-letter</td>
<td>any element's first letter</td>
</tr>
</tbody>
</table>
Code 4.5 The style is applied to the emphasis tag if it is in a paragraph with any preceding sibling that is a paragraph.

```html
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Alice's Adventures in Wonderland</title>
  <style type="text/css" media="all">
    strong ~ em { 
      color: red;
      background: gray;
      font-size: 2em; 
    }
  </style>
</head>
<body>
<article class="chaptertext">
  <p><strong>Alice was beginning to get very tired</strong> of sitting by her sister on the bank, and of having nothing to do: <strong>once or twice</strong> she had peeped into the book her sister was reading, but it <em>had no pictures or conversations in it</em>, <q>and what is the use of a book, </q> <strong>thought Alice</strong>, <q>without pictures or conversations?</q></p>
</article>
</body>
</html>
```

The results of Code 4.5. The text in red matches the general sibling criteria—in this case the emphasis tags within the second, third, and fourth paragraphs.
Working with Pseudo-Classes

Many HTML elements have special states or uses associated with them that can be styled independently. One prime example of this is the link tag, `<a>`, which has link (its normal state), a visited state (when the visitor has already been to the page represented by the link), hover (when the visitor has the mouse over the link), and active (when the visitor clicks the link). All four of these states can be styled separately.

A pseudo-class is a predefined state or use of an element that can be styled independently of the default state of the element.

- **Links** *(Table 4.3)*—Pseudo-classes are used to style not only the initial appearance of the anchor tag, but also how it appears after it has been visited, while the visitor hovers the mouse over it, and when visitors are clicking it.

- **Dynamic** *(Table 4.3)*—Pseudo-classes can be applied to any element to define how it is styled when the user hovers over it, clicks it, or selects it.

- **Structural** *(Table 4.4)*—Pseudo-classes are similar to the sibling combinatory selectors but allow you to specifically style elements based on an exact or computed numeric position.

- **Other** *(Table 4.4)*—Pseudo-classes are available to style elements based on language or based on what tag they are **not**.
### TABLE 4.3 Link and Dynamic Pseudo-Classes

<table>
<thead>
<tr>
<th>Format</th>
<th>Selector Name</th>
<th>Elements Are Styled If…</th>
</tr>
</thead>
<tbody>
<tr>
<td>:link</td>
<td>Link</td>
<td>the value of href is not in history</td>
</tr>
<tr>
<td>:visited</td>
<td>Visited link</td>
<td>the value of href is in history</td>
</tr>
<tr>
<td>:target</td>
<td>Targeted link</td>
<td>a targeted anchor link</td>
</tr>
<tr>
<td>:active</td>
<td>Active</td>
<td>the element is clicked</td>
</tr>
<tr>
<td>:hover</td>
<td>Hover</td>
<td>the pointer is over the element</td>
</tr>
<tr>
<td>:focus</td>
<td>Focus</td>
<td>the element has screen focus</td>
</tr>
</tbody>
</table>

### TABLE 4.4 Structural/Other Pseudo-Classes

<table>
<thead>
<tr>
<th>Format</th>
<th>Selector Name</th>
<th>Elements Are Styled If…</th>
</tr>
</thead>
<tbody>
<tr>
<td>:root</td>
<td>Root</td>
<td>is the top level element in a document</td>
</tr>
<tr>
<td>:empty</td>
<td>Empty</td>
<td>has no children</td>
</tr>
<tr>
<td>:only-child</td>
<td>Only child</td>
<td>has no siblings</td>
</tr>
<tr>
<td>:only-of-type</td>
<td>Only of type</td>
<td>has its unique selector among its siblings</td>
</tr>
<tr>
<td>:first-child</td>
<td>First-child</td>
<td>is the first child of another element</td>
</tr>
<tr>
<td>:nth-of-type(n)</td>
<td>Nth of type</td>
<td>is the nth element with that selector</td>
</tr>
<tr>
<td>:nth-last-of-type(n)</td>
<td>Nth from last of type</td>
<td>is the nth element with that selector from the last element with that selector</td>
</tr>
<tr>
<td>:last-child</td>
<td>Last child</td>
<td>is the last child in the parent element</td>
</tr>
<tr>
<td>:first-of-type</td>
<td>First of type</td>
<td>is the first of its selector type in the parent element</td>
</tr>
<tr>
<td>:last-of-type</td>
<td>Last of type</td>
<td>is the last of its selector type in the parent element</td>
</tr>
<tr>
<td>:lang()</td>
<td>Language</td>
<td>has a specified language code defined</td>
</tr>
<tr>
<td>:not(s)</td>
<td>Negation</td>
<td>is not using specific selectors</td>
</tr>
</tbody>
</table>
Styling links

Although a link is a tag, its individual states are not. To set properties for these states, you must use the pseudo-classes associated with each state that a link can have (in this order):

- **:link** lets you declare the appearance of hypertext links that have not yet been selected.
- **:visited** lets you set the appearance of links that the visitor selected previously—that is, the URL of the `href` attribute in the tag that is part of the browser’s history.
- **:hover** lets you set the appearance of the element when the visitor’s pointer is over it.
- **:active** sets the style of the element when it is clicked or selected by the visitor.

For ideas on which styles to use with links, see the sidebar “Picking Link Styles.”

**To set contrasting link appearances:**

1. Style the anchor tag.
   ```html
   a {...}
   ```

   Although not required, it’s best to first define the general anchor style (**Code 4.6**). This differs from setting the `:link` pseudo-class in that these styles are applied to all the link pseudo-classes. So, you want to declare any styles that will remain constant or are changed in only one of the states.

   *continues on page 86*
The link styles are set for the default and then all four link states, creating color differentiation. Notice also that I've turned off underlining with text decoration but added an underline effect using border bottom.
2. **Style the default link state.** Type the selector (anchor tag, class, or ID) of the element you want to style, followed by a colon (:), and then `link`.
   
   ```html
   a:link {...}
   ```
   
   You can override styles set for the anchor tag, but this rule should always come before the `:visited` pseudo-class.

3. **Style the visited link style.** Type the selector (anchor, class, or ID) of the element you want to style, followed by a colon (:), and then `visited`.
   
   ```html
   a:visited {...}
   ```

4. **Style the hover link state.** Type the selector (anchor, class, or ID) of the element you want to style, followed by a colon (:), and then `hover`.
   
   ```html
   a:hover {...}
   ```

5. **Style the active link state.** Type the selector (anchor, class, or ID) of the element you want to style, followed by a colon (:), and then `active`.
   
   ```html
   a:active {...}
   ```

6. **Style is applied to the link state as needed.**

   ```html
   <a href="AIWL-ch01.html">...</a>
   ```

   All links on the page will obey the rules you lay down here when styling the various link states. You can—and should—use selective styling to differentiate link types.

   In this example, the pseudo-classes are applied directly to the anchor tag, but any class or ID could have been used as long as it was then applied to an anchor tag.

---

**Picking Link Styles**

Most browsers default to blue for unvisited links and red or purple for visited links. The problem with using two different colors for visited and unvisited links is that visitors may not remember which color applies to which type of link. The colors you choose must distinguish links from other text on the screen and distinguish among the states (link, visited, hover, and active) without dominating the screen and becoming distracting.

I recommend using a color for unvisited links that contrasts with both the page’s background color and the text color. Then, for visited links, use a darker or lighter version of the same color that contrasts with the background but is dimmer than the unvisited link color. Brighter not-followed links will then stand out dramatically from the dimmer followed links.

For example, on a page with a white background and black text, I might use bright red for links (rgb(255,0,0)) and pale red (rgb(255,153,153)) for visited links. The brighter version stands out; the paler version is less distinctive, but still obviously a link.
You can apply the dynamic pseudo-classes :hover, :active, and :focus to any element, not just links.

The general anchor link styles will be inherited by the different states and between states. The font you set for the :link appearance, for example, will be inherited by the :active, :visited, and :hover states.

The Web is a hypertext medium, so it is important that users be able to distinguish among text, links, and visited links. Because users don’t always have their Underline Links option turned on, it’s a good idea to set the link appearance for every document.

If you use too many colors, your visitors may not be able to tell which words are links and which are not.

The link styles are set for the entire page in this example, but links can be used for a variety of purposes. For example, links might be used for global navigation, in a list of article titles, or even as a dynamic control. To that end, it’s a good idea to style links depending on their usage:

```css
nav a { ... }
nav a:link { ... }
nav a:visited { ... }
```

The preceding styles would be applied only to links in the navigation element.
Styling for interaction

Once loaded, Web pages are far from static. Users will start interacting with the page right away, moving their pointers across the screen and clicking hither and yon. The dynamic pseudo-classes allow you to style elements as the user interacts with them, providing visual feedback:

- **:hover**—Same as for links, but sets the appearance of the element when the pointer is hovering over it.
- **:focus**—Applied to elements that can receive focus, such as form text fields.
- **:active**—Same as for links, but sets the style of the element when it is clicked or selected.

To define a dynamic pseudo-class:

1. **Style the default element.**
   
   ```html
   input {...}
   ```
   
   Although optional, it’s generally a good idea to set the default, non-dynamic style for the elements receiving dynamic styles (Code 4.7).

2. **Style the hover state of the element.**
   
   Type the selector (HTML, class, or ID), a colon (:), and then **hover**.
   
   ```html
   input:hover {...}
   ```
   
   As soon as the pointer enters the element’s box (see Chapter 10 for details about the box model), the style change will occur.

Code 4.7 The input elements, with a special style for the button type, are set to change style when the user interacts with them by hovering, selecting (focus), or clicking (active).

```html
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<title>Alice’s Adventures in Wonderland</title>
<style type="text/css" media="all">
input {
  border: 3px solid gray;
  background-color: silver;
  color: gray;
  padding: 0 5px;
  font-size: 1.5em; }
input[type="button"] {
  border-radius: 1em;
  color: silver;
  background-color: gray; }
input:hover {
  background-color: white;
  border-color: pink;
  color: silver; }
input:focus {
  border-color: red;
  background-color: white;
  color: black;
  outline: none; }
input:active {
  color: red;
  border-color: pink;
  background-color: silver; }
</style>
</head>
<body>
<footer>
<label>Mailing List:</label>
<input type="text" value="email" --> placeholder="enter your eMail">
<input type="button" class="active" --> value="submit">
</footer>
</body>
</html>
```
3. **Style the focus state of the element.**
   Type the selector (HTML, class, or ID), a colon (:), and then **focus**.
   ```css
   input:focus {...}
   ```
   As soon as the element receives focus (is clicked or tabbed to), the style change occurs and then reverts to the hover or default style when the element loses focus (called **blur**).

4. **Style the active state of the element.**
   Type the selector (HTML, class, or ID), a colon (:), and then **active**.
   ```css
   input:active {...}
   ```
   As soon as the user clicks within the element's box (explained in Chapter 10), the style change will occur and then revert to either the hover or default style when released.

5. **The styles are applied to the elements’ states as necessary in reaction to the user.**
   ```html
   <input type="button" value="Submit">
   ```
   All the tags using the specific selector will have their states styled.

---

**Tip** I recommend caution when changing some attributes for :hover. Changing typeface, font size, weight, and other properties may make the text grow larger or smaller than the space reserved for it in the layout and force the whole page to reflow its content, which can really annoy visitors.

**Tip** In this example, input is used to show the dynamic states. The input has one styling drawback in that all input types use the same tag. Later in this chapter, you will see how to use tag attributes to set styles, which will allow you to set different styles for text fields and buttons.

---

**Tip** The order in which you define your link and dynamic pseudo-classes makes a difference. For example, placing the :hover pseudo-class before the :visited pseudo-class keeps :hover from working after a link has been visited. For best results, define your styles in this order: link, visited, hover, focus, and active.

**Tip** One way to remember the pseudo-element order is the meme LoVe HAte: Link Visited Hover Active.

**Tip** You will want to always set :focus if you’re setting :hover. Why? Hover is applied only to non-keyboard (mouse) interactions with the element. For keyboard-only Web users, :focus will apply.
Styling specific children with pseudo-classes

Designers often want to apply a style to an element that is the first element to appear within another element, such as a parent’s first child.

The first-child pseudo-element has been available since CSS2; however, CSS3 offers an assortment of new structural pseudo-elements for styling an element’s child element exactly (Table 4.4):

- :first-child—Sets the appearance of the first instance of a selector type if it is the first child of its parent.
- :first-of-type—Sets the appearance of an element the first time its selector type appears within the parent.
- :nth-child(#)—Sets the appearance of the specific occurrence of the specified child element. For example, the third child element of a paragraph would be p:nth-child(3).
- :nth-of-type(#)—Sets the appearance of the specific occurrence of a selector type within the parent. For example, the seventh paragraph would be p:nth-of-type(7).
- :nth-last-of-type(#)—Sets the appearance of the specific occurrence of a selector type within the parent, but from the bottom. For example, the third paragraph from the bottom would be p:nth-last-of-type(3).
- :last-child—Sets the appearance of the element of the indicated selector type if it is the last child of the parent.
- :last-of-type—Sets the appearance of the last instance of a particular selector type within its parent.

Text Decoration: To Underline or Not

Underlining is the standard way of indicating a hypertext link on the Web. However, the presence of many underlined links turns a page into an impenetrable mass of lines, and the text becomes difficult to read. In addition, if visitors have underlining turned off, they cannot see the links, especially if the link and text colors are the same.

CSS allows you to turn off underlining for links, overriding the visitor’s preference. I recommend this practice and prefer to rely on clear color choices to highlight hypertext links or to rely on the alternative underlining method of border-bottom, which allows you better control over the style of the underline. See Chapter 14 for more information.
To style the children of an element:

1. Style the children based on their positions in the parent. Type the selector (HTML, class, or ID) of the element you want to style, a colon (:), and one of the structural pseudo-elements from Table 4.4 (Code 4.8).

   - li:first-child {...}
   - li:first-of-type {...}
   - li:nth-child(n) {...}
   - li:nth-of-type(n) {...}
   - li:last-child {...}
   - li:last-of-type {...}

2. Elements will be styled if they match the pattern.

```html
<li>...</li>
```

Set up your HTML with the selectors from Step 1 in mind.

```html
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<title>Alice's Adventures in Wonderland</title>
<style type="text/css" media="all">
li {
  font-size: 1.5em;
  margin: .25em; }
li:first-child { color: red; }
li:first-of-type { border-bottom: 1px solid orange; }
li:nth-child(2) { color: yellow; }
li:nth-of-type(6) { color: green; }
li:nth-last-of-type(2) { color: blue; }
li:last-of-type { border-bottom: 1px solid indigo; }
li:last-child { color: violet; }
</style>
</head>
<body>
<nav>
<ol>
<li>Down the Rabbit-hole</li>
<li>The Pool of Tears</li>
<li>A Caucus-race and a Long Tale</li>
<li>The Rabbit sends in a Little Bill</li>
<li>Advice from a Caterpillar</li>
<li>Pig and Pepper</li>
<li>The Queen's Croquet-ground</li>
<li>The Mock Turtle's Story</li>
<li>The Lobster Quadrille</li>
<li>Who Stole the Tarts?</li>
<li>Alice's Evidence</li>
</ol>
</nav>
</body>
</html>
```

The results of Code 4.8 show the items in the list styled separately. In this case, the first child and first of type are the same element as the last element and last of type.
Styling for a particular language

The World Wide Web is just that—all around the world—which means that anyone, anywhere can see your pages. It also means that Web pages are created in many languages.

The :lang() pseudo-class lets you specify styles that depend on the language specified by the language property.

To set a style for a specific language:

1. **Style an element based on its language code.** Type the selector (HTML, class, or ID) of the element you want to style, a colon (:), `lang`, and enter the letter code for the language you are defining within parentheses (Code 4.9).

   ```html
   p:lang(fr) {...}
   ```

2. **The element is styled if it has a matching language code.** Set up your tag in the HTML with the language attributes as necessary.

   ```html
   <p lang="fr">...</p>
   ```

   If the indicated selector has its language attribute equal to the same value that you indicated in parentheses in Step 1, the style is applied.

   You can use any string as the language letter code, as long as it matches the value in the HTML. However, the W3C recommends using the codes from RFC 3066 or its successor. For more on language tags, visit [www.w3.org/International/articles/language-tags](http://www.w3.org/International/articles/language-tags).

**Tip** Language styles can go far beyond simple colors and fonts. Many languages have specific symbols for quotes and punctuation, which CSS can add. In Chapter 9, you will find information on how to style quotes for a particular language.
Code 4.9 Styles are set to turn paragraphs red if they are in French (fr).

```html
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<title>Alice’s Adventures in Wonderland</title>
<style type="text/css" media="all">
q:lang(fr) {
  quotes: '«' '»';
  color: red; }
</style>
</head>
<body>
<article class="chaptext">
<p>Alice was beginning to get very tired of sitting by her sister on the bank, and of having nothing to do: once or twice she had peeped into the book her sister was reading, but it had no pictures or conversations in it, <q>and what is the use of a book,</q> thought Alice, <q>without pictures or conversations?</q></p>
<p class="translation" lang="fr">Alice commençait à être très fatigué d’être assis par sa sœur sur la rive, et de n’avoir rien à faire: une fois ou deux, elle avait regardé dans le livre de sa sœur – lisait, mais il n’avait pas d’images ni dialogues en elle, <q>et ce qui est l’utilisation d’un livre?</q> pensait Alice, <q>sans images ni dialogues?</q></p>
</article>
</body>
</html>
```

The results of Code 4.9 show the paragraph in French rendered in red (with my apologies to French speakers).
Not styling an element

So far you’ve looked at ways to style a tag if it is something. The negation selector, :not, allows you to not style something for a particular selector.

To not set a style for a particular element:

1. Style elements to exclude certain selectors. Type the selector (HTML, class, or ID) of the element you want to style, a colon (:), not, and enter the selectors you want excluded from this rule in parentheses (Code 4.10).

   ```
   p:not(.dialog) {...}
   ```

2. The element is not styled if it contains the indicated selector.

   ```
   <p class='dialog'>...</p>
   → <p>...</p>
   ```

   The styles are applied to elements that match the initial selector but not the selector in parentheses.
Code 4.10 When the element is a paragraph that does not use the dialog class, it will be displayed in red and italics.

```html
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<title>Alice's Adventures in Wonderland</title>
<style type="text/css" media="all">
q:lang(fr) {
  quotes: ' «' ' » ' ;
}
p:not(.translation) {
  color: red;
}
</style>
</head>
<body>
<article class="chaptertext">
<p>Alice was beginning to get very tired of sitting by her sister on the bank, and of having nothing to do: once or twice she had peeped into the book her sister was reading, but it had no pictures or conversations in it, <q>and what is the use of a book, </q> thought Alice, <q>without pictures or conversations?</q></p>
<p class="translation" lang="fr">Alice commençait à être très fatigué d'être assis par sa sœur sur la rive, et de n'avoir rien à faire: une fois ou deux, elle avait regardé dans le livre de sa sœur lisait, mais il n'avait pas d'images ni dialogues en elle, <q>et ce qui est l'utilisation d'un livre, </q> pensait Alice, <q>sans images ni dialogues?</q></p>
</article>
</body>
</html>
```

The results of Code 4.10. This shows that the paragraph that does use the dialog class does not receive the style.
Working with Pseudo-Elements

A pseudo-element is a specific, unique part of an element—such as the first letter or first line of a paragraph—that can be styled independently of the rest of the element. (For a list of other pseudo-elements, see Table 4.5.)

Working with first letters and lines

You can access the first letter of any block of text directly using the :first-letter pseudo-element. The first line of any block of text can be isolated for style treatment using the :first-line pseudo-element A.

To highlight the beginning of an article:

1. Style the default version of the element.

   article p {...}

   Although not required, it’s generally a good idea to set the default style of the selector for which you will be styling the :first-letter pseudo-element (Code 4.11).

### TABLE 4.5 Pseudo-Elements

<table>
<thead>
<tr>
<th>Format</th>
<th>Selector Name</th>
<th>Elements Are Styled If…</th>
</tr>
</thead>
<tbody>
<tr>
<td>:first-letter, ::first-letter</td>
<td>the first letter</td>
<td>first letter in text</td>
</tr>
<tr>
<td>:first-line, ::first-line</td>
<td>the first line of text</td>
<td>they are the first line of text</td>
</tr>
<tr>
<td>:after, ::after</td>
<td>After</td>
<td>space immediately before element</td>
</tr>
<tr>
<td>:before, ::before</td>
<td>Before</td>
<td>space immediately after element</td>
</tr>
</tbody>
</table>

A The general syntax for pseudo-elements. Pseudo-elements can have either a single or double colon, but use a single colon at present for increased browser compatibility.

Chapter I Down the Rabbit-Hole

Alice was beginning to get very tired of sitting by her sister on the bank, and of having nothing to do: once or twice she had peeped into the book her sister was reading, but it had no pictures or conversations in it, "and what is the use of a book," thought Alice, "without pictures or conversations?"

A The results of Code 4.11. A common typographic trick to draw the reader’s eye to the beginning of a paragraph is to use a drop cap and to bold the first line of text, as shown here.
2. Style the first letter of the element if it is the first of its type. Type the selector you want to style the first letter of (article p), a colon (:) and then first-letter.

```
article p:first-of-type::first-letter {...}
```

To affect only the first paragraph in an article, you can add the :first-of-type pseudo-class, as in this example.

```
article p:first-of-type::

→ first-letter {...}
```

3. Style the first line of the element’s text if it is the first of its type. Type the selector (article p) for which you want to style the first letter, a colon (:) and then first-line.

```
article p:first-of-type::first-line {

→ first-line {...}
```

In this example, the first-of-type pseudo-class is added so that only the first paragraph in an article is styled.

4. The element’s first letter and first line of text is styled if it is the first of its type in the parent element. Add the class attribute to the relevant HTML tag.

```
<p>...</p>
```

Although you do not have to use a class, you generally will want to selectively style the first letter of elements rather than style them all universally.

**Tip** Drop-cap styled letters are a time-honored way to start a new section or chapter by making the first letter of a paragraph larger than subsequent letters and moving several lines of text to accommodate the larger letter. Medieval monks used drop caps with illuminated manuscripts. Now you can use them on the Web.
Setting content before and after an element

The :before and :after pseudo-elements can be used to generate content that appears above or below a selector. Generally, these pseudo-classes are used with the content property. (See “Adding Content Using CSS” in Chapter 9.) The pseudo-elements let you add and style repetitive content to the page in a consistent way.

To set content before and after an element:

1. Style the element.
   
   h2 {...

   Although not required, it’s generally a good idea to set the default style of the selector for which you will be styling the :before and :after pseudo-elements. (See Code 4.12.)

2. Add content before the element. Type the selector (HTML, class, or ID) you want to add content before, a colon (:), and then the keyword before.

   h2:before { content:... }

   Next, declare the content property and define what generated content goes before the element and how it should be styled.

   Code 4.12 Before and after pseudo-elements are used to add content—images in this case—to the page header.

   ```html
   <!DOCTYPE html>
   <html lang="en">
   <head>
   <meta charset="UTF-8">
   <title>Alice’s Adventures in Wonderland</title>
   <style type="text/css" media="all">
   h2 {
     font-size: 2em;
     color: red;
   }
   h2:before {
     content: url('bullet-01.png');
   }
   h2:after {
     content: url('bullet-02.png');
   }
   </style>
   </head>
   <body>
   <article class="chaptertext">
   <h2>Chapter I</h2>
   <span class="chaptertitle">Down the Rabbit-Hole</span>
   <p>Alice was beginning to get very tired of sitting by her sister on the bank, and of having nothing to do: once or twice she had peeped into the book her sister was reading, but it had no pictures or conversations in it, thought Alice, without pictures or conversations!</p>
   </article>
   </body>
   </html>
   ```
3. Add content after the element. Type the selector (HTML, class, or ID) you want to add content after, a colon (:) and then the keyword `after`.

```css
h2:after { content: ...; }
```

Next, declare the `content` property and define what generated content goes after the element and how it should be styled.

The pseudo-element syntax in CSS3 has undergone a slight change from the CSS2 syntax (which is rare). Pseudo-elements now have a double colon to distinguish them from pseudo-classes. Existing pseudo-elements can use either single or double colons. New and future pseudo-elements should use double colons, but will work with a single colon.

**Tip** Since IE8 does not support double colon syntax for CSS2 pseudo-elements, it’s a good idea to use single colon syntax for older pseudo-elements until all browsers have adopted the syntax. Double colon will not work in IE8 anyway.

**Tip** Be careful when using `before` and `after` to add content to your page. This content will not appear to search engines or screen readers, so do not rely on it for anything vital.
Defining Styles Based on Tag Attributes

Although style attributes should all be handled by CSS, many HTML tags still have attributes that define how they behave. For example, the image tag, `img`, always includes the `src` attribute to define the source for the image file to be loaded.

Styles can be assigned to an HTML element based on an attribute or an attribute value, allowing you to set styles if the attribute has been set, is or is not a specific value, or contains a specific value (Table 4.6).

To set styles based on an element’s attributes:

1. Set styles if the element has a specific property. To set styles based on the existence of an attribute, type the selector you want to style (HTML, class, or ID), a left bracket ([), the name of the attribute you want to check for, and a right bracket (]) (Code 4.13) A.

   ```html
   a[title] {...}
   ```

   ```css
   a[title=""Author"] { color: red; }
   ```

   A The general syntax of an attribute selector.

   B The results of Code 4.13. This shows how styles are applied to elements based on their properties.

   ### Table 4.6 Attribute Selectors

<table>
<thead>
<tr>
<th>Format</th>
<th>Name</th>
<th>Elements Are Styled If That Element:</th>
</tr>
</thead>
<tbody>
<tr>
<td>[attr]</td>
<td>Attribute</td>
<td>has specified attribute</td>
</tr>
<tr>
<td>[attr=&quot;value&quot;]</td>
<td>Exact value</td>
<td>has specified attribute equal to exact value</td>
</tr>
<tr>
<td>[attr~=&quot;value&quot;]</td>
<td>Spaced list</td>
<td>has specified attribute equal to exact value within space-separated list</td>
</tr>
<tr>
<td>[attr^=&quot;value&quot;]</td>
<td>Hyphenated list</td>
<td>has specified attribute equal to exact value within hyphen-separated list</td>
</tr>
<tr>
<td>[attr$=&quot;value&quot;]</td>
<td>Begins with</td>
<td>has specified attribute equal to exact value at beginning</td>
</tr>
<tr>
<td>[attr*=&quot;value&quot;]</td>
<td>Ends with</td>
<td>has specified attribute equal to exact value at end</td>
</tr>
<tr>
<td>[attr=&quot;value&quot;]</td>
<td>Contains</td>
<td>has specified attribute equal to exact value anywhere</td>
</tr>
</tbody>
</table>

---

100 Chapter 4
This will assign the styles you declare only if the tag has this attribute assigned to it regardless of the value.

2. **Set styles if a string exactly matches the property's value.** To set styles based on an attribute's exact value, type the selector you want to style (HTML, class, or ID), a left bracket ([], the name of the attribute you want to check for, an equals sign (=), the value you are testing for in quotes ("..."), and a right bracket (]). The value is case sensitive.

```html
a[title='home'] {...}
```

This will assign the styles you declare only if the tag has this attribute assigned to it with the exact assigned value.

3. **Set styles if a string is in a space-separated list of values.** To set styles based on an attribute’s value that is within a list of space-separated values (for example, a particular word in a sentence), type the selector you want to style (HTML, class, or ID), a left bracket ([), the name of the attribute you want to check for, a tilde (~), an equals sign (=), the value you are testing for in quotes ("..."), and a right bracket (]).

```html
a[title~="email"] {...}
```

This will assign the styles you declare only if the tag has the attribute assigned to it with a value that contains the string as part of a space-separated list. Generally, this means that it is a word in a sentence. Partial words do not count. So in this example, testing for 'mail' would not work.

*continues on next page*
4. **Sets the style if the string is in a hyphenated list of values assigned to the property.** To set styles based on an attribute’s value being the first in a list separated by hyphens, type the selector you want to style (HTML, class, or ID), a left bracket ([), the name of the attribute you want to check for, a bar (|), an equals sign (=), the value you are testing for in quotes ("..."), and a right bracket (]).

   `a[title|="resume"]`

   This will assign the styles you declare only if the tag has this attribute assigned to it with a value that contains the string at the beginning of a hyphen-separated list. Generally, this is used for styling languages as an alternative to using the language pseudo-class.

5. ★**Set styles if a string is the value’s prefix.** To set styles based on the value at the beginning of an attribute, type the selector you want to style (HTML, class, or ID), a left bracket ([), the name of the attribute you want to check for, a carat (^), an equals sign (=), the value you are testing for in quotes ("..."), and a right bracket (]).

   `a[href^="http://"]`

   This will assign the styles you declare only if the value string occurs exactly as it is in the quotes at the beginning of the attribute value.
6. ★Set styles if a string is the property value’s suffix. To set styles based on an attribute’s value being the first in a hyphen-separated list, type the selector you want to style (HTML, class, or ID), a left bracket ([), the name of the attribute you want to check for, a dollar sign ($), an equals sign (=), the value you are testing for in quotes (‘...’), and a right bracket (]).

   a[href$=".info"]

   This will assign the styles you declare only if the value occurs at the end of the attribute’s value.

7. Set styles if a string is anywhere in the property value. To set styles based on an attribute’s value being the first in a hyphen-separated list, type the selector you want to style (HTML, class, or ID), a left bracket ([), the name of the attribute you want to check for, an asterisk (*), an equals sign (=), the value you are testing for in quotes (‘...’), and a right bracket (]).

   a[href*="speakinginstyles"]

   This will assign the styles you declare if the value occurs anywhere in the attribute’s value.

Values are case sensitive. In other words, “Alice” and “alice” are two different values.
Querying the Media

In Chapter 3 you learned how to specify style sheets for a particular media type, allowing you to set styles depending on whether the HTML is output to a screen, print, TV, or a handheld or other device (Table 4.7). CSS3 adds an important new capability that allows you to set styles based on common interface properties such as width, height, aspect ratio, and number of available colors.

Media queries and the @media rule can be used to tailor your page, not just to a general device type but to the specific device your site visitor is using. This includes sizing for print, for mobile devices, or to best fit the size of the open browser window.

Media queries

If you want to know the current size of the browser window, why not just ask the browser? JavaScript gives you the ability to do this, but it’s a cumbersome way to get some basic facts about the Webbed environment your design is trying to fit into.

Media queries provide you with several common media properties that you can test and then deliver the style sheet that best suits the environment.

Although media queries have many properties (Table 4.8), they come in five basic flavors:

- **Aspect-ratio** looks for the relative dimensions of the device expressed as a ratio: 16:9, for example.
- **Width** and **height** looks for the dimensions of the display area. These can also be expressed as maximum and minimum values.

<table>
<thead>
<tr>
<th>Value</th>
<th>Intended for</th>
</tr>
</thead>
<tbody>
<tr>
<td>screen</td>
<td>Computer displays</td>
</tr>
<tr>
<td>tty</td>
<td>Teletypes, computer terminals, and older portable devices</td>
</tr>
<tr>
<td>tv</td>
<td>Television displays</td>
</tr>
<tr>
<td>projection</td>
<td>Projectors</td>
</tr>
<tr>
<td>handheld</td>
<td>Portable phones and PDAs</td>
</tr>
<tr>
<td>print</td>
<td>Paper</td>
</tr>
<tr>
<td>braille</td>
<td>Braille tactile readers</td>
</tr>
<tr>
<td>speech</td>
<td>Speech synthesizers</td>
</tr>
<tr>
<td>all</td>
<td>All devices</td>
</tr>
</tbody>
</table>

A The general syntax for media queries.

continues on page 106
<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>aspect-ratio</td>
<td>&lt;ratio&gt;</td>
</tr>
<tr>
<td>max-aspect-ratio</td>
<td>&lt;ratio&gt;</td>
</tr>
<tr>
<td>min-aspect-ratio</td>
<td>&lt;ratio&gt;</td>
</tr>
<tr>
<td>device-aspect-ratio</td>
<td>&lt;ratio&gt;</td>
</tr>
<tr>
<td>max-device-aspect-ratio</td>
<td>&lt;ratio&gt;</td>
</tr>
<tr>
<td>min-device-aspect-ratio</td>
<td>&lt;ratio&gt;</td>
</tr>
<tr>
<td>color</td>
<td>&lt;integer&gt;</td>
</tr>
<tr>
<td>max-color</td>
<td>&lt;integer&gt;</td>
</tr>
<tr>
<td>min-color</td>
<td>&lt;integer&gt;</td>
</tr>
<tr>
<td>color-index</td>
<td>&lt;integer&gt;</td>
</tr>
<tr>
<td>max-color-index</td>
<td>&lt;integer&gt;</td>
</tr>
<tr>
<td>min-color-index</td>
<td>&lt;integer&gt;</td>
</tr>
<tr>
<td>device-height</td>
<td>&lt;length&gt;</td>
</tr>
<tr>
<td>max-device-height</td>
<td>&lt;length&gt;</td>
</tr>
<tr>
<td>min-device-height</td>
<td>&lt;length&gt;</td>
</tr>
<tr>
<td>device-width</td>
<td>&lt;length&gt;</td>
</tr>
<tr>
<td>max-device-width</td>
<td>&lt;length&gt;</td>
</tr>
<tr>
<td>min-device-width</td>
<td>&lt;length&gt;</td>
</tr>
<tr>
<td>height</td>
<td>&lt;length&gt;</td>
</tr>
<tr>
<td>max-height</td>
<td>&lt;length&gt;</td>
</tr>
<tr>
<td>min-height</td>
<td>&lt;length&gt;</td>
</tr>
<tr>
<td>monochrome</td>
<td>&lt;integer&gt;</td>
</tr>
<tr>
<td>max-monochrome</td>
<td>&lt;integer&gt;</td>
</tr>
<tr>
<td>min-monochrome</td>
<td>&lt;integer&gt;</td>
</tr>
<tr>
<td>orientation</td>
<td>portrait, landscape</td>
</tr>
<tr>
<td>resolution</td>
<td>&lt;resolution&gt;</td>
</tr>
<tr>
<td>max-resolution</td>
<td>&lt;resolution&gt;</td>
</tr>
<tr>
<td>min-resolution</td>
<td>&lt;resolution&gt;</td>
</tr>
<tr>
<td>scan</td>
<td>progressive, interlaced</td>
</tr>
<tr>
<td>width</td>
<td>&lt;length&gt;</td>
</tr>
<tr>
<td>max-width</td>
<td>&lt;length&gt;</td>
</tr>
<tr>
<td>min-width</td>
<td>&lt;length&gt;</td>
</tr>
</tbody>
</table>
- **Orientation** looks for landscape (height greater than width) or portrait (width greater than height) layout. This allows you to tailor designs for devices that can flip.

- **Color, color-index, and monochrome** finds the number of colors or bits per color. These allow you to tailor your design for black-and-white mobile devices.

- **Resolution** looks at the density of pixels in the output. This is especially useful when you want to take advantage of display devices that have a higher resolution than 72 dpi.

By default, media queries are for the viewport (see Chapter 11 for details on the viewport) with the exception of those that specify `device`, in which case they are for the entire screen or output area. For example, width is the width of the visible browser viewport within the screen, whereas device-width is the width of the entire screen.
Using media queries to specify styles:

1. Create your style sheets. Create a default media style sheet that captures all the general styles for your design and save it. I like to call mine default.css (Code 4.14).

Create style sheets for the various media or specific devices for which you will be designing. Print is generally good to include (Code 4.17). You can call the sheet print.css, but you might also want to create style sheets specifically for tablets (Code 4.15) and for desktop computers (Code 4.16).

---

**Code 4.14 default.css**—These styles are applied regardless of the screen size; but we are tailoring the styles for small devices, most likely mobile devices such as smart phones. We start with the small sizes first, and then tailor for larger sizes in the next two CSS files.

```html
/***	Default	Screen	Styles	***/
body {
    color: charcoal;
    font: normal 1.5em/1 helvetica, arial, sans-serif;
    background: silver url('alice23c.gif') no-repeat center 0;
    padding: 120px 20px;
}

h1 { color: purple; font-size: 1.5em; }

h2 { color: black; font-size: 1.25em; }

p { line-height: 2; font-size: 1em; }
```

**Code 4.15 medium.css**—A custom view for medium-size screens. Generally, these styles will be used by tablet devices.

```html
/***	Medium	Device	Styles	***/
body {
    color: dimgray;
    background-color: gray;
    font-size: 1.25em;
    padding: 200px 2em;
}

h1 { color: gold; }

h2 { color: silver; }
```

**Code 4.16 large.css**—The final style sheet will be used to serve a page tailored to larger computer screens.

```html
/***	Large	Device	Styles	***/
body {
    color: silver;
    font: normal 1.1em/2 georgia,times,serif;
    background: black url('alice23b.gif') no-repeat 0 0;
    padding: 200px 175px;
}

h1 {
    color: red;
    font-style: italic;
}

h2 { color: gray; }
```

**Code 4.17 print.css**—These styles are tailored for the printed page, changing the background to white (assuming white paper), serif fonts, black text, and a different background image.

```html
/***/
body {
    color: rgb(0,0,0);
    background: white url('alice23a.gif') no-repeat 0 0;
    padding: 200px 0 0 175px;
}

h1 {
    color: gray;
}

p { 
    font: normal 12pt/2 Constantia, palatino, times, "times new roman", serif;
}
```
2. **Add the viewport meta tag.** In the head of your HTML document (Code 4.18), add a meta tag with a name equal to viewport and content, as shown.

```html
<meta name="viewport"
→ content="width=device-width,
→ initial-scale=1, maximum-
→ scale=1, minimum-scale=1,
→ user-scalable=no" />
```

This will prevent devices with smaller screens, most notably the iPhone, from resizing the page, overriding your styles to be set in Step 5.

3. **Link to your default style sheet.** In the head of your HTML document (Code 4.18), type a `<link>` tag that references the default version of the CSS and define `media` as `all`.

```html
<link rel="stylesheet" media="all"
→ href="default.css" />
```

---

*B Code 4.18 output to a computer screen. This version uses a dark background and an inverted version of the Alice’s Adventures in Wonderland illustration. On an LCD screen, the lightly colored text will look fine.*

*C Code 4.18 on a tablet device, in this case an iPad.*
Code 4.18 The HTML code links to all three of the style sheets, which are displayed in default 📦, tablet 📱, smart phone 📱, and print 📜. The iPhone style sheet uses media queries to set a device’s width range in keeping with the iPhone. Notice that I used screen for the media type because the iPhone identifies itself as a screen, not a handheld device.

```html
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1, maximum-scale=1, minimum-scale=1, user-scalable=no" />
  <title>Alice’s Adventures in Wonderland</title>
  <link rel="stylesheet" media="screen" href="14.css">
  <link rel="stylesheet" media="screen and (min-width: 740px) and (min-device-width: 740px), (max-device-width: 800px) and (min-width: 740px) and (orientation:landscape)" href="15.css">
  <link rel="stylesheet" media="screen and (min-width: 980px) and (min-device-width: 980px)" href="16.css">
  <link rel="stylesheet" media="print" href="17.css">
</head>
<body>
<hgroup>
  <h1>Alice’s Adventures in Wonderland</h1>
  <h2 id="ch01">Chapter 1 <span class="chaptitile">Down the Rabbit-Hole</span></h2>
</hgroup>
<article>
  Alice was beginning to get very tired of sitting by her sister on the bank, and of having nothing to do: once or twice she had peeped into the book her sister was reading, but it had no pictures or conversations in it, "and what is the use of a book," thought Alice, "without pictures or conversations?"
</article>
<footer><nav>Next: <a class="chaptitile" href="AAIW-ch02.html">The Pool of Tears</a></nav></footer>
</body>
</html>
```

![Code 4.18 on a mobile device, in this case an iPhone. A specially tailored version to fit the width of smaller devices uses a custom header of the Cheshire cat.](image1)

![Code 4.18 output to a printer. The background is white, and the background image is no longer inverted. This works better in print.](image2)
4. **Use a media query to link to a style sheet.** Immediately after the previous `<link>` tag, add more `<link>` tags that reference the style sheets for a specific media type and then add media queries (Table 4.8) in parentheses connecting multiple queries with **and**.

```html
<style type="text/css" media="screen and (min-width: 740px) and (min-device-width: 740px), (max-device-width: 800px) and (min-width: 740px) and (orientation:landscape)">
  @import url("css/medium.css");
</style>

<style type="text/css" media="screen and (min-width: 980px) and (min-device-width: 980px)">
  @import url("css/medium.css");
  @import url("css/large.css");
</style>
```
5. Link to your *print* style sheet. Immediately after the `<link>` tag, add another `<link>` tag that references the print version of the CSS and define `media` as `print`.

```html
<link rel="stylesheet" media="print" href="print.css">
```

**Tip** Before media queries were introduced, Web developers used JavaScript to detect browser dimensions and colors. Media queries render those techniques obsolete, at least for styling purposes.

**Tip** In this example, media queries are applied to the media property value of the `<link>` tag, but you can just as easily apply them to the media property of the `<style>` tag.
Using the @media rule

Media queries allow you specify styles in the media property of `<link>` and `<style>` tags, but the `@media` rule allows you to embed media queries directly into a style sheet.

Using @media to specify styles:

1. Create your style sheets. Create an external style sheet or embed a style sheet in the body of your document (Code 4.19).

2. Use the `@media` rule to specify styles with media queries. In the head of your HTML document, type `@` and `media`. Then specify the media type (Table 4.7) and any media queries (Table 4.8) for the styles.

   ```css
   @media screen and (max-device-width: 480px) {
   ...}
   ```

   For example, you might specify that these styles are for screens with a width up to 480px wide. Finish with curly brackets. Add any media-specific styles between the curly brackets.

3. Add other styles as necessary.

   ```css
   body {...}
   ```

   You can add more `@media` rules or other nonmedia-specific rules. However, all CSS rules that are not in `@rules` (`@media`, `@font-face`, `@import`, and so on) must come after the `@rules`.

   **Tip** Remember that `@media` rules can go in external or embedded style sheets.
Code 4.19 The HTML code links to the various style sheets for different media types G and H.

```html
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1, maximum-scale=1, minimum-scale=1, user-scalable=no" />
<title>Alice’s Adventures in Wonderland</title>
<style type="text/css">
body {
  font: normal 12pt/2 "times new roman", serif;
  background: white url('alice23a.gif') no-repeat 0 0;
  padding: 200px 175px;
}

h1 { color: gray; }

h2 { color: silver; }

p { color: black; }

@media screen and (max-width: 480px) {
  /*** Small screen Styles ***/
  body {
    -webkit-text-size-adjust:none;
    color: red;
    background: gray url('alice23c.gif') no-repeat center 0;
    padding: 120px 20px 20px 20px;
  }

  h1 {
    color: red;
    text-shadow: 0 0 5px black;
  }

  h2 { color: silver; }

  p {
    font-size: 1.5em;
    color: white;
  }
}
</style>
</head>
<body>
<hgroup>
<h1>Alice’s Adventures In Wonderland</h1>
<h2 id="ch01">Chapter 1 <span class="chaptertitle">Down the Rabbit-Hole</span></h2>
</hgroup>
<article>
<p>Alice was beginning to get very tired of sitting by her sister on the bank, and of having nothing to do: once or twice she had peeped into the book her sister was reading, but it had had no pictures or conversations in it, <q>and what is the use of a book, thought Alice, q>without pictures or conversations?"</p>
</article>
<footer><nav>Next:
<a class="chaptertitle" href="AALW/ch02.html">The Pool of Tears</a></nav></footer>
</body>
</html>
```
Styling for Print

With the advent of laser and inkjet printers, we seem to be buried under mounds of perfectly printed paper. Even the Web seems to have increased the amount of paper we use. If an article on the Web is longer than a couple of scrolls, many people print it.

But the Web was created to display information on the screen, not on paper. Web graphics look blocky when printed, and straight HTML lacks much in the way of layout controls. That said, you can take steps to improve the appearance of printed Web pages. Looking good in print and on the Web may take a little extra effort, but your audience will thank you in the long run.

Here are six simple things you can do to improve the appearance of your Web page when it is printed:

- **Use page breaks before page headers** to keep them with their text.

- **Separate content from navigation.** Try to keep the main content—the part your audience is interested in reading—in a separate area of the design from the site navigation. You can then use CSS to hide navigation in the printed version with

```css
nav { display: none }
```

included in the print style sheet.

- **Avoid using transparent colors in graphics.** This is especially true if the graphic is on a background color or a graphic other than white. The transparent area of a GIF image usually prints as white regardless of the color behind it in the window. This situation is not a problem if the graphic is on a white background to begin with, but the result is messy if the graphic is supposed to be on a dark background.

- **Avoid using text in graphics.** The irony of printing content from the Web is that text in graphics, which may look smooth in the window, can look blocky when printed; but regular HTML text, which may look blocky on some PC screens, can print smoothly on any decent printer. Try to stick with HTML text as much as possible.

- **Avoid dark-colored backgrounds and light-colored text.** Generally you want to keep white as your background color for most of the printed page, and black or dark gray for the text.

- **Do not rely on color to convey your message when printed.** Although color printers are quite common these days, many people are still printing with black-and-white printers or printing in black and white on color printers to save money.
Inheriting Properties from a Parent

No, this book hasn’t suddenly become the Visual QuickStart Guide to Real Estate. Child and descendent HTML tags generally assume the styles of their parents—inherit them—whether the style is set using CSS or is inherited from a browser style. This is called inheritance of styles.

For example, if you set an ID called copy and give it a font-family value of Times, all of its descendents would inherit the Times font style. If you set a bold tag to red with CSS, all of its descendents will inherit both the applied red and the inherent bold style.

In some cases, a style property is not inherited from its parent—obvious properties such as margins, width, and borders. You will probably have no trouble figuring out

```css
strong { color: red; }
.chaptext { font-family: georgia; }
```

```html
<div class="chaptext">
  <strong>
    red (bold)
  </strong>
  (italics)
  <em>
    Curiouser and curiously!
  </em>
</div>
```

Results

Curiouser and curiously!

A The final result of the styles applied and inherited is bold, red, and italicized text in Times font. Styles in parentheses are inherent styles applied by the browser for the particular HTML tag.
which properties are inherited and which are not. For example, if you set a padding of four pixels for the paragraph tag, you would not expect bold tags within the paragraph to also add a padding of four pixels. If you have any doubts, see Appendix A, which lists all the CSS properties and how they are inherited.

If you did want to force an element to inherit a property of its parent, many CSS properties include the `inherit` value. So, in the previous example, to force all the bold tags in a paragraph to take on the 4px padding, you could set their `padding` value to `inherit`.

### Managing existing or inherited property values

When defining the styles for a selector, you do not cause it to lose any of its inherited or inherent attributes unless you specifically override those styles. All those properties are displayed unless you change the specific existing properties that make up its appearance.

In addition to overriding the relevant property with another value, many CSS properties have values that allow you to override inheritance:

- **inherit**—Forces a property to be inherited that would normally not be inherited, or overrides other applied style values and inherits the parent’s value.

- **none**—Hides a border, image, or other visual element.

- **normal**—Forces the default style to be applied.

- **auto**—Allows the browser to determine how the element should be displayed based on context.
You can add the `!important` declaration to a property-value declaration to give it the maximum weight when determining the cascade order. Doing so ensures that a declaration is applied regardless of the other rules in play. (See “Determining the Cascade Order” in this chapter.)

To force use of a declaration:

1. Add your CSS rule (Code 4.20).

   ```css
   h2 {
     color: green !important;
     font-size: 3em;
   }
   h2 {
     color: red;
     font-size: 2em;
   }
   ```

   You can use an HTML, class, or ID selector. CSS rules can be defined within the `<style>` tags in the head of your document (see “Embedded: Adding Styles to a Web Page” in Chapter 3) or in an external CSS file that is then imported or linked to the HTML document (see “External: Adding Styles to a Web Site” in Chapter 3).

2. Make it important. Type a style declaration, a space, `!important`, and a semicolon (`;`) to close the declaration.

   ```css
   color: green !important;
   ```

   continues on next page

---

A The general syntax for `!important`.

Code 4.20 The `!important` value has been added to the color property in the first `h2`, but not in the second B. Typically, the second `h2` would override the first, but not in this case.

```html
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Alice’s Adventures in Wonderland</title>
  <style type="text/css" media="all">
    h2 {
      color: green !important;
      font-size: 3em;
    }
    h2 {
      color: red;
      font-size: 2em;
    }
  </style>
</head>
<body>
<article class="chaptertext">
  <h2>Chapter I
    <span class="chaptertitle">Down the Rabbit-Hole</span>
  </h2>
  <p>Alice was beginning to get very tired of sitting by her sister on the bank, and of having nothing to do: once or twice she had peeped into the book her sister was reading, but it had no pictures or conversations in it, and what is the use of a book," thought Alice, "without pictures or conversations?"</p>
</article>
</body>
</html>

B The result of Code 4.20. The style that is most important wins the day, so the text is green rather than red, despite the fact that the red declaration comes later in the cascade.
3. Add other styles.

```css
font-size: 3em;
```

Add any other declarations you wish for this rule, making them `important` or not, as you desire.

`important` is a powerful tool, second only to inline styles for determining style cascade. `important` is great for debugging your CSS; but, because it can interfere with making changes later, it should never be used in the final Web site code.

Setting a shorthand property to `important` (background, for example) is the same as setting each sub-property (such as `background-color`) to be `important`.

**TIP** A common mistake is to locate `important` after the semicolon in the declaration. This causes the browser to ignore the declaration and, possibly, the entire rule.

**TIP** If you are debugging your style sheet and can’t get a particular style to work, try adding `important` to it. If it still doesn’t work, the problem is most likely a typo rather than another overriding style.

**TIP** Many browsers allow users to define their own style sheets for use by the browser. Most browsers follow the CSS 2.1 specification in which a user-defined style sheet overrides an author-defined style sheet.
Determining the Cascade Order

Within a single Web page, style sheets may be linked, imported, or embedded. Styles may also be declared inline in the HTML. In addition, many browsers allow visitors to have their own style sheets that can override yours. It’s guaranteed, of course, that simultaneous style sheets from two or more sources will have conflicting declarations. Who comes out on top?

The cascade order refers to the way styles begin at the top of the page and, as they cascade down, collect and replace each other as they are inherited. The general rule of thumb is that the last style defined is the one that is used.

However, at times, two or more styles will conflict. Use the following procedure to determine which style will come out on top and be applied to a given element.
To determine the cascade-order value for an element:
Collect all styles that will be applied to the element. Find all the inherent, applied, and inherited styles that will be applied to the element, and then use the following criteria to determine which styles are applied in the cascade order, with the criteria at the top being most important.

1. User styles
   Most Web browsers allow users to specify their own default style sheets. In principle, these always have precedence over other styles.

2. Inline styles
   If the style is inline (see Chapter 3), it is always applied regardless of all other factors. That's why you should never use them in your final HTML code.

3. Media type
   Obviously, if the media type is set for a style and element that is not being displayed in that media type, the style will not be used.

4. Importance
   Including !important with a declaration gives it top billing when displayed. (See “Making a Declaration !important” in this chapter.)

   Many browsers let users define their own style sheets for use by the browser. If both the page author and the visitor have included !important in their declarations, the user's declaration wins.

   In theory, an author's style sheets override a visitor's style sheets unless the visitor uses the !important value. In practice, however, most browsers favor a user's style sheet when determining which declarations are used for a tag.

   A The cascade order from most important to least important.
5. **Specificity**

The more contextually specific a rule is, the higher its cascade priority. So the more HTML, class, and ID selectors a particular rule has, the more important it is. In determining this priority, ID selectors count as 100, classes count as 10, and HTML selectors are worth only 1. Thus,

```css
#copy p b { color: red; }
```

is worth 102, whereas

```css
b { color : lime; }
```

is worth only 1. So, the first rule would have higher specificity and the color would be red.

This priority setting may seem a bit silly at first, but it allows context-sensitive and ID rules to carry more weight, ensuring that they will be used.

6. **Order**

If the conflicting declarations applied to an element are equal at this point, CSS gives priority to the last rule listed, in order. Remember that inline styles always win.

7. **Inherited**

These styles are inherited from the parent.

8. **Inherent**

These styles are applied by the browser to HTML tags and are the least important.
This page intentionally left blank
Symbols

"..." (double quotation marks), declaration lists, 39
'...' (single quotation marks), 39
/ (slash), comments, 67
& (ampersand), character entities, 129, 424
* (asterisk), universal selector, 61–63
( ) curly brackets. See curly brackets { }, CSS rules

Numbers

2D transformations, 312–315
3D
stacking objects in, 294–295
text shadow effect, 168
transformations, 316–319
8-bit Unicode Transformation Format (UTF-8)
character set, 126, 423–428

A

absolute positioning, 289–291
absolute values
font size, 143–144
transforming length value, 311
vertical text alignment, 174–175
accessibility
access keys for, 339
color and, 201
content in <body> for, 240
CSS2 emphasis on, 7
accesskey attribute, 339
:active pseudo-class, 84–89
adaptive design
large.css, 368–369
with media queries, 370–373
medium.css, 366–368
print.css, 368–369
responsive Web design, 353
small.css, 366–367
adjacent sibling selector, 78–79
adjacent siblings, family tree, 70
Adobe's Kuler tool, online color scheme, 200
:after pseudo-element, 96, 98–99
Aggregate method, style sheets, 384
alpha values
color transparency, 190
element opacity, 301
text color, 203
alphabetical order, selectors, 378
ampersand (&), character entities, 129, 424
analogous color-combination scheme, 198
angle quotation marks, 244
angle value
2D transforms, 314
3D transforms, 318–319
linear gradients, 191–192
transformations, 311
ANI images, 239
animations, in CSS3, 14
apps, mobile, 357
<article> tag, 28–29, 31
<aside> tag, 28–29, 31, 332
aspect-ratio, media queries, 104–105
asterisk (*), universal selector, 61–63
attachment property, background image, 205–206, 208
attributes, assigning to styles, 2, 100–103
Audio and Video Timed media playback, 27
author styles, 118
auto value
background-image, 205
background-size, 209
clip, 298–299
column-count, 278–279
cursor, 238
margin, 265–267
overflow, 260–261, 336
overriding inheritance, 116
position, 292–293
table-layout, 228
text-align, 171–172
text-align-last, 172
text-justify, 173
width and height, 256
z-index, 294–295
automatic method, table-layout, 228
Available symbol, browsers, 410

B

backface-visibility property, 3D transformations, 318
background, specifying gradient as, 195
background color
choosing, 196–197
differentiating hypertext from text, 340
background color (continued)
  setting, 204–205
  specifying with background image, 211
  styling for print, 114
  text and, 203
  using CSS sprites, 344
background images
  best practices, 404
  border, 274–275
  differentiating hypertext from text, 340
  gradients and multiple, 212–216
  new in CSS3, 14
  setting, 205–211
  using CSS sprites, 343–344
background properties
  adding multiple images and gradients, 212–216
  getting started, 184
  overview of, 183
  putting it all together, 217–218
  quick reference, 414
  setting background color, 204
  setting background images, 205–211
  setting text color, 202–203
<background> property, 204, 251
<background shorthand property, 212–216
<background-color> property
  setting background color, 204
  text shadow effects, 168–170
  transitions, 321
  values, 212
<background-image> property
  setting background images, 206
  setting gradients in background, 195
  transitions, 321
  values, 212
backward compatibility, XHTML2 and, 25
:before pseudo-element, 96, 98–99
best practices
  CSS libraries and frameworks, 376
  minify CSS, 396–398
  overview of, 375
  readable style sheets, 376–380
  style sheet strategy, 381–385
  summary of, 399–407
  troubleshooting CSS code, 386–389
  validating CSS code, 395
  viewing CSS with Firebug, 390–392
  viewing CSS with Web Inspector, 390, 393–394
billing systems, Web font service bureaus, 143
blink value, obsolete for text, 180
block level elements, 22, 330
block value, display property, 252, 254
Blueprint CSS framework, 376
blur
  setting element's shadow, 302–303
  text drop shadow, 166
  text shadow effect, 168
<body> tag
  adding unique name or ID to each, 401
  choosing background color for body, 196
  HTML document setup, 30
  HTML document structure, 19
  keeping content within, 240
  margin setup, 266, 291
  specifying styles with @media, 112
bold, bolder, boldest, for text, 149
border property, 190, 251, 269–270
border-bottom property, 340, 359
border-collapse property, 230–231
border-image property, 274–275
border-radius properties, 272–273
borders
  background image for, 208, 211, 274–275
  collapsing between table cells, 230–231
  colors, 197
  CSS resets for, 359
  element edge within, 287
  of element's box, 269–271
  multiple, 304
  new in CSS3, 14
  rounding corners, 272–273
  setting, 269–271
  setting how box sizes, 258
  setting image, 274–275
  setting padding, 276–277
  border-spacing property, 229
  border-style values, 270–271
  border-width values, 270–271
  box properties
    border image, 274–275
    coming soon, 280
    controlling overflow content, 259–261
    displaying element, 252–254
    element's border, 269–271
    element's margin, 265–267
    element's outline, 268
    element's padding, 276–277
    element's width and height, 255–258
    floating elements in window, 262–264
    multicolumn text layout, 278–279
    overview of, 247–249
    putting it all together, 281–282
    quick reference, 417–419
    rounding border corners, 272–273
    setting how box sizes, 258
    understanding element's box, 250–251
  box shadows, 302–304
  boxes
    fixing for older IE versions, 333, 337–338
    new features in CSS3, 15
    understanding, 250–251
  box-resize property, 251
  box-shadow property, 190, 302–303
  box-sizing property, 258, 338
  break points, 352–353, 366–367
  break tag clearfix, fixing float problem, 335
brightness, using color wheel to choose, 198
browser extensions, CSS3, 12–13
browsers
default margin issues, 267
default styles, 20, 358–362
defining style sheets, 118
displaying documents, 286
evolution of CSS, 6–7
how CSS works, 4–5
inherited styles, 20
mouse pointer appearance and, 238
responsive design for multiple. See responsive Web design
symbols indicating values available for use by.
  See quick reference
teaching to count, 242–243
using HTML5, 29
Web font formats for, 134–135
Web font support, 134
z-index order determined by, 295
bullets, 223–227

cache, font file, 139
Canvas element, new in HTML5, 27
capitalize value, text-transform, 164–165
caption keyword, mimicking visitor's font style, 152
caption-side property, tables, 233–234
cascade order
  !important declaration and, 117–118
  best practices, 403
determining, 119–121
grouped selectors and, 66
troubleshooting CSS code, 386
typeface overrides and, 132
case, setting text, 164–165
characters
  encoding HTML, 129
  encoding HTML and UTL, 423–428
  specifying character set, 126
child elements
  box model, 250
  of element's box, 250
  family tree, 70
  floating elements in window, 263
  in nested tags, 250
  pseudo-classes for styling, 90–91
  setting position of, 293
child selectors, 76–77, 346
choke, setting shadows, 302
Chrome, 393
circle shape value, radial gradients, 193
class selectors
  CSS rules, 9
  defining, 53–56
  elements styled by, 36
troubleshooting CSS code, 386
classes
  defining reusable, 53–56
  generic names for, 401
  mixing and matching, 402
  setting up float, 264
clear property, 264
clearfix class, 335
clip property
  setting background images, 205–206, 210
text-overflow, 261
visibility area, 298–299
clipping, defined, 288
colon (:) CSS declarations, 39
colour Contrast Check tool, 203
ColRD: Palette Creator tool, 200
column-count property, 278
column-gap property, 279
column-rule property, 279
columns. See multicolumn layouts
reusable classes, 53–56
unique IDs, 57–60
universal styles, 61–63
CSS resets
Eric Meyer's, 362
overriding browser default styles with, 358–359
with universal selectors, 63
using universal selector for simple, 360
what you should reset, 359
Yahoo's Reset CSS, 361
CSS sprites
adding CSS image rollovers to Web page, 342–344
best practices using RGB for, 405
creating using background images, 211
origin of, 344
overview of, 342
CSS Validator, 395
CSS1 (CSS Level 1), 7
CSS2 (CSS Level 2), 7
CSS2.1 (CSS Level 2.1), 118
CSS3 (CSS Level 3), 7, 14–15, 29
CSS3 Gradient Generator, 195
CSS4 (CSS Level 4), working draft, 15
CUR images, 239
currentcolor keyword, 188, 204
cursive fonts, 128
cursors, mouse pointer appearance, 238–239
D
debugging CSS, 118
decimal values, color, 185–187, 189
decimals, setting bullet style, 223
declarations
colons in, 39
CSS rules and, 11
grouped selectors receiving same, 64–66
HTML tags and, 38–39, 52
making important, 117–118
quotation marks in, 39
reusable classes and, 54
troubleshooting CSS code, 386, 389
unique IDs and, 58
universal styles and, 61
viewing CSS, 392, 394
default styles, browser, 20, 358–362
DeGraeve’s Color Palette Generator tool, 200
deleted tag, text strikethrough vs., 180
dependent class selector, 36, 54
dependent ID selector, 36, 60
descendants, 70–71, 115–116
Index

design and interface techniques
creating CSS drop-down menu, 345–346
creating multicolumn layouts with float, 330–333
fixing box model for older versions of IE, 337–338
fixing float problem, 334–336
getting started, 328–330
putting it all together, 347–348
styling links vs. navigation, 339–341
using CSS sprites, 342–344
dingbats, 129
display area, browser, 287
display property, 252–254, 297
<dir> tag, 27, 56, 332
Divide and Conquer method, style sheet strategy, 383
doctype (<!DOCTYPE>)
  browser modes set by, 363
  HTML document structure, 19, 30–31, 399
  for markup languages, 29
  reasons to include, 32–33
document type definition (DTD), 337–338
document type definition (DTD), 337–338
documents
  basic HTML, 30–31
  browser windows displaying, 286
  editing in HTML5, 27
  parts of, 286–287
  structure of HTML, 20
double quotation marks ("..."), declaration lists, 39
drag-and-drop, 27
drop shadows, 166–168, 302–303
drop-down menus, CSS, 345–346
DTD (document type definition), 337–338
Dynamic method, style sheet strategy, 385
dynamic pseudo-classes, 82–83, 88–89, 320–324
dynamic styles, 402
E
  editing declarations, 392, 394
element edge, 287
element family tree, 70
elements, HTML
  applying CSS properties to specific, 23
  block-level, 22
  controlling overflow, 259–261
  displaying, 252–254
  floating in window, 262–264
  including default styles for, 401
  inspecting, 392–394
  not setting style for particular, 94–95
  setting border, 269–270
  setting margins of, 265–267
  setting outline, 268
  setting padding, 276–277
  setting positions, 290–291
  setting width and height, 255–258, 287
  types of, 21–23
understanding box of, 250–251
visual formatting properties. See visual formatting properties
ellipsis, text-overflow property, 261
elliptical corners, borders, 273
elliptical shape value, radial gradients, 193
<em> tag, 20, 72–81, 144–145, 160, 174
Emastic CSS framework, 376
Embedded OpenType (EOT) Web font format, 134–135, 137–138, 140
embedded style sheets
  adding, 40–42
  making declaration !important, 117–118
  not placing in final code, 42
  using @media rule to specify styles, 112
emboss text, text shadows, 168
emotional associations, color, 196
empty-cells property, 232
encoding
  HTML and URL character, 423–428
  HTML character entities, 129
End User License Agreements (EULA), Web fonts, 140
EOT (Embedded OpenType) Web font format, 134–135, 137–138, 140
EULA (End User License Agreements), Web fonts, 140
evolution, of CSS, 6–7
expanded fonts, 150
extensions, CSS browser
  creating color gradients, 191
  defined, 12
  importance of coding with, 326
external CSS files
  adding styles to Web sites, 43–49
  defining CSS rules in, 52, 57
external style sheets
  best practices, 400
  making declaration !important, 117–118
  placing all styles in, 381
  progressive enhancement and, 355
  using @media rule to specify styles, 112
F
  fantasy fonts, 128
<figcaption> tag, HTML5, 28
<figure> tag, HTML5, 28
fire effect text, text shadows, 168
Firebug, 390–392, 394
:first-child pseudo-class, 90–91
:first-letter pseudo-element, 96–99
:first-line pseudo-element, 96–99
:first-of-type pseudo-class, 90–91
fixed design, multicolumn layout, 332–333
fixed method, table-layout, 228
fixed positioning, 290–291
Flash, 20
float property, 262–264, 330–336
:focus pseudo-class, 88–89
font families
  defined, 125
  fonts vs., 130
  generic, 127–128
  naming, 137
  setting font-stack, 130–132

font properties
  condensed and expanded fonts, 150
  dingbats, 129
  encoding HTML character entities, 129
  generic font families, 127–128
  getting started, 124
  making text italic, 147–148
  overview of, 123
  putting it all together, 155–156
  quick reference, 412
  setting bold, bolder, boldest, 149
  setting font-stacks. See font-stacks
  setting multiple font values at same time, 152–154
  setting size, 144–145
  sizing understudy fonts, 146
  small caps, 151
  specifying character set, 126
  understanding Web typography, 125
  using Web fonts, 133

font property, 152–154
  Font Squirrel Web site, 140
  @font-face Kit Generator, Font Squirrel, 140
  @font-face rule
    italic text, 147
    placing at top of CSS code, 378
    small caps, 151
    Web font service bureaus using, 142–143
    Web fonts using, 136–140
font-family property, 154
  fonts. See also Web fonts
    applied to styles, 2
    new features in CSS3, 15
    for readable style sheets, 378
    word processor styles vs. CSS, 3–4
font-size property, 143–145, 153, 175
font-size-adjust property, 146
  font-stacks
    readable style sheets, 378–379
    setting, 130–132
    setting with Web fonts, 135–139
font-stretch property, 138–139, 150
font-style property
  defining Web font to font stack, 138–139
  making text italic, 147–148
  mimicking visitor’s font style, 152
  setting multiple font values, 152, 404
font-variant property, 151, 153
font-weight property, 138–139, 149, 153
  <footer> tag, HTML5, 28–29, 31
footnotes, hyperlinking, 175
  foreground color, 201–203

formats
  visual. See visual formatting properties
  Web font, 134–135
forms, background color for, 197
frame effect, box shadows, 304
frame tags, eliminated in HTML5, 27
frameworks, CSS libraries and, 376
functionality
  organizing style sheets by, 378
  progressive enhancement guidelines, 355

G
  general sibling selector, 78, 80–81
  generated content properties
    adding content using CSS, 240–241
    getting started, 236–237
    overview of, 235
    putting it all together, 246
    quick reference, 416
    specifying quote style, 244–245
    teaching browser to count, 242–243
  generic class names, 401
  generic font families, 127–128, 131
  glyphs
    encoding HTML and UTL characters, 423–428
    encoding HTML characters, 129
    overview of, 125–126
    graceful degradation, 356
  gradients, color
    adding to backgrounds, 206, 216
    best practices, 405
    creating, 191–195
  graphics
    as bullets, 223
    CSS Sprites. See CSS Sprites
  grids, page layout using column, 330–333
  grouping selectors, 64–66, 402

H
  hanging indents, bulleted lists, 225
  hanging punctuation, 181
  <head>
    defining class selector in, 53
    defining CSS rules in, 52, 57
    embedded styles in, 40–42
    HTML document structure, 19, 30
    linking to external CSS files, 46
    specifying character set in, 126
    using @media rule to specify styles, 112
    using links to add styles in, 400
  <header> tag
    background color for, 197
    conditional styles for IE, 365
    defined, 28–29
    HTML document structure, 31
  height
    browser, 286
    CSS resets for line, 359
ID selectors
adding to body tag of page, 401
CSS rules, 10
defining unique IDs, 57–60
determining cascade order, 121
elements styled by, 36
troubleshooting CSS code, 386
using specific ID names, 401
IE Conditional, 32–33
iframes, 286
image rollovers, 342–344
images
resizing using width and height properties, 257
setting background, 205–211
setting border, 274–275
setting bullet style using, 223
using as custom cursor, 239
<img> tag, 174, 404
@import rule
adding styles with, 400
best practices, 381
discouraged due to browser issues, 49
favoring <link> over, 406
importing external CSS files, 48–49
importing style sheets, 48
placing at top of CSS code, 378
style sheet methods using, 384–385
!important declaration
avoiding, 403
cascade order and, 117–118, 120
troubleshooting CSS code, 389
importing external styles, to site, 43–44, 48–49
indented paragraphs, 176
indented text, bulleted lists, 225
inherent styles, 115–116, 120–121
inherit keyword, 291, 296
inherit value
clip, 298
display, 253
inheritance of styles, 116
opacity, 301
inherent styles, 115–116, 119–121
inline elements, 21
inline styles, 37–40, 120
inline value, display, 252
inline-block value, display, 253
inset, shadows, 302–304
interaction, styling Web page, 88
Internet Explorer
adjusting CSS for, 363–365
best practices, 405
converting OTF/TFF files to, 140
defining Web font to font stack, 135–139
fixing box model for older versions of, 337
fixing code in, 405
HTML5 for older versions of, 32–33
quirks mode, 363
resetting browser default styles, 360
Intuit CSS framework, 376
iPhone, 107–110
ISO 8859-1 character set, 126
italicized text, 147–148, 340

J
JavaScript, 8, 32–33
justified text, 173

K
kerning, 160, 181
keywords
color, 185–187
linear gradients, 191
overflowing content, 260
radial gradients, 193
shadow, 302–303
transformation origin, 315

L
:lang() pseudo-class, 92–93
language, styling for specific, 92–93
large.css, 368–369
:last-child pseudo-class, 90–91
:last-of-type pseudo-class, 90–91
layouts
  multicolumn, 278–279, 330–333
  Web design. See responsive Web design
leading, adjusting, 162–163
left keyword, 263–264
legal issues, Web fonts, 140
legibility, and font size, 144
length value
  adjusting leading, 162–163
  background images, 208–209
  borders, 270, 272
  clipping visibility area, 299
defined, 311
element width or height, 256
indenting paragraphs, 176
letter spacing, 159
margins, 266
multicolumn text layout, 278
multiple background images, 215
padding, 276
positioned elements, 292
shadows, 302–303
spacing table cells, 229
text drop shadows, 166
transformations, 311
transitions, 320–324
word spacing, 161
letterforms. See fonts
letterpress text, text shadows, 168
letters
  adjusting space between, 159–160
  bullet style for, 223
letter-spacing property, 159–160
libraries and frameworks, CSS, 376
licensed fonts, 140–143
lightness, HSL values, 189–190, 198
line height, CSS resets for, 359
linear gradients, 191–192, 195
line-height property, 153, 162–163
line-through, text, 179
link pseudo-classes
  adding transitions between states, 320–324
  importance of order, 89
  overview of, 82–83
  styling links, 84–87
link states
  add CSS image rollovers to Web page, 344
  contrasting link appearances, 86
  styling all, 406
$link tag
  best practices, 381, 400
  conditional styles for IE, 364
  connecting external CSS and HTML files, 43, 45
  favoring over @import, 406
  linking to style sheets with, 46–47
  media queries, 108–111
  specifying styles with @media rule, 112
  WFSBs using, 142
links
  color for, 197
  contrasting appearances for, 84–87
  to external CSS file, 43, 46–47
  minimizing, 47
  navigation vs. styling, 406
  states, 82
  to style sheet, 46
  styling, 84–87
  styling documents in <head>, 400
  styling navigation, 339–341
text shadows for, 167
troubleshooting CSS code, 386
list properties
  bullets, 223–225
  getting started, 220–222
  multiple list styles, 226–227
  quick reference, 415
list-item value, display, 253
lists
  background color for, 197
  sequentially numbered, 242–243
list-style shorthand property, 226–227
list-style-image property, 224, 226
list-style-position property, 225, 227
list-style-type property, 223, 227
Little Troublemaker Girl font, 139
local versions, Web font service bureau, 143
logical operators, conditional styles for IE, 364
LoVe HAte mnemonic, pseudo-element order, 89
case text, 164
multiple backgrounds
  adding with **background** shorthand, 212–216
  layering images, 205

multi-screens, 357

naming conventions
  class and ID selectors, 60
  defining Web font to font stack, 137
  external CSS files styling HTML pages, 45
  generic class names, 401
  reusable classes, 53–56
  unique IDs, 57

<nav> tag, 28–29, 31

navigation
  choosing color for links, 197
  preventing noise, 345–346
  styling for print, 114
  styling links, 339–341

negative margins, 266

neon glow text effect, 168

nested comments, not allowed, 67

nested tags, 70, 250

“New in CSS3” mark, in this book, 14

newspaper style, horizontal text alignment, 171

noise, reducing navigation, 345–346

none value
  clear property, 264
  display property, 254–255, 297
  float property, 263
  inheritance of styles, 116

normal flow, 287

normal text, 147–148

normal value, inheritance of styles, 116

Not available symbol, browsers, 410

nowrap value, controlling white space, 177–178

: nth-child( # ) pseudo-class, 90–91

: nth-last-of-type( # ) pseudo-class, 90–91

: nth-of-type( # ) pseudo-class, 90–91

number sign (#), ID rules, 57–58

numbers
  adjusting font size for understudy fonts, 146
  bullet style, 223
  centering in blocks horizontally, 336
  font-weight, 149
  multiple counter lists, 242–243
  positioning in browsers, 291
  stacking elements in 3D, 294–295
  transformations, 311

O

-o- extension, CSS, 12–13

oblique text, 147–148

offset, shadows, 166, 302

One for All method, style sheet strategy, 382

online references
  color and accessibility, 201
  Colour Contrast Check tool, 203
online references (continued)
CSS frameworks, 376
CSS3 development, 7
CSS3 Gradient Generator, 195
CSS4 working draft, 15
development of new text properties, 181
Eric Meyer’s CSS reset, 362
Firebug, 391
Font Squirrel Web site, 140
Mac and Windows fonts, 139
online color scheme, 200
Scalable Vector Graphics, 20
Web font service bureaus, 142
Web font stores, 142
Web Inspector, 393
Web Typography NOW, 143
Web-safe fonts, 133
World Wide Web Consortium, 10
Yahoo’s Reset CSS, 361
opacity
alpha values for color transparency, 190
new features in CSS3, 15
setting element’s, 300–301
transitioning CSS properties, 320–324
opacity property, 190, 300–301
OpenType (OTF) Web font format, 134, 138, 140
optimization, creating minified version of CSS code, 396–398
organization scheme, readable style sheets, 378–379
orientation property, media queries, 106
origin property, background images, 205–206, 211
OTF (OpenType) Web font format, 134, 138, 140
outline property, 268
outlines
CSS resets for, 359
defining color with, 190
of element’s box, 251, 268
text, 181
overflow property
controlling content, 257, 259–260
fixing float problem with, 335
setting for text, 261
overlapping text, 266
overlines, text, 179–180
overrides
float property, 263
font size, 144
font-stack, 132
mouse pointer, 238
P
padding
background images, 208, 211
CSS resets for, 359
of element’s box, 251
favoring margins over, 405
fixing older versions of Internet Explorer, 338
indenting paragraphs and, 176
multicolumn layouts and, 333
padding property, 276–277
page breaks, styling for print, 114
page structure, style sheets by, 378
paragraphs
indenting, 176
justifying last line of text in, 172
parent elements
centering element within, 267
family tree, 70
floating elements in window, 263
in nested tags, 250
positioning, 293
styling descendants, 71
parent selector, 71, 76–77
percentage values, RGB, 189
perspective property, 3D transformations, 317
perspective-origin property, 3D transformations, 317
picture fonts, 129
pixel perfection, 354
position property, background images, 205–206, 208–209, 215
positioned elements, 291–295
positioning
bullets, 225
radial gradients, 193
table captions, 233
transitioning CSS properties, 320–324
types of, 288–291
pre value, white space, 177–178
preceding sibling elements, family tree, 70
presentation tags, eliminated in HTML5, 27
primary font, 130
print
indenting paragraphs for, 176
specifying styles with media queries, 107, 109–111
styling for, 114
print.css, responsive Web design, 368–369, 373
progressive enhancements, 352, 354–356
properties
CSS rules and, 11
giving transitions to, 320–321
inheritance of styles, 115–116
media queries, 104–105
overview of CSS, 4
redefining HTML tag, 50–52
styling based on HTML tags, 20, 23, 38–39, 100–103
troubleshooting CSS code typos, 386
pseudo-classes
defining dynamic, 88–89
not styling elements with, 94–95
overview of, 82–83
quick reference, 411
setting bullet type, 223
styling children, 90–91
styling contrasting links, 84–87
styling for interaction, 88
styling for languages, 92–93
styling links, 84
pseudo-elements
defined, 96
highlighting beginning of article, 96–97
quick reference, 411
setting content before and after element, 98–99
working with first letters and lines, 96
pt (point) size, print media, 144
Q
<q> tag (quotation), 236–237, 244–245
quick reference
basic selectors, 410
box properties, 417–419
color and background properties, 414
defining Web font to font stack, 137
font properties, 412
list properties, 415
pseudo-classes, 411
pseudo-elements, 411
text properties, 413
transform properties, 421
transition properties, 422
user interface and generated content properties, 416
visual formatting properties, 420
quirks mode, Internet Explorer
defined, 363
fixing box model for older versions of Internet Explorer, 337
quotation (<q> tag), 236–237, 244–245
quotation marks
specifying quote style with <q> tag, 236–237, 244–245
tips for using, 39
quotes property, 244–245
R
radial gradients
available in all browsers, 191
overview of, 193–194
repeating, 195
readable style sheets, best practices
@rules at top of CSS code, 378
colors, fonts, and other constants, 378
introduction and TOC, 376–377
organization scheme, 378–379
overview of, 376
section headers, 378
specificity, 380
Recently available symbol, browsers, 410
rect value, clip, 298–299
references
online. See online references for subjects in this book. See quick reference
rel property, troubleshooting CSS code, 386
relationship, link, 46, 386
relative positioning, 175, 289–291
relative values
aligning text vertically, 174–175
font sizes, 143–144
transforming length value, 311
rendering engine, 4–5, 12–13
repeat property, background images, 205–207, 214
repeating gradients, 195
repetition, avoiding unnecessary code, 403
resetting styles, responsive Web design, 352
resolution property, media queries, 106
responsive Web design
adapting to environment, 366–373
adjusting CSS for Internet Explorer, 363–365
developing multi-screen strategy, 357
getting started, 350–351
overview of, 349
with progressive enhancements, 354–356
resetting browser default styles, 358–362
understanding, 352–353
reusable classes, 53–56, 401
RGB (red, green, blue)
best practices, 404
decimal values, 189
hex values, 188
percentage values, 189
setting alpha channels for decimal values, 190
RGBA value, adding text color, 203
right keyword, 263–264
rotate(), rotateX(), or rotateY() value, 2D transforms, 314–315
rotate3d() value, 3D transforms, 319
rotateZ() value, 3D transforms, 319
rounded corners, borders, 272–273
rules, CSS
adding embedded style to Web page, 41–42
applying with CSS selectors, 36
class selector, 9
combining into selector lists, 402
defining Web font to font stack, 137
default style of external CSS file, 45
HTML selector, 9
ID selector, 10
multicolumn layout, 332–333
parts of, 11
placing @rules at top of CSS code, 378
separating chunks of content, 197
syntax of, 11
tips for, 42
troubleshooting CSS code, 389
types of, 9–10
universal selector, 10
using specificity for hierarchy of, 380
@rules, placing at top of CSS code, 378
run-in value, display, 253
S
Safari, 393
sans-serif fonts, 127
saturation, HSL values, 189–190, 198
Scalable Vector Graphics (SVG) Web fonts, 20, 135, 139
scale(), scaleX(), or scaleY() value, 2D transforms, 314–315
scale3d() value, 3D transforms, 319
scaleZ() value, 3D transforms, 319
scientific notation, 175
scroll, 205, 208, 215
scroll keyword, overflow, 260
search engines, content property and, 241
section headers, readable style sheets, 378–379
<section> tag, 28–29, 336
selective styling
  !important declaration, 117–118
  @media rule, 112–113
  based on context. See contextual styles
  based on tag attributes, 100–103
  cascade order, 119–121
  element family tree, 70
  inheritance of styles, 115–116
  media queries, 104–111
  overview of, 69
  for print, 114
  with pseudo-classes. See pseudo-classes
  with pseudo-elements, 96–99
selectors
  attribute, 100
  basic CSS, 36
  combinatorial, 71
  grouping, 64–66
  HTML. See HTML selectors
  organizing style sheets by types of, 378
  pseudo-class, 83
  pseudo-element, 96
  quick reference to, 410
  styling elements to exclude certain, 94–95
  troubleshooting CSS code, 386
semicolons (;)
  character entities beginning with, 129
  defining styles directly in HTML tag, 38
  locating in !important declaration, 118
  separating multiple declarations, 11
  troubleshooting CSS code, 386
separate value, border-collapse, 230–231
serif fonts, 127
SGML (Standard Generalized Markup Language), 19
shadows
  adding text drop, 166–170
  setting element’s, 302–303
shape value, radial gradients, 193
shorthand properties
  !important, 404
  background, 212–216
  best practice to use, 404
  font, 152–154, 163
  list-style, 226–227
  matrix(), 314
  matrix3d(), 319
  overriding value set by, 154
  transition, 322–324
  sibling selector, 78–79
  siblings, family tree, 70
  single quotation marks (‘…’), 39
  size, file
    best practices, 381
    drawbacks of single master style sheet, 382
    reducing, 381
  setting font-stack and, 132
  size property, background images, 205–206, 209
  size values, radial gradients, 193
  skew(), skewX(), or skewY() value, 2D transforms, 314
  slash (/), comments, 67
  small-caps, 151
  small-caption: keyword, 152
  small.css, 366–367, 370–372
  smart quotes, 39
  spacing
    horizontal text alignment, 171–173
    between table cells, 229
  span tag, 56
  specificity
    best practices, 402–403, 406
    cascade order determined by, 121
    hierarchy of CSS rules, 380
    stacking order, 288, 294–295
    Standard Generalized Markup Language (SGML), 19
  states
    adding transitions between element, 320–324
    link, 82, 84
    styling navigation and link, 339–341
    using CSS sprites, 342–344
  static positioning, 288, 290–291
  status-bar: keyword, 152
  stretched images, borders, 275
  strict mode, browsers, 363
  strikethrough, text, 180
  strong tag
    aligning text vertically, 174
    defining font size, 145
    as nested tag, 70
    redefining in CSS, 8, 50–52
    styling descendents, 72–75
    styling siblings, 78–81
  structural elements
    CSS rules, 11
    new features in HTML5, 27–28
    placing before designing, 399
    using, 29
  structural pseudo-classes, 82–83
style sheet strategies
  Aggregate method, 384
  best practices, 381
  Divide and Conquer method, 383
  Dynamic method, 385
  One for All method, 382
<styles> tag
  adding embedded styles, 40–42
  applying media queries, 111
  responsive design with media queries, 373
  troubleshooting CSS code, 386
styles
  browser default, 20
  horizontal text alignment, 171–172
  making text italic, 147–148
  progressive enhancement honoring user, 355
  resetting browser default, 358–362
  text justification, 173
  word processor, 2
  word processor vs. CSS, 3–4
subscript, vertical text alignment, 175
sub-settings, Web font service bureaus, 143
superscript, hyperlinking footnotes, 175
SVG (Scalable Vector Graphics) Web fonts, 20, 135, 139
symbol fonts, 129
syntax
  !important declaration, 117–118
  @media rule, 112
  CSS rules, 11
  defining styles in HTML tag, 37–39
  older gradient, 195

T

  table of contents (TOC), section headers
    mimicking, 378–379
  table properties
    collapsing borders between table cells, 230–231
    getting started, 220–222
    setting space between table cells, 229
    setting table caption position, 233
    setting table layout, 228
    showing/hiding empty table cells, 232
  table value, display property, 253
  table-layout property, 228
  tables, background color, 197
tags, HTML
  adding inline style to, 36
  associated with browser inherited styles, 20
  CSS files should not contain, 45
  defining styles based on attributes of, 100–103
  HTML document setup, 30–31
  HTML properties and, 19–20
  HTML5 in older versions of IE, 32–33
  HTML5 structure and, 28
  redefining, 50–52
  understanding, 18
  <td> tag, 174

  technology, Web font service bureaus, 142–143
testing
  best practices, 405
  minified version of CSS code, 398
  troubleshooting CSS code, 389
  using inline styles, 39

  text. See also font properties
  adding HTML tags to, 18
  controlling overflow, 261
  CSS resets for underlining, 359
  decorating, 179–181
  design limitations of, 125
  in graphics or Flash vs., 125
  multicolumn layout, 278–279
  new features in CSS3, 14
  positioning overlapping, 266
  styling for print by avoiding in graphics, 114
  wrapping and bullets, 225

  text properties
    aligning horizontally, 171–173
    aligning vertically, 174–175
    coming soon, 181
    decorating, 179–180
    drop shadows, 166–170
    getting started, 158
    indenting paragraphs, 176
    overview of, 157
    putting it all together, 182
    quick reference, 413
    setting case, 164–165
    spacing, 159–163
    white space control, 177–178
  text-align property, 171–173
  text-align-last property, 172
  text-decoration property, 179–181
  text-indent property, 176
  text-justify property, 173
  text-overflow property, 261
  text-shadow property, 166–170, 190
  text-transform property, 164–165
  tiled images, borders, 275
titles, small caps for, 151
TOC (table of contents), section headers
    mimicking, 378–379
tracking, 159–160
transform properties
  2D transformations, 312–315
  3D transformations, 316–319
  getting started, 308–310
  new features in CSS3, 14
  overview of, 307
  putting it all together, 325–326
  quick reference, 421
  transforming elements, 311
  transform property, 311, 313–315
  transform value, 324
  transform-origin keywords, 315
  transform-style property, 316–319
transition properties
3D transformations, 316–319
adding transitions between element states, 320–324
going started, 308–310
new features in CSS3, 14
overview of, 307
putting it all together, 325–326
quick reference, 422
transition property, 322–324
transition-delay values, 323–324
transition-duration values, 323–324
transition-property value, 323–324
transition-timing-function values, 323–324
translate(), translateX(), or translate(Y)
value, 2D transforms, 314
translated() value, 3D transforms, 319
translateZ() value, 3D transforms, 319
transparency
creating border with, 304
setting alpha values for color, 190
styling for print by avoiding color, 114
transparent keyword, 204, 213
triad color-combination scheme, 199
troubleshooting CSS code, 386–389
TrueType (TTF) Web font, 134, 138, 140
TTF (TrueType) Web font, 134, 138, 140
type families. See font families
typeface overrides, 132
typography
affecting how text appears, 157
understanding on Web, 125
using Web fonts for. See Web fonts
V
undecorating text, 179–180
underlines
differentiating hypertext from text, 340
text decorations, 179–180
using border-bottom for text, 359
understudy fonts, 131, 133, 146
universal selector
adding default transitions to, 324
CSS resets using, 360, 362
CSS rules, 10
defining universal styles, 61–63
elements styled by, 36
styling descendants universally, 74–75
unvisited links, setting appearance, 86–87
uppercase, setting text case, 164–165
URL
adding multiple background images, 214
changing mouse pointer appearance, 239
defining background image, 206
defining own graphic bullets, 224
setting border image, 275
user interface
designing. See design and interface techniques
inline styles, 39
user interface properties
going started, 236–237
mouse pointer appearance, 238–239
overview of, 235
putting it all together, 246
quick reference, 416
UTF-8 (8-bit Unicode Transformation Format)
character set, 126, 423–428
visibility
  clipping element’s area of, 298–299
  setting element’s, 296–297
visibility property, 254, 288
visible keyword, overflow, 260
visited links, setting appearance of, 86–87
:visited pseudo-class, 84–87
visitor styles, determining cascade order, 120
visual formatting properties
  clipping visibility area, 298–299
  getting started, 284–285
  opacity, 300–301
  overview of, 283
  position, 292–293
  positioning type, 288–291
  putting it all together, 305
  quick reference, 420
  shadows, 302–304
  stacking objects in 3D, 294–295
  visibility, 296–297
  window and document, 286–287
W
W3C (World Wide Web Consortium)
  browser specifications for rendering Web code, 5
  CSS Validator for CSS code, 395
  evolution of CSS under, 6–7
  evolution of HTML5 under, 24–25
  understanding, 10
Web design. See responsive Web design
Web fonts
  converting using Font Squirrel, 140
  defining to font stack, 137–139
  formats, 134–135
  overview of, 133–134
  setting better stack font with, 135–136
  using dingbats as, 129
  using for icons, 408
  Web-safe fonts, 133
Web forms, 27
Web Inspector, viewing CSS with, 390, 393–394
Web Open Font Format (WOFF), 135, 138, 140
Web page
  adding embedded styles to, 40–42
  as document. See documents
Web sites
  adding external CSS file to, 43–49
  for multi-screen strategy, 357
  Webdings, 129
  Webkit browser extensions, 13
  -webkit extension, CSS, 12–13
  weights, setting bold/bolder/boldest, 149
  WFSB (Web font service bureaus), 141–143
  WHATWG (Web HyperText Application Technology Working Group), HTML5, 25–26
white-space property, text, 177–178
width
  browser windows and documents, 286–287
  defining element, 255–257
  of element’s box, 250
  fixing older versions of IE, 337–338
  floating elements in window, 263
  media queries and, 104–105
  multicolumn layouts and, 332–333
  viewport, 287
width property, 255–257
windows, browser, 286–287
Windows fonts, online resource, 139
WOFF (Web Open Font Format), 135, 138, 140
word-processor styles, 2–4
word-spacing property, 161
wrapped text, bullet positions, 225
X
XHTML, 24, 29
XHTML2, 24–26
XHTML5, 26, 29
XML, 24–26
Y
Yahoo’s Reset CSS, 361
Yahoo’s YUI Grids framework, 376
Z
z-index property, 294–295
Unlimited online access to all Peachpit, Adobe Press, Apple Training and New Riders videos and books, as well as content from other leading publishers including: O’Reilly Media, Focal Press, Sams, Que, Total Training, John Wiley & Sons, Course Technology PTR, Class on Demand, VTC and more.

No time commitment or contract required! Sign up for one month or a year. All for $19.99 a month

SIGN UP TODAY
peachpit.com/creativeedge