

VISUAL **QUICKSTART** GUIDE



CSS3



JASON CRANFORD TEAGUE

© LEARN THE QUICK AND EASY WAY!

VISUAL QUICKSTART GUIDE

CSS3

SIXTH EDITION

JASON CRANFORD TEAGUE



Peachpit Press

Visual QuickStart Guide

CSS3

Sixth Edition

Jason Cranford Teague

Peachpit Press

www.peachpit.com

Find us on the Web at www.peachpit.com

To report errors, please send a note to errata@peachpit.com

Peachpit Press is a division of Pearson Education

Copyright © 2013 by Jason Cranford Teague

Project Editor: Nancy Peterson

Development Editor: Bob Lindstrom

Copyeditors: Liz Merfeld and Darren Meiss

Production Editor: Katerina Malone

Compositor: David Van Ness

Indexer: Jack Lewis

Cover Design: RHDG / Riezebos Holzbaur Design Group, Peachpit Press

Interior Design: Peachpit Press

Logo Design: MINE™ www.minesf.com

Notice of Rights

All rights reserved. No part of this book may be reproduced or transmitted in any form by any means—electronic, mechanical, photocopying, recording, or otherwise—without the prior written permission of the publisher. For information on obtaining permission for reprints and excerpts, contact permissions@peachpit.com.

Notice of Liability

The information in this book is distributed on an “As Is” basis, without warranty. While every precaution has been taken in preparation of this book, neither the author nor Peachpit shall have any liability to any person or entity with respect to any loss or damage caused or alleged to be caused directly or indirectly by the instructions contained in this book or by the computer software and hardware products described in it.

Trademarks

Visual QuickStart Guide is a registered trademark of Peachpit Press, a division of Pearson Education.

Other trademarks are the property of their respective owners.

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and Peachpit was aware of the trademark claim, the designations appear as requested by the owner of the trademark. All other product names and services identified throughout the book are used in an editorial fashion only and for the benefit of such companies with no intention of infringement of the trademark. No such use, or the use of any trade name, is intended to convey endorsement or other affiliation with this book.

ISBN 13: 978-0-321-88893-8

ISBN10: 0-321-88893-6

9 8 7 6 5 4 3 2 1

Printed and bound in the United States of America

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*.

Douglas Adams, **H.P. Lovecraft**, **Neil Gaiman**, **Philip K. Dick**, and **Carl Sagan** whose teachings and writings inspire me every day.

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**, **Smooove 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

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

This page intentionally left blank

Table of Contents

	Introduction	xiii
Chapter 1	Understanding CSS3	1
	What Is a Style?	2
	What Are Cascading Style Sheets?	3
	The Evolution of CSS	6
	CSS and HTML.	8
	Types of CSS Rules	9
	The Parts of a CSS Rule.	11
	CSS Browser Extensions	12
	New in CSS3	14
Chapter 2	HTML5 Primer	17
	What Is HTML?	18
	Types of HTML Elements	21
	The Evolution of HTML5	24
	What's New in HTML5?	27
	How Does HTML5 Structure Work?	28
	Using HTML5 Structure Now.	29
Chapter 3	CSS Basics.	35
	The Basic CSS Selectors	36
	Inline: Adding Styles to an HTML Tag	37
	Embedded: Adding Styles to a Web Page	40
	External: Adding Styles to a Web Site.	43
	(Re)Defining HTML Tags	50
	Defining Reusable Classes.	53
	Defining Unique IDs.	57
	Defining Universal Styles.	61
	Grouping: Defining Elements That Are Using the Same Styles.	64
	Adding Comments to CSS	67

Chapter 4	Selective Styling	69
	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
 Chapter 5	 Font Properties	 123
	Getting Started	124
	Understanding Typography on the Web	125
	Setting a Font-Stack.	130
	★Using Web Fonts	133
	Setting the Font Size	144
	★Adjusting Font Size for Understudy Fonts.	146
	Making Text Italic	147
	Setting Bold, Bolder, Boldest.	149
	★Using Condensed and Expanded Fonts.	150
	Creating Small Caps.	151
	Setting Multiple Font Values at the Same Time	152
	Putting It All Together.	155
 Chapter 6	 Text Properties.	 157
	Getting Started	158
	Adjusting Text Spacing	159
	Setting Text Case	164
	★Adding a Text Drop Shadow.	166
	★Aligning Text Horizontally	171
	Aligning Text Vertically	174
	Indenting Paragraphs.	176
	Controlling White Space	177
	Decorating Text	179
	★Coming Soon!.	181
	Putting It All Together.	182

Chapter 7	Color and Background Properties	183
	Getting Started	184
	Choosing Color Values	185
	★ Creating Color Gradients	191
	Choosing Your Color Palette	196
	Setting Text Color	202
	Setting a Background Color	204
	★ Setting Background Images.	205
	Using Background Shorthand to Add Multiple Background Images and Gradients.	212
	Putting It All Together.	217
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

	★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

Chapter 14	Responsive Web Design.	349
	Getting Started	350
	What Is Responsive Design?	352
	Designing with Progressive Enhancements	354
	Resetting Browser Default Styles	358
	Adjusting CSS for Internet Explorer	363
	Adapting to the Environment.	366
 Chapter 15	 CSS Best Practices	 375
	Create Readable Style Sheets	376
	Have a Style Sheet Strategy	381
	Troubleshoot Your CSS Code	386
	View CSS in Firebug or Web Inspector	390
	Validate Your CSS Code	395
	Minify Your CSS	396
	33 CSS Best Practices	399
 Appendix A	 CSS Quick Reference.	 409
	Basic Selectors	410
	Pseudo-Classes	411
	Pseudo-Elements	411
	Font Properties	412
	Text Properties.	413
	Color and Background Properties.	414
	List Properties	415
	Table Properties.	415
	User Interface and Generated	
	Content Properties	416
	Box Properties	417
	Visual Formatting Properties.	420
	Transform Properties	421
	Transition Properties	422
 Appendix B	 HTML and UTF Character Encoding	 423
	HTML and UTF Character Encoding.	424
	 Index	 429

This page intentionally left blank

Introduction

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.

CSS3 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 `
` 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

Value Type	What It Is	Example
<number>	A number	1, 2, 3
<length>	A measurement of distance or size	1in
<color>	A chromatic expression	red
<percentage>	A proportion	35%
<URL>	The absolute or relative path to a file on the Internet	http://www.mySite.net/images/01.jpg

TABLE i.2 Relative Length Values

Unit	Name	What It Is	Example
em	Em	Relative to the current font size (similar to percentage)	3em
ex	x-height	Relative to the height of lowercase letters in the font	5ex
px	Pixel	Relative to the monitor's resolution	125px

TABLE i.3 Absolute Length Values

Unit	Name	What It Is	Example
pt	Point	72pt = 1inch	12pt
pc	Picas	1pc = 12pt	3pc
mm	Millimeters	1mm = .24pc	25mm
cm	Centimeters	1cm = 10mm	5.1cm
in	Inches	1in = 2.54cm	8.25in

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 **A**. **Table i.4** lists the browser icons and abbreviations used in this book.

TABLE i.4 Browser Abbreviations

Icon	Abbreviation	Browser
	IE	Microsoft Internet Explorer
	FF	Mozilla Firefox
	Op	Opera
	Sa	Apple Safari
	Ch	Google Chrome

Text-Overflow Values					
Value					
Values supported by this property.	clip	◊	●9	●	◊
	ellipsis	◊	●	●	◊
	inherit	◊	●	●	◊
A circle indicates browser support.		A diamond indicates support with browser extension (-moz-, -webkit-, -o-, or -e-).			
		If a number is added, indicates support is recent since that version.			
		A blank entry indicates no support.			

A 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:

```
<style type="text/css">
p { font-size: 12pt; }
</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:

```
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—retyping 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.

4

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

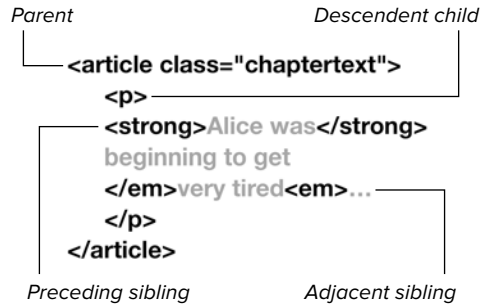
When a tag is surrounded by another tag—one inside another—the tags are *nested*.

```
<h2><strong>Chapter 2</strong> The  
→ Pool of Tears</h2>
```

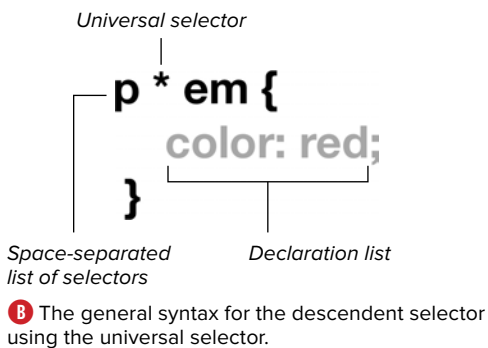
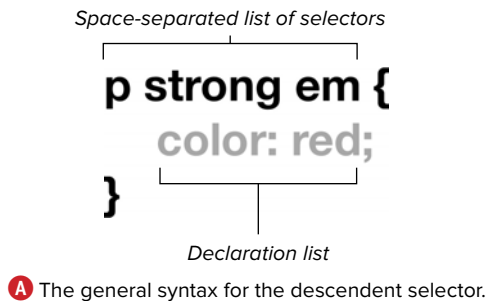
In a nested set, the outer element in this example (`<h2>`) is called the *parent*, and the inner element (``) is the *child*. The child tag and any children of that child tag are the parents' *descendents*. Two tags in the same parent are called *siblings*, and two tags immediately next to each other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

- **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 descendent elements 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.



A The article element is the parent to the elements created by the paragraph, strong, and emphasis tags, which are its descendents. Only the paragraph tag is a direct child. The elements created by the emphasis and strong tags are the children of the paragraph tag, and each other's siblings.



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 *combinatory* 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 4.1 Combinatory Selectors

Format	Selector Name	Elements Are Styled If...					
a b c	Descendent	c descendent of b descendent of a	●	●	●	●	●
a * b	Universal	b within a regardless of b's parents	●	●	●	●	●
a > b	Direct child	b direct child of a	●	●	●	●	●
a + b	Adjacent sibling	sibling b immediately after a	●	●	●	●	●
a ~ b	General sibling	sibling b anywhere after a	●	●	●	●	●

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**).

```
article.chaptertext 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.

```
<article class="chaptertext"><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 **chaptertext** (**article.chaptertext**).


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:

```
<article class="chaptertext"><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 **chaptertext** class:


```
<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.

Code 4.1 The style is set for the emphasis tag if its parents are the **h1** tag and the article tag using the copy class .

```
<!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,
→ </strong> 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
→ <em>what</em> is the use of a book,</q> <strong>thought Alice</strong>, <q>without pictures or
→ conversations?</q></p>
</article>
</body>
</html>
```

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?"

 **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 descendants 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**).

```
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 descendants (children). However, the styles will not be directly applied to those children's descendants.

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

```
<article class="chaptertext"><p>  
→ <strong><em>...</em></strong>  
→ </p></article>
```

Or:

```
<article class="chaptertext"><p>  
→ <q><em>...</em></q></p></article>
```

However, an emphasis tag that is *not* in another tag in the paragraph will *not* be styled.

```
<article class="chaptertext"><p>  
→ <em>...</em></p></article>
```

TIP Like grouped selectors, contextual selectors can include class selectors (dependent or independent), ID selectors in the list, and HTML selectors.

Code 4.2 The style is set for the emphasis tag with *any* parent that's in an article tag using the copy class **D**.

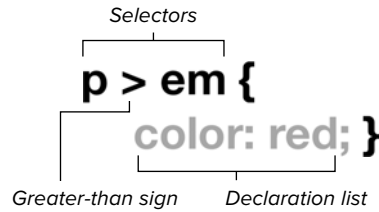
```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<title>Alice's Adventures in Wonderland</title>
<style type="text/css" media="all">
  article.chaptertext p * em {
    border: 1px double red;
    font-size: 2em;
    font-weight: normal; }
</style>
</head>
<body>
<article class="chaptertext">
<p><strong>Alice was beginning to get <em>very tired</em> of sitting by her sister on the bank,
→ </strong> 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
→ <em>what</em> is the use of a book,</q> <strong>thought Alice</strong>, <q>without pictures or
→ conversations?</q></p>
</article>
</body>
</html>
```

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?"

D 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 (>) **E**.



E The general syntax of the direct child selector.

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).

```
article.chaptertext > p > em {...}
```

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.

```
<article class="chaptertext"><p>  
→ <em>...</em></p></article>
```

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:

```
<article class="chaptertext"><p>  
→ <q><em>...</em></q><p></article>
```

Code 4.3 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 **F**.

```
<!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 > em {
    color: silver;
    background: red;
    font-size: 2em;
    font-weight: 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,
→ </strong> 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
→ <em>what</em> is the use of a book,</q> <strong>thought Alice</strong>, <q>without pictures or
→ conversations?</q></p>
</article>
</body>
</html>
```

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?"

F **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 **G** or occurs anywhere after that sibling **H**.

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**).

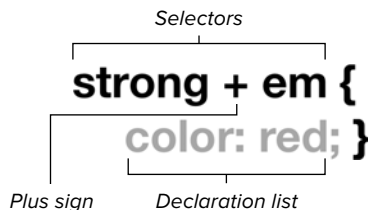
```
strong + em {...
```

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

```
<strong>...</strong>...<em>...</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:

```
<strong>...</strong>...<q>...</q>...  
→ <em>...</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 **I**.

```
<!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 <em>very tired</em> of sitting by her sister on the bank,
→ </strong> 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
→ <em>what</em> is the use of a book,</q> <strong>thought Alice</strong>, <q>without pictures or
→ conversations?</q></p>
</article>
</body>
</html>
```

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?"

I **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).

```
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.

```
<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 **strong** tags

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 4.2 Universal Selector Examples

Format	Elements Are Styled If...
a * b	b within a regardless of b's parents
a > * > b	b is the direct child of any element that is the direct child of a
a + * + b	sibling b immediately after any element that is immediately after a
*:hover	mouse pointer over any element
*:disabled	any element that is disabled
*:first-child	first child of any element
*:lang()	any element using specified language code
*:not(s)	any element that is not the using indicated selectors
*::first-letter	any element's first letter

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 **J**.

```
<!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 <em>very tired</em> of sitting by her sister on the bank,
→ </strong> 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
→ <em>what</em> is the use of a book,</q> <strong>thought Alice</strong>, <q>without pictures or
→ conversations?</q></p>
</article>
</body>
</html>
```

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?"

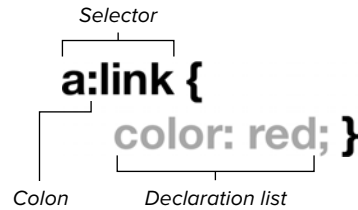
J **The results of Code 4.5.** The text in red matches the general sibling criteria—in this case the emphasis tags within the second, third, and fourth paragraphs.

★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 **A**.

- **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*.



A General syntax of a pseudo-class.

TABLE 4.3 Link and Dynamic Pseudo-Classes











Format	Selector Name	Elements Are Styled If...					
:link	Link	the value of href is not in history	●	●	●	●	●
:visited	Visited link	the value of href is in history	●	●	●	●	●
:target	Targeted link	a targeted anchor link	●	●	●	●	●
:active	Active	the element is clicked	●	●	●	●	●
:hover	Hover	the pointer is over the element	●	●	●	●	●
:focus	Focus	the element has screen focus	●	●	●	●	●

TABLE 4.4 Structural/Other Pseudo-Classes

Format	Selector Name	Elements Are Styled If...					
:root	Root	is the top level element in a document	●9	●	●	●	●
:empty	Empty	has no children	●9	●	●	●	●
:only-child	Only child	has no siblings	●9	●	●	●	●
:only-of-type	Only of type	has its unique selector among its siblings	●9	●	●	●	●
:first-child	First-child	is the first child of another element	●9	●	●	●	●
:nth-of-type(n)	Nth of type	is the <i>n</i> th element with that selector	●9	●	●	●	●
:nth-last-of-type(n)	Nth from last of type	is the <i>n</i> th element with that selector from the last element with that selector	●9	●	●	●	●
:last-child	Last child	is the last child in the parent element	●9	●	●	●	●
:first-of-type	First of type	is the first of its selector type in the parent element	●9	●	●	●	●
:last-of-type	Last of type	is the last of its selector type in the parent element	●9	●	●	●	●
:lang()	Language	has a specified language code defined	●8	●	●	●	●
:not(s)	Negation	is not using specific selectors	●9	●	●	●	●

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.

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

TOC:	
1. Down the Rabbit-hole	Link
2. The Pool of Tears	Visited
3. A Caucus-race and a Long Tale	Hover
4. The Rabbit sends in a Little Bill	Active
5. Advice from a Caterpillar	
6. Pig and Pepper	
7. A Mad Tea-party	
8. The Queen's Croquet-ground	
9. The Mock Turtle's Story	
10. The Lobster Quadrille	
11. Who Stole the Tarts?	
12. Alice's Evidence	

B The results of Code 4.6 show the links styled for each state to help the user understand what's going on.

Code 4.6 The link styles are set for the default and then all four link states, creating color differentiation **B**. Notice also that I've turned off underlining with text decoration but added an underline effect using border bottom.

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<title>Alice's Adventures in Wonderland</title>
<style type="text/css" media="all">
  a {
    text-decoration: none;
    font-size: 2em; }
  a:link {
    color: darkred;
    border-bottom: 1px solid red; }
  a:visited {
    color: darkred;
    border-bottom: 1px dashed red; }
  a:hover {
    color: red;
    border-bottom: 1px solid pink; }
  a:active {
    color: pink;
    border-bottom: 1px solid pink; }
</style>
</head>
<body>
<nav>
<h2>TOC:</h2>
<ol>
<li><a href="AAIWL-ch01.html">Down the Rabbit-hole</a></li>
<li><a href="AAIWL-ch02.html">The Pool of Tears</a></li>
<li><a href="AAIWL-ch03.html">A Caucus-race and a Long Tale</a></li>
<li><a href="AAIWL-ch04.html">The Rabbit sends in a Little Bill</a></li>
<li><a href="AAIWL-ch05.html">Advice from a Caterpillar</a></li>
<li><a href="AAIWL-ch06.html">Pig and Pepper</a></li>
<li><a href="AAIWL-ch07.html">A Mad Tea-party</a></li>
<li><a href="AAIWL-ch08.html">The Queen's Croquet-ground</a></li>
<li><a href="AAIWL-ch09.html">The Mock Turtle's Story</a></li>
<li><a href="AAIWL-ch010.html">The Lobster Quadrille</a></li>
<li><a href="AAIWL-ch011.html">Who Stole the Tarts?</a></li>
<li><a href="AAIWL-ch012.html">Alice's Evidence</a></li>
</ol>
</nav>
</body>
</html>
```

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**.

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**.

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**.

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**.

a:active {...}

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

...

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.

TIP You can apply the dynamic pseudo-classes `:hover`, `:active`, and `:focus` to any element, not just links.

TIP 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.

TIP 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.

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

TIP 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:

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

TIP 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.

```
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**.

```
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) .

```
<!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>
```

Default

Mailing List:

Hover

Mailing List:

Active

Mailing List:

Focus

Mailing List:

C The results of Code 4.7. This shows a simple form field in the four dynamic states. Providing this visual feedback can help users know which form field is ready for use or that they have clicked a button.

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.

3. Style the *focus* state of the element.

Type the selector (HTML, class, or ID), a colon (:), and then **focus**.

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**.

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.

```
<input type="button" value=
  → "Submit">
```

All the tags using the specific selector will have their states styled.

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.

Code 4.8 The list has styles set based on the location within the list **D**.

```
<!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>
```

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-of-type(3) {...}

li:nth-last-of-type(2) {...}

li:last-child {...}

li:last-of-type {...}

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

...

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

1. Down the Rabbit-hole

2. The Pool of Tears

3. A Caucus-race and a Long Tale

4. The Rabbit sends in a Little Bill

5. Advice from a Caterpillar

6. Pig and Pepper

7. The Queen's Croquet-ground

8. The Mock Turtle's Story

9. The Lobster Quadrille

10. Who Stole the Tarts?

11. Alice's Evidence

D 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**).

```
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.

```
<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.

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) **E**.

```
<!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="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 regarda 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>
```

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?"

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 regarda dans le livre de sa sœur lisait, mais il n'avait pas d'images ni dialogues en elle, «et ce qui est l'utilisation d'un livre,» pensait Alice, «sans images ni dialogues?»

E 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 **F**.

```
<!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 regarda 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>
```

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?"

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 regarda dans le livre de sa sœur lisait, mais il n'avait pas d'images ni dialogues en elle, «et ce qui est l'utilisation d'un livre,» pensait Alice, «sans images ni dialogues?»

F **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

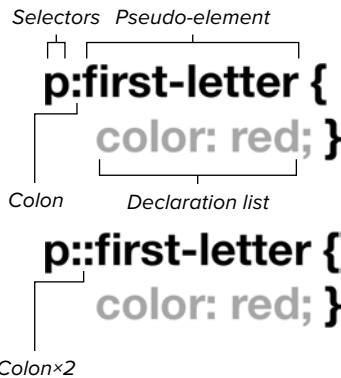
You can access the first letter of any block of text directly using the `:first-letter` pseudo-element. The first line of any block of text can be isolated for style treatment using the `:first-line` pseudo-element **A**.

To highlight the beginning of an article:

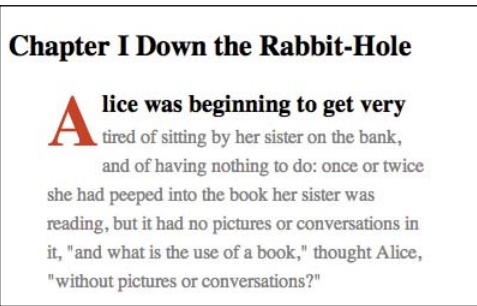
- 1. Style the default version of the element.

```
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).








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.



B 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.

TABLE 4.5 Pseudo-Elements

Format	Selector Name	Elements Are Styled If...					
<code>:first-letter, ::first-letter</code>	the first letter	first letter in text	●	●	●	●	●
<code>:first-line, ::first-line</code>	the first line of text	they are the first line of text	●	●	●	●	●
<code>:after, ::after</code>	After	space immediately before element	●8	●	●	●	●
<code>:before, ::before</code>	Before	space immediately after element	●8	●	●	●	●

Code 4.11 Styles are set for the first letter and first line of the first paragraph in an article **B**.

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<title>Alice&#8217;s Adventures in
→ Wonderland</title>
<style type="text/css" media="all">
  article p {
    color: gray;
    font-size: 1em;
    line-height: 1.5;
    margin: .875em 2em;
  }
  article p:first-of-type::first-letter {
    color: red;
    font-size: 3em;
    float: left;
    margin: -.25em .05em 0 0; }
  article p:first-of-type::first-line {
    color: black;
    font-size: 1.25em;
    font-weight: bold; }
</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,
→ <q>and what is the use of a book,</q>
→ thought Alice, <q>without pictures or
→ conversations</q></p>
</article>
</body>
</html>
```

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.

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 {...}**

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 **C** in this case—to the page header **D**.

```
<!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
  <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, <q>
→ and what is the use of a book,</q>
→ thought Alice, <q>without pictures or
→ conversations?</q></p>
</article>
</body>
</html>
```



C bullet-01.png & bullet-02.png will be used as flourishes around titles.

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?"

So she was considering in her own mind (as well as she could, for the hot day made her feel very sleepy and stupid) whether the pleasure of making a daisy-chain would be worth the trouble of getting up and picking the daisies, when suddenly a White Rabbit with pink eyes ran close by her.

There was nothing so *very* remarkable in that; nor did Alice think it so *very* much out of the way to hear the Rabbit say to itself, "Oh dear! Oh dear! I shall be too late!" (when she thought it over afterwards, it occurred to her that she ought to have wondered at this, but at the time it all seemed quite natural); but when the Rabbit actually *took a watch out of its waistcoat-pocket*, and looked at it, and then hurried on, Alice started to her feet, for it flashed across her mind that she had

D The header now has a bit of flourish added before and after by the CSS. These images take up space as if they were in an image tag, but do not show up in the HTML code.

- 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**.

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

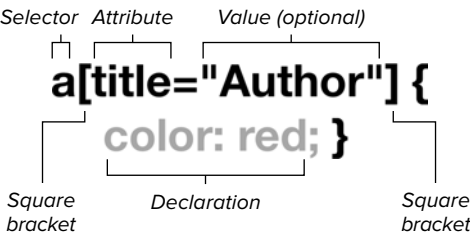
Although style attributes should all be handled by CSS, many HTML tags still have attributes that define how they behave. For example, the image tag, `img`, always includes the `src` attribute to define the source for the image file to be loaded.

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**.

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








A The general syntax of an attribute selector.

About The Book:

- [Alice’s Adventures in Wonderland](#)
- [Lewis Carroll](#)
- [John Tenniel](#)
- [Arthur Rackham](#)
- [Download Examples](#)
- [More Info](#)
- [Order The Book](#)

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

TABLE 4.6 Attribute Selectors

Format	Name	Elements Are Styled If That Element:					
[attr]	Attribute	has specified attribute	●	●	●	●	●
[attr="value"]	Exact value	has specified attribute equal to exact value	●	●	●	●	●
[attr~="value"]	Spaced list	has specified attribute equal to exact value within space-separated list	●	●	●	●	●
[attr ="value"]	Hyphenated list	has specified attribute equal to exact value within hyphen-separated list	●	●	●	●	●
[attr^="value"]	Begins with	has specified attribute equal to exact value at beginning	●	●	●	●	●
[attr\$="value"]	Ends with	has specified attribute equal to exact value at end	●	●	●	●	●
[attr*="value"]	Contains	has specified attribute equal to exact value anywhere	●	●	●	●	●

Code 4.13 HTML tags can have different attributes, and you can add styles to an element based on its attributes **B**.

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<title>Alice&#8217;s Adventures in
→ Wonderland</title>
<style type="text/css" media="all">
  a { display: block; font-size: 2em;}
  a[title] { color: red; }
  a[title="Author"] {color: orange; }
  a[title~="white"] { color: yellow; }
  a[title="illustrations"] { color:
→ green; }
  a[href^="http://"] {color: blue; }
  a[href*="order"] {color: indigo; }
  a[href$=".css3-vqs"] {color: violet; }

</style>
</head>
<body>
<article class="chaptertext">
<h1>About The Book:</h1>
<ul>
  <li><a href="index.html" title="Alice's
→ Adventures in Wonderland">
→ Alice&#8217;s Adventures in
→ Wonderland</a></li>
  <li><a href="index.html"
→ title="Author">Lewis Carroll</a></li>
  <li><a href="index.html" title=
→ "illustrations black white">John
→ Tenniel</a></li>
  <li><a href="index.html" title=
→ "illustrations-full-color">Arthur
→ Rackham</a></li>
  <li><a href="http://www.jasonspeeking
→ .com">Download Examples</a></li>
  <li><a href="http://www.jasonspeeking
→ .com/css3-vqs/order">More Info</a>
→ </li>
  <li><a href="http://www.jasonspeeking
→ .com/css3-vqs">Order The Book</a></li>
</article>
</body>
</html>
```

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.

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 (]).

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 caret (^), 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 **A** 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.

continues on page 106






TABLE 4.7 Media Values

Value	Intended for
screen	Computer displays
tty	Teletypes, computer terminals, and older portable devices
tv	Television displays
projection	Projectors
handheld	Portable phones and PDAs
print	Paper
braille	Braille tactile readers
speech	Speech synthesizers
all	All devices

media="
screen
and (min-width: 740px)
and (max-width: 980px)"

A The general syntax for media queries.

TABLE 4.8 Media Query Properties

Property	Value					
aspect-ratio	<ratio>	●9	●	●	●	●
max-aspect-ratio	<ratio>	●9	●	●	●	●
min-aspect-ratio	<ratio>	●9	●	●	●	●
device-aspect-ratio	<ratio>	●9	●	●	●	●
max-device-aspect-ratio	<ratio>	●9	●	●	●	●
min-device-aspect-ratio	<ratio>	●9	●	●	●	●
color	<integer>	●9	●	●	●	●
max-color	<integer>	●9	●	●	●	●
min-color	<integer>	●9	●	●	●	●
color-index	<integer>	●9	●	●	●	●
max-color-index	<integer>	●9	●	●	●	●
min-color-index	<integer>	●9	●	●	●	●
device-height	<length>	●9	●	●	●	●
max-device-height	<length>	●9	●	●	●	●
min-device-height	<length>	●9	●	●	●	●
device-width	<length>	●9	●	●	●	●
max-device-width	<length>	●9	●	●	●	●
min-device-width	<length>	●9	●	●	●	●
height	<length>	●9	●	●	●	●
max-height	<length>	●9	●	●	●	●
min-height	<length>	●9	●	●	●	●
monochrome	<integer>	●9	●	●	●	●
max-monochrome	<integer>	●9	●	●	●	●
min-monochrome	<integer>	●9	●	●	●	●
orientation	portrait, landscape	●9	●	●	●	●
resolution	<resolution>	●9	●	●	●	●
max-resolution	<resolution>	●9	●	●	●	●
min-resolution	<resolution>	●9	●	●	●	●
scan	progressive, interlaced	●9	●	●	●	●
width	<length>	●9	●	●	●	●
max-width	<length>	●9	●	●	●	●
min-width	<length>	●9	●	●	●	●

- **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.

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.

```
/**/ 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.

```
/**/ 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.

```
/**/ 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; }
```

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).

continues on next page

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.

```
/**/ For Print ***/

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.

```
<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.

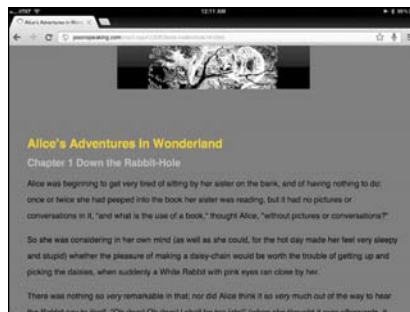
3. Link to your *default* style sheet. In the head of your HTML document (Code 4.18), type a <link> tag that references the default version of the CSS and define **media** as **all**.

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

continues on page 110



- 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 **B**, tablet **C**, smart phone **D**, and print **E**. 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.

```
<!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="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 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>
</article>
<footer><nav> Next:
<a class="chaptertitle" href="AAIWL-ch02.html">The Pool of Tears</a>
</nav></footer>
</body>
</html>
```



D **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.



E **Code 4.18** output to a printer. The background is white, and the background image is no longer inverted. This works better in print.

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**.

```
<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**.

```
<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 **F** 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.

@media screen and

→ **(max-device-width: 480px) {...}**

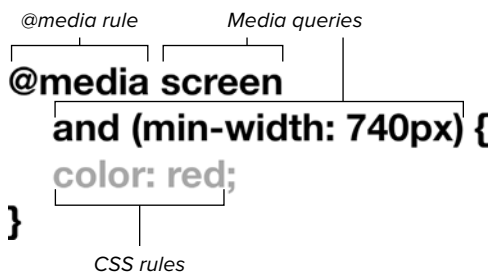
For example, you might specify that these styles are for screens with a width up to 480px wide. Finish with curly brackets. Add any media-specific styles between the curly brackets.

3. **Add other styles as necessary.**

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`.

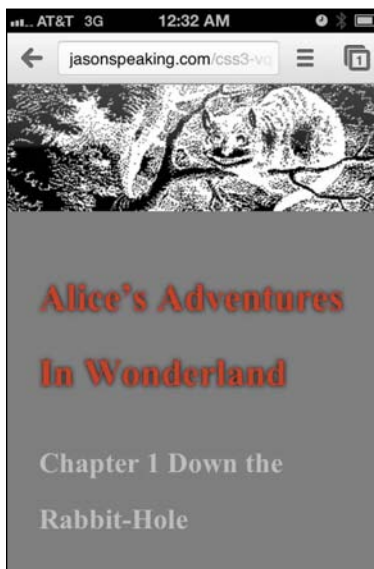
TIP Remember that `@media` rules can go in external or embedded style sheets.



F The general syntax of the `@media` rule.



G Code 4.19 on a computer screen.



H Code 4.19 on a mobile device screen. The styles and background have been modified based on the device width.

Code 4.19 The HTML code links to the various style sheets for different media types **G** and **H**.

```
<!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, "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 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>
</article>
<footer><nav> Next:
<a class="chaptertitle" href="AAIWL-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 a

```
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.

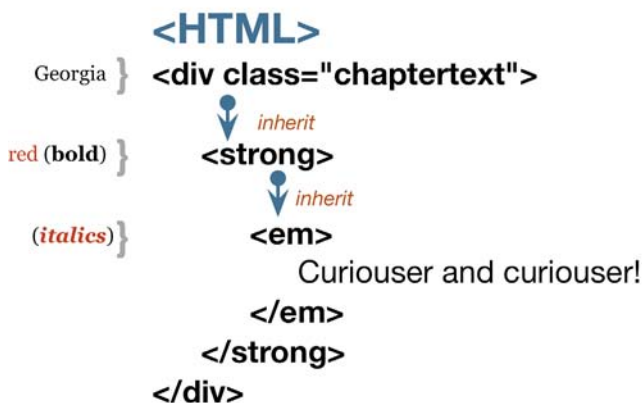
Inheriting Properties from a Parent

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 **A**.

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; }
.chaptertext { font-family: georgia; }
```



Results

Curiouser and curiouser!

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.

Selector

h1 {
 color: red; !important }

Declaration !important

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.

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<title>Alice&#8217;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, <q>and what is the
→ use of a book,</q> thought Alice, <q>
→ without pictures or conversations?</q></p>
</article>
</body>
</html>
```

Making a Declaration **!important**

You can add the **!important** declaration to a property-value declaration to give it the maximum weight when determining the cascade order **A**. 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).

h2 {...}

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.

color: green !important;

continues on next page

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?"

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.

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 **A**.

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

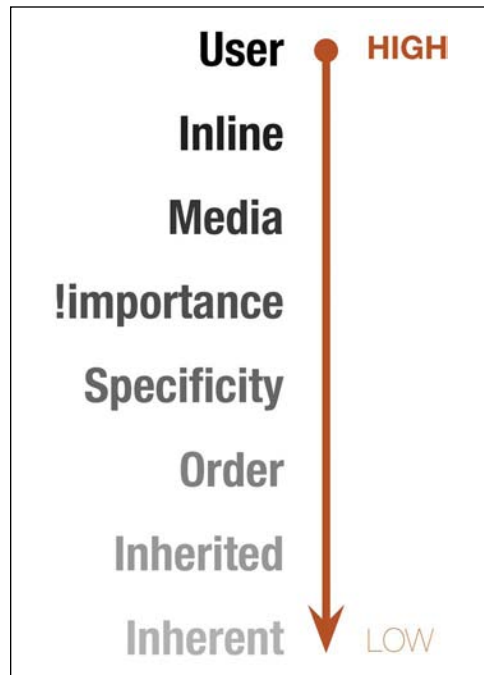
Obviously, if the media type is set for a style and element that is not being displayed in that media type, the style will not be used.

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,

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

is worth 102, whereas

```
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

Index

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, 251
 - fixing box model for older versions of IE, 338
 - 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
- both** keyword, **clear**, 264
- 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

C

- 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
- color
 - best practices, 404–405
 - emotional associations of color, 196
 - gradients. *See* gradients, color
 - links, 86–87
 - new in CSS3, 14
 - for readable style sheets, 378–379
 - shadows, 302
 - text drop shadows, 166–167
 - transitioning CSS properties, 320–324
- color palette, 196, 200
- color properties
 - accessibility for visually impaired, 201
 - choosing color palette, 196–201
 - choosing color values, 185–190
 - creating color gradients, 191–195
 - getting started, 184
 - other ways to add color, 190
 - overview of, 183
 - putting it all together, 217–218
 - quick reference, 414
- color** property, 106, 202
- Color Scheme Designer tool, 200
- color stop, 192, 194
- color values
 - alpha values for transparency, 190
 - backgrounds, 204
 - borders, 270–271
 - for color keywords, 185–187
 - color wheel, 198
 - HSL, 189
 - multiple background images, 213
 - overview of, 185
 - RGB, 188–189
 - shadows, 303
 - text, 202
- color wheel
 - basics, 198–200
 - online tool for advanced, 200
- color-index** property, media queries, 106
- 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

- combinatory selectors, 71
- comments
 - adding to CSS, 67–68
 - best practices, 406
 - grouping selectors with, 64–66
 - section headers as, 378–379
 - setting up conditional styles for IE, 364–365
- compact** property, 254
- complementary color-combination scheme, 199
- compound color-combination scheme, 199
- compression, CSS code, 396–398
- condensed fonts, 150
- conditional styles, Internet Explorer
 - fixing box model for older versions, 338
 - overview of, 363–365
 - responsive Web design, 353, 370–373
- content. *See also* generated content properties
 - adding using CSS, 240–241
 - background color, 196–197
 - controlling overflow, 259–260
 - defining background image, 208, 211
 - of element's box, 250
 - new features in CSS3, 15
 - progressive enhancement and, 355
 - setting how box sizes, 258
 - styling for print, 114
- content** property, 240, 242–243, 250–251
- content-box** value, **box-sizing**, 258
- contextual selectors, 70
- contextual styles
 - descendants, 71–75
 - only children, 76–77
 - overview of, 71
 - siblings, 78–81
- converting licensed fonts, 140
- copy, color(s) for, 197
- counter lists, multiple, 242–243
- couplet values, RGB hex, 188
- CSS (Cascading Style Sheets), overview
 - browser extensions, 12–13
 - defined, 1
 - evolution of, 6–7
 - HTML and, 8
 - libraries and frameworks, 376
 - rule parts, 11
 - styles, 2
 - types of rules, 9–10
 - understanding, 3–5
 - what's new in CSS3, 14–15
 - word processor styles vs., 3
- CSS basics
 - basic selectors, 36
 - comments, 67–68
 - embedded styles, 40–42
 - external styles, 43–49
 - grouping, 64–66
 - HTML tags, 50–52
 - inline styles, 37–39
 - overview of, 35
 - 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
- curly brackets { }, CSS rules
 - class selectors, 53
 - embedded styles, 41
 - external CSS file, 45
 - HTML tags, 50–52
 - troubleshooting CSS code, 386
- 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
- ** tag, text strikethrough vs., 180
- dependent class selector, 36, 54
- dependent ID selector, 36, 60
- descendants, 70–71, 115–116

- 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
- <div>** 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
- 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
- ** tag, 20, 72–81, 144–145, 160, 174
- Elastic 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 UTL 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

- document, 287
- element, 287
- of element's box, 250
- fixing box model for older versions of IE, 337–338
- viewport, 287
- height** property, 106, 250–251, 255–257
- hex values, 185–189
- <hgroup>** tag, HTML5, 28
- hidden** keyword, 260, 296
- horizontal alignment, centering numbers in blocks, 336
- horizontal text alignment, 171–173
- hotspot, mouse pointer, 239
- :hover** pseudo-class, 84–89, 167
- HSL (hue, saturation, and lightness) values, 189–190, 198
- HSVA value, adding text color with, 203
- HTML (HyperText Markup Language)
 - basic document structure, 19
 - browser inherited styles and, 20
 - CSS and, 3–5, 8
 - CSS vs., 1
 - defined, 17
 - properties, 19–20
 - selectors vs. attributes, 11
 - types of elements, 21–23
 - understanding, 18
 - UTL character encoding and, 423–428
- HTML selectors
 - CSS browser extensions working with, 13
 - CSS rules, 9–11
 - determining cascade order by specificity, 121
 - redefining HTML tags, 50–52
 - styling HTML elements, 21–23, 36
- HTML4, 24
- HTML5
 - elements, 21–23
 - evolution of, 24–26
 - new features, 27
 - overview of, 17
 - structure of, 28–33
 - understanding, 18–20
- HTML5 Shiv, 32–33
- hue, HSL values, 189–190, 198
- hyperlinks
 - setting negative margins, 266
 - turning off underlining on, 180
 - using color change to show states of, 203
 - using with footnotes, 174

I

- icon**: keyword, 152
- icons
 - CSS sprites for, 342–344
 - dingbats, 129
 - Web fonts for. See Web fonts
- id** attribute, unique IDs, 58

- 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
- ** 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
- inheritance of 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/TTF 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 Trouble Girl font, 139
local versions, Web font service bureau, 143
logical operators, conditional styles for IE, 364
LoVe HAte mnemonic, pseudo-element order, 89
lowercase text, 164

M

- Mac fonts, online resource, 139
- margin** property, 251, 265–267
- margins
 - collapse of, 267
 - CSS resets for, 359
 - of element's box, 251
 - favoring padding over, 405
 - indenting paragraphs and, 176
 - setting element's, 265–267
 - setting in body tag, 291
 - table cells not using, 229
- markup languages, 19, 29
- matrix()** value, 2D transforms, 314
- matrix3d()** value, 3D transforms, 319
- max-width** property, 257
- media queries
 - best practices, 402
 - embedding into style sheets, 112–113
 - multi-screens with, 357
 - new in CSS3, 15
 - overview of, 104–106
 - in responsive Web design, 352, 370–373
 - specifying styles, 107–111
- @media rule
 - best practices, 402
 - placing at top of CSS code, 378
 - specifying styles with, 112–113
- media type
 - determining cascade order value, 120
 - specifying style sheets for particular, 47
 - values, 104
- medium.css, 366–368
- menu:** keyword, 152
- message-box:** keyword, 152
- Meyer's CSS reset, 362
- mid-width** property, 257
- Minify CSS Compressor, 396–398
- minifying your CSS code, 396–398, 407
- mobile devices
 - adapting to environment, 366–373
 - CSS not accommodated by many browsers, 5
 - media queries specifying styles for, 107–110, 370–373
 - multi-screen strategy for, 357
 - progressive enhancement across multiple, 354–356
- monochromatic color schemes, 198
- monochrome color schemes, 198
- monochrome property, media queries, 106
- monospace fonts, 127–128
- mouse pointer, changing appearance, 238–239
- moz- extension, CSS, 12–13
- ms- extension, CSS, 12–13
- multicolumn layouts
 - with **float**, 330–333
 - for text, 278–279

- multiple backgrounds
 - adding with **background** shorthand, 212–216
 - layering images, 205
- multi-screens, 357

N

- 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
 - 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
- Reset CSS, Yahoo, 361
- 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 **rotate(Y)** 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
 - 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 **scale(Y)** 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
 - combinatory, 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 **skew(Y)** 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
- ** 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
- ** tag
 - aligning text vertically, 174
 - defining font size, 145
 - as nested tag, 70
 - redefining in CSS, 8, 50–52
 - styling descendants, 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
- <style>** 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
- 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 vs. 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
 - getting 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
- translate3d()** 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

U

- 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
 - getting 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

V

- validation, CSS code, 395
- values
 - 2D transform, 311, 313–315
 - 3D transform, 318–319
 - clip**, 298–299
 - color, 185–190
 - content**, 241
 - CSS browser extensions working with, 13
 - cursor**, 238–239
 - defining styles based on tag attributes, 100–103
 - defining styles in HTML tag, 38
 - display type, 252–254
 - font-stretch**, 150
 - grouped selector changing, 66
 - indenting paragraphs, 176
 - letter-spacing**, 159–160
 - linear gradient, 191
 - line-height**, 162–163
 - media type, 104
 - multiple font, 152–154
 - placing at top of CSS code in comments, 67–68
 - position, 292–293
 - radial gradient, 192
 - specifying units for, 399
 - structural elements of CSS rules, 11
 - text decorations, 179–180
 - text transform, 164–165
 - transformations, 311, 316
 - transitions, 323–324
 - troubleshooting CSS code, 386
 - vertical text alignment, 174–175
 - word-spacing**, 161
- vertical alignment
 - centering numbers in blocks, 336
 - CSS resets for text, 359
 - text, 174–175
- vertical-align** property, 174–175
- viewport
 - browser, 286
 - responsive design with media queries, 370–372

- 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
- World Wide Web Consortium. *See* W3C (World Wide Web Consortium)
- 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



WATCH READ CREATE

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

creative
edge