

VISUAL **QUICKSTART** GUIDE

Get up and running in no time!



*Dozens of downloadable
code samples!*

CSS**3**

JASON CRANFORD TEAGUE

© LEARN THE QUICK AND EASY WAY!

Visual QuickStart Guide

CSS3

Jason Cranford Teague

Peachpit Press
1249 Eighth Street
Berkeley, CA 94710
510/524-2178
510/524-2221 (fax)

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 © 2011 by Jason Cranford Teague

Project Editor: Nancy Peterson
Development Editor: Bob Lindstrom
Copyeditors: Anne Marie Walker, Liz Merfeld
Technical Editor: Chris Mills
Production Editor: Cory Borman
Compositor: Danielle Foster
Indexer: Jack Lewis
Cover design: Peachpit Press

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: 978-0-321-71963-8

ISBN 0-321-71963-8

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 and **Anne Marie**, who dotted my i's and made sure that everything made sense.

Chris, who held my feet to the fire on every line of code.

Danielle and **Cory**, who put the book together with great patience and made it look pretty.

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, for his incredible illustrations of *Alice's Adventures in Wonderland*.

Douglas Adams, Neil Gaiman, Philip K. Dick, and Carl Sagan whose writings inspire me every day.

BBC 6 Music, The Craig Charles Funk and Soul Show, Rasputina, Stricken City, Groove Armada, Electrocute, Twisted Tongue, Bat for Lashes, Cake, Client, Jonathan Coulton, Cracker, Nine Inch Nails, 8mm, KMFDM, Nizlopi, the Pogues, Ramones, New Model Army, Cocteau Twins, the Sisters of Mercy, the Smiths, Mojo Nixon, Bauhaus, Lady Tron, David Bowie, Falco, T. Rex, Bad Religion, The National, Dr. Rubberfunk, Smoove and Turell, JET, Depeche Mode, Ian Dury, The Kinks, This Mortal Coil, Rancid, Monty Python, the Dead Milkmen, New Order, Regina Spektor, The Sex Pistols, Dead Can Dance, 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	xv
Chapter 1	Understanding CSS3	1
Chapter 2	HTML5 Primer	15
Chapter 3	CSS Basics	33
Chapter 4	Selective Styling	69
Chapter 5	Font Properties	117
Chapter 6	Text Properties	151
Chapter 7	Color and Background Properties	179
Chapter 8	List and Table Properties	213
Chapter 9	User Interface and Generated Content Properties	229
Chapter 10	Box Properties	241
Chapter 11	Visual Formatting Properties	279
Chapter 12	Transformation and Transition Properties	303
Chapter 13	Fixing CSS	323
Chapter 14	Essential CSS Techniques	343
Chapter 15	Managing Style Sheets	361
Appendix A	CSS Quick Reference	393
Appendix B	HTML and UTF Character Encoding	407
	Index	413

This page intentionally left blank

Table of Contents

	Introduction	xv
Chapter 1	Understanding CSS3	1
	What Is a Style?	2
	What Are Cascading Style Sheets?	3
	How does CSS work?	4
	The Evolution of CSS	6
	CSS Level 1 (CSS1)	7
	CSS Level 2 (CSS2)	7
	CSS Level 3 (CSS3)	7
	CSS and HTML.	8
	Types of CSS Rules	9
	The Parts of a CSS Rule.	11
	Browser CSS extensions	11
	New in CSS3	13
Chapter 2	HTML5 Primer	15
	What Is HTML?	16
	Basic HTML Document Structure	17
	HTML Properties.	17
	HTML and CSS.	18
	Types of HTML Elements	19
	The Evolution of HTML5	22
	And then came XHTML	22
	The problem with XHTML2	23
	And then there was HTML5.	23
	Is it HTML5 or XHTML5?	24
	What's New in HTML5?	25
	How Does HTML5 Structure Work?	26
	Using HTML5 Structure Now.	27
	Making HTML5 work in Internet Explorer	30
	Putting It All Together.	32

Chapter 3	CSS Basics	33
	The Basic CSS Selectors	34
	Inline: Adding Styles to an HTML Tag	35
	Embedded: Adding Styles to a Web Page	38
	External: Adding Styles to a Web Site.	41
	Creating an external style sheet	41
	Linking to a style sheet	44
	Importing a style sheet	46
	(Re)Defining HTML Tags	48
	Defining Reusable Classes.	51
	Defining Unique IDs.	55
	Defining Universal Styles.	59
	Grouping: Defining Elements Using the	
	Same Styles.	62
	Adding Comments to CSS	66
	Putting It All Together.	68
Chapter 4	Selective Styling	69
	The Element Family Tree	70
	Defining Styles Based on Context.	71
	Styling descendents.	71
	Styling only the children	74
	Styling siblings	76
	Working with Pseudo-classes	80
	Styling links.	82
	Styling for interaction	86
	NEW IN CSS3: Styling specific children with	
	pseudo-classes ★	88
	Styling for a particular language	89
	NEW IN CSS3: <i>Not</i> styling an element ★	91
	Working with Pseudo-elements	92
	Working with first letters and lines.	92
	Setting content before and after an element.	94
	Defining Styles Based on Tag Attributes	96
	NEW IN CSS3: Querying the Media ★	100
	Media queries	100
	Using the @media rule	106
	Inheriting Properties from a Parent	109
	Managing existing or inherited property values	110

	Making a Declaration !important.	111
	Determining the Cascade Order.	113
	Putting It All Together.	116
Chapter 5	Font Properties	117
	Understanding Typography on the Web	119
	Specifying the character set	119
	Generic font families	120
	Dingbats	122
	HTML character entities.	123
	Setting a Font-Stack.	124
	Finding Fonts	126
	Web-safe fonts.	126
	Downloadable Webfonts	127
	Setting a better font stack	128
	Setting the Font Size	133
	NEW IN CSS3: Adjusting Font Size for	
	Understudy Fonts ★	136
	Making Text Italic	139
	Setting Bold, Bolder, Boldest.	142
	Creating Small Caps.	144
	Setting Multiple Font Values	146
	Putting It All Together.	150
Chapter 6	Text Properties.	151
	Adjusting Text Spacing	153
	Adjusting the space between letters (tracking)	153
	Adjusting space between words.	155
	Adjusting space between lines of text (leading)	156
	Setting Text Case	158
	NEW IN CSS3: Adding a Text Drop Shadow ★	160
	Aligning Text Horizontally	162
	Aligning Text Vertically	164
	Indenting Paragraphs	167
	Controlling White Space	169
	Decorating Text	172
	Coming Soon!	175
	Putting It All Together.	177

Chapter 7	Color and Background Properties	179
	Choosing Color Values	181
	Color keywords	181
	RGB hex values	182
	RGB decimal values	182
	Percentage values	183
	New in CSS3: HSL values ★	183
	New in CSS3: Color alpha values ★	183
	New in CSS3: Color Gradients in Backgrounds ★	187
	Internet Explorer gradients	187
	Mozilla gradients	188
	Webkit gradients	189
	Choosing Your Color Palette	191
	Color wheel basics	194
	Online color scheme tools	195
	Setting Text Color	196
	Setting a Background Color	198
	Setting a Background Image	200
	Using Background Shorthand	208
	Putting It All Together	212
Chapter 8	List and Table Properties	213
	Setting the Bullet Style	216
	Creating Your Own Bullets	217
	Setting Bullet Positions	218
	Setting Multiple List Styles	219
	Setting the Table Layout	220
	Setting the Space Between Table Cells	222
	Collapsing Borders Between Table Cells	223
	Dealing with Empty Table Cells	225
	Setting the Position of a Table Caption	226
	Putting It All Together	227
Chapter 9	User Interface and Generated Content Properties	229
	Changing the Mouse Pointer Appearance	232
	Adding Content Using CSS	234
	Teaching the Browser to Count	236
	Specifying the Quote Style	238
	Putting It All Together	240

Chapter 10	Box Properties	241
	Understanding an Element's Box	245
	Parts of the box	246
	Displaying an Element	248
	Setting the Width and Height of an Element	251
	Controlling Overflowing Content	254
	Floating Elements in the Window	257
	Clearing a floated element	258
	Setting an Element's Margins	260
	Setting an Element's Outline	263
	Setting an Element's Border	265
	NEW IN CSS3: Rounding Border Corners ★	268
	NEW IN CSS3: Setting a Border Image ★	271
	Setting an Element's Padding	274
	Coming Soon!	276
	Putting it All Together	277
Chapter 11	Visual Formatting Properties	279
	Understanding the Window and Document	283
	Setting the Positioning Type	285
	Static positioning	285
	Relative positioning	286
	Absolute positioning	286
	Fixed positioning.	287
	Setting an Element's Position	290
	Stacking Objects in 3D	292
	Setting the Visibility of an Element	294
	Clipping an Element's Visible Area	296
	NEW IN CSS3: Setting an Element's Opacity ★	298
	NEW IN CSS3: Setting an Element's Shadows ★	300
	Putting It All Together	302
Chapter 12	Transformation and Transition Properties	303
	NEW IN CSS3: Transforming an Element ★	307
	2D transformations	308
	3D transformations	311
	NEW IN CSS3: Adding Transitions Between Element States ★	316
	What can be transitioned?	316
	Putting It All Together	321

Chapter 13	Fixing CSS	323
	Adjusting CSS for Internet Explorer	324
	The underscore hack	325
	IE conditional CSS	328
	Fixing the Internet Explorer Box Model	333
	Resetting CSS	335
	A simple CSS reset	336
	YUI2: Reset CSS	337
	Eric Meyer's reset	338
	Fixing the Float	340
	Break tag clear all fix	340
	Overflow fix	342
Chapter 14	Essential CSS Techniques	343
	Creating Multicolumn Layouts with Float	346
	Styling Links Versus Navigation	350
	Using CSS Sprites	354
	Creating a CSS Drop-down Menu	357
Chapter 15	Managing Style Sheets	361
	Creating Readable Style Sheets	362
	Include an introduction and TOC	362
	Define colors, fonts, and other constants	362
	Use section headers	364
	The @ rules go at the top	364
	Choose an organization scheme	365
	Use specificity for hierarchy	366
	CSS Libraries and Frameworks	367
	Style Sheet Strategies	368
	The One For All method	368
	The Divide and Conquer method	369
	The Aggregate method	370
	The Dynamic method	371
	Troubleshooting CSS Code	372
	Ask these questions	372
	If all else fails, try these ideas	375
	Debugging CSS in Firebug and Web Inspector	376
	Firebug for Firefox	377
	Web Inspector in Safari and Chrome	379
	Validating Your CSS Code	381

Minifying Your CSS	382
32 CSS Best Practices	385
Appendix A CSS Quick Reference.	393
Basic Selectors	394
Pseudo-Classes	395
Pseudo-Elements	395
Text Properties.	396
Font Properties	397
Color and Background Properties.	398
List Properties	399
Table Properties	399
User Interface and Generated	
Content Properties.	400
Box Properties	401
Visual Formatting Properties.	404
Transform Properties	
(-webkit-, -moz-, -o-)	405
Transition Properties	
(-webkit-, -moz-, -o-)	406
Appendix B HTML and UTF Character Encoding.	407
HTML and UTF Character Encoding.	408
Index	413

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 `<bx>` tag. That said, the more knowledge of HTML you bring to this book, the more you'll get out of it.

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

What tools do you need for this book?

The great thing about CSS and DHTML is that, like HTML, they don't require any special or expensive software. Their 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.

Browser-safe Colors?

Certain colors always display properly on any monitor. These colors are called browser-safe colors. You'll find them fairly easy to remember because their values stay consistent. In hexadecimal values, you can use any combination of 00, 33, 66, 99, CC, and FF. In numeric values, use 0, 51, 102, 153, 204, or 255. In percentages, use 0, 20, 40, 60, 80, or 100.

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 and CSS levels compatible with those values **A**. The Compatibility column displays the first browser version that supported the value type. **Table i.4** lists the browser abbreviations used in this book. Keep in mind, though, that even if the value is available in a particular version of the browser, it may not be available for all operating systems.

TABLE I.4 Browser Abbreviations

Abbreviation	Browser
IE	Microsoft Internet Explorer
FF*	Mozilla Firefox
O	Opera
S	Apple Safari
C	Google Chrome

* Includes other Mozilla-based browsers: Camino and Flock

Cursor Values	
Value	Compatibility
<cursor name>	IE4, N6, CSS2
<URL>	CSS2
auto	IE4, N6, CSS2

Values supported by this property (bracketed on the left side of the table)

Version of CSS where this value was introduced (line pointing to the CSS2 entry in the Compatibility column)

Earliest version of the browser in which this value is supported (line pointing to the IE4, N6, CSS2 entry in the Compatibility column)

A The property tables in Part 1 of this book 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>
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 then 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 about 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.speaking-in-styles.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	80
Working with Pseudo-elements	92
Defining Styles Based on Tag Attributes	96
NEW IN CSS3: Querying the Media ★	100
Inheriting Properties from a Parent	109
Making a Declaration !important	111
Determining the Cascade Order	113
Putting It All Together	116

The Element Family Tree

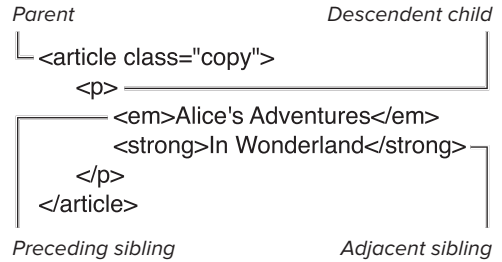
When a tag is surrounded by another tag—one inside another—the tags are *nested*.

`<h2>Chapter 2 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' *descendants*. 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 descendants. 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.

Space-separated
list of selectors

Declaration List

```
.copy h1 em { color: red; }
```

A The general syntax for the descendent selector.

Space separated
list of selectors

Declaration list

```
.copy * em { color: red; }
```

Universal selector

B The general syntax for the descendent selector using the universal selector.

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 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 Combinator Selectors

Format	Selector Name	Elements Are Styled If...	Compatibility
a b c	Descendent	c descendent of b descendent of a	IE4, FF1, O3.5, S1, C1, CSS1
a * b	Universal	b within a regardless of b's parents	IE7, FF1, O4, S1, C1, CSS2
a > b	Direct Child	b direct child of a	IE7 FF1, O3.5, S1, C1, CSS1
a + b	Adjacent Sibling	sibling b immediately after a	IE7, FF1, O5, S1, C1, CSS2
a ~ b	General Sibling	sibling b anywhere after a	IE8, FF1, O5, S1, C1, CSS2

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.copy h1 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 if the pattern is matched.

```
<article class="copy">  
→ <h1><em>...</em></h1></article>
```

The style will be applied if and only if the final selector occurs as a descendent nested within the previous selectors.

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 **C**.

```
<!-- HTML5 -->  
<!DOCTYPE html>  
<html lang="en">  
<head>  
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />  
<title>Alice's Adventures in Wonderland</title>  
<style type="text/css" media="all">  
  article.copy h1 em {  
    color: red;  
    font-weight: bold;  
    font-style: italic; }  
</style>  
</head>  
<body>  
<article class="copy">  
  <h1>Alice's Adventures in <em>Wonderland</em></h1>  
  <h2><em>Chapter 2.</em> The Pool of Tears</h2>  
  <p>'Curiouser and curiouser!' <em>cried</em> Alice,...</p>  
  <p>And she went on <em>planning</em>,...</p>  
  <p>Poor <em>Alice!</em></p>  
  <blockquote>ALICE'S RIGHT FOOT, <em>ESQ.</em></blockquote>  
  <p>Oh dear, what <em>nonsense</em> I'm talking!',...</p>  
</article>  
</body>  
</html>
```

Wonderland

Oh dear, what *nonsense* I'm talking!...

C 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.

Chapter 2.

cried
 planning
 Alice!
 ESQ.
 nonsense

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.

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

To style descendents universally:

1. Set up a list of descendent selectors including a universal selector. Type the HTML selector of the parent tag, followed by a space, and then an asterisk (*) or other selectors (**Code 4.2**).

```
article.copy * {...}
```

2. Styles will be used 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 to all of a parent's children.

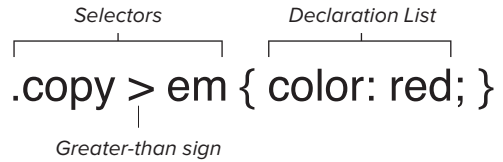
```
<article class="copy">
→ <h1>Alice's Adventures in
→ <em>Wonderland</em></h1>
→ <h2><em>Chapter 2.</em> The Pool
→ of Tears</h2>
→ <p>...<em>...</em>...</p>
→ </article>
```

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**.

```
<!-- HTML5 -->
<!DOCTYPE html>
<html lang="en">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<title>Alice's Adventures in Wonderland</title>
<style type="text/css" media="all">
  article.copy * em {
    color: red;
    font-weight: bold; }
</style>
</head>
<body>
<article class="copy">
  <h1>Alice's Adventures in <em>Wonderland</em></h1>
  <h2><em>Chapter 2.</em> The Pool of Tears</h2>
  <p>'Curiouser and curiouser!' <em>cried</em> Alice,...</p>
  <p>And she went on <em>planning</em>,...</p>
  <p>Poor <em>Alice!</em></p>
  <blockquote>ALICE'S RIGHT FOOT, <em>ESQ.</em></blockquote>
  <p>Oh dear, what <em>nonsense</em> I'm talking!',...</p>
</article>
</body>
</html>
```

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 close 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.copy > 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 will be used if the pattern is matched.

```
<article class="copy"><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.

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**.

```
<!-- HTML5 -->
<!DOCTYPE html>
<html lang="en">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<title>Alice's Adventures in Wonderland</title>
<style type="text/css" media="all">
  article.copy > p > em {
    color: red;
    font-weight: bold; }
</style>
</head>
<body>
<article class="copy">
  <h1>Alice's Adventures in <em>Wonderland</em></h1>
  <h2><em>Chapter 2.</em> The Pool of Tears</h2>
  <p>'Curiouser and curiouser!' <em>cried</em> Alice,...</p>
  <p>And she went on <em>planning</em>,...</p>
  <p>Poor <em>Alice!</em></p>
  <blockquote>ALICE'S RIGHT FOOT, <em>ESQ.</em></blockquote>
  <p>Oh dear, what <em>nonsense</em> I'm talking!',...</p>
</article>
</body>
</html>
```

Alice's Adventures in Wonderland

Chapter 2. The Pool of Tears

cried

planning

Alice!

nonsense

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:

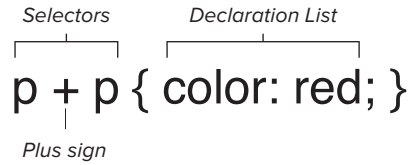
1. **Set up a list of adjacent sibling selectors.** Type the selector for the first element (HTML, class, or ID), a plus sign (+), and then the selector (HTML, class, or ID) for the adjacent element to which you want the style applied (**Code 4.4**).

p + p { ... }

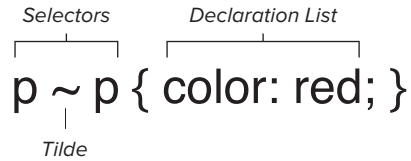
2. **Styles will be used if the pattern is matched.**

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

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.



G The general syntax for the adjacent sibling selector.



H The general syntax for the general sibling selector.

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**.

```
<!-- HTML5 -->
<!DOCTYPE html>
<html lang="en">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<title>Alice's Adventures in Wonderland</title>
<style type="text/css" media="all">
  p + p em {
    color: red;
    font-weight: bold; }
</style>
</head>
<body>
<article class="copy">
  <h1>Alice's Adventures in <em>Wonderland</em></h1>
  <h2><em>Chapter 2.</em> The Pool of Tears</h2>
  <p>'Curiouser and curiouser!' <em>cried</em> Alice,...</p>
  <p>And she went on <em>planning</em>,...</p>
  <p>Poor <em>Alice!</em></p>
  <blockquote>ALICE'S RIGHT FOOT, <em>ESQ.</em></blockquote>
  <p>Oh dear, what <em>nonsense</em> I'm talking!',...</p>
</article>
</body>
</html>
```

Alice's Adventures in Wonderland

Chapter 2. The Pool of Tears

'Curiouser and curiouser!' *cried* Alice...

planning

Alice!

ALICE'S RIGHT FOOT, *ESQ.*

Oh dear, what *nonsense* I'm talking!...

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).

`p ~ p {...}`

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 will be used if the pattern is matched.

```
<p>...</p><p>...</p><p>...</p>  
→ <blockquote>...</blockquote>  
→ <p>...</p>
```

TIP The styles will be applied to *any* siblings that occur after the first sibling selector, not just the first one, but any siblings of the same type until another type of element is encountered. Child siblings are not supported in IE6, so you will need to style these separately. See Chapter 13 for more information on adding styles specifically for Internet Explorer.

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...
<code>a * b</code>	b within a regardless of b 's parents
<code>a > * > b</code>	b is the direct child of any element that is the direct child of a
<code>a + * + b</code>	sibling b immediately after any element that is immediately after a
<code>*:hover</code>	mouse pointer over any element
<code>*:disabled</code>	any element that is disabled
<code>*:first-child</code>	first child of any element
<code>*:lang()</code>	any element using specified language code
<code>*:not(s)</code>	any element that is not the using indicated selectors
<code>*::first-letter</code>	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 **1**.

```
<!-- HTML5 -->
<!DOCTYPE html>
<html lang="en">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<title>Alice's Adventures in Wonderland</title>
<style type="text/css" media="all">
  p ~ p em {
    color: red;
    font-weight: bold; }
</style>
</head>
<body>
<article class="copy">
  <h1>Alice's Adventures in <em>Wonderland</em></h1>
  <h2><em>Chapter 2.</em> The Pool of Tears</h2>
  <p>'Curiouser and curiouser!' <em>cried</em> Alice,...</p>
  <p>And she went on <em>planning</em>,...</p>
  <p>Poor <em>Alice!</em></p>
  <blockquote>ALICE'S RIGHT FOOT, <em>ESQ.</em></blockquote>
  <p>Oh dear, what <em>nonsense</em> I'm talking!',...</p>
</article>
</body>
</html>
```

Alice's Adventures in Wonderland

Chapter 2. The Pool of Tears

'Curiouser and curiouser!' *cried* Alice...

planning

Alice!

nonsense

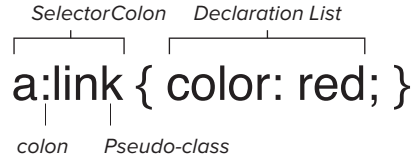
1 The results of Code 4.5. The text in red matches the general sibling criteria—in this case the emphasis tags within the second, third, and fourth paragraphs.

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 their 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 their 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	Name	Elements Are Styled If...	Compatibility
:link	Link	the value of href is not in history	IE4, FF1, O3.5, S1, CSS1
:visited	Visited Link	the value of href is in history	IE4, FF1, O3.5, S1, CSS1
:target	Targeted Link	a targeted anchor link	FF1.3, S1.3, C1, O9.5 CSS3
:active	Active	the element is clicked	IE7, FF1, O3.5, S1, CSS1
:hover	Hover	the pointer is over the element	IE4*, FF1, O3.5, S1, CSS2
:focus	Focus	the element has screen focus	IE7, FF1, O7, S1, CSS2

* Only available for anchor tags until IE7

TABLE 4.4 Structural/Other Pseudo-Classes

Format	Name	Elements Are Styled If...	Compatibility
:root	Root	is the top level element in a document	FF1.5, O9.5, S3.1, C3, CSS3
:empty	Empty	has no children	FF1.5, O9.5, S3.1, C3, CSS3
:only-child	Only Child	has no siblings	FF1.5, O9.5, S3.1, C3, CSS3
:only-of-type	Only of Type	has its unique selector among its siblings	FF1.5, O9.5, S3.1, C3, CSS3
:first-child	First-Child	is the first child of another element	FF1.5, O9.5, S3.1, C3, CSS2
:nth-of-type(n)	Nth of Type	is the <i>n</i> th element with that selector	FF1.5, O9.5, S3.1, C3, CSS3
: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	FF1.5, O9.5, S3.1, C3, CSS3
:last-child	Last Child	is the last child in the parent element	FF1.5, O9.5, S3.1, C3, CSS3
:first-of-type	First of Type	is the first of its selector type in the parent element	FF1.5, O9.5, S3.1, C3, CSS3
:last-of-type	Last of Type	is the last of its selector type in the parent element	FF1.5, O9.5, S3.1, C3, CSS3
:lang()	Language	has a specified language code defined	IE8, FF1.5, O9.5, S3.1, C3, CSS2.1
:not(s)	Negation	is not using specific selectors	FF1.5, O9.5, S3.1, C3, CSS3

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 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 84

Chapter 1.

Down The Rabbit-Hole

Chapter 2. The Pool of Tears Link

Chapter 3. A Caucus-Race and a Long Tale Visited

Chapter 4. The Rabbit Sends in a Little Bill Hover

Active

Chapter 6. Pig and Pepper

Chapter 7. A Mad Tea-Party

Chapter 8. The Queen’s Croquet-Ground

Chapter 9. The Mock Turtle’s Story

Chapter 10. The Lobster Quadrille

Chapter 11. Who Stole The Tarts?

Chapter 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.

```
<!-- HTML5 -->
<!DOCTYPE html>
<html lang="en">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<title>Alice's Adventures in Wonderland</title>
<style type="text/css" media="all">
  a {
    display: block;
    text-decoration: none;
    padding: 5px;
    width: 200px; }
  a:link {
    color: rgb(255,102,102);
    border-bottom: 1px dotted rgb(255,51,51); }
  a:visited {
    color: rgb(255,153,153);
    border-bottom: 1px dotted rgb(255,235,235); }
  a:hover {
    color: rgb(255,0,0);
    border-bottom: 1px solid rgb(255,0,0); }
  a:active {
    color: rgb(0,0,255);
    border-bottom: 1px dotted rgb(102,102,102); }
</style>
</head>
<body>
<navigation>
  <a href=""><strong>Chapter 1. </strong>Down The Rabbit-Hole</a>
  <a href=""><strong>Chapter 2. </strong>The Pool of Tears</a>
  <a href=""><strong>Chapter 3. </strong>A Caucus-Race and a Long Tale</a>
  <a href=""><strong>Chapter 4. </strong>The Rabbit Sends in a Little Bill</a>
  <a href=""><strong>Chapter 5. </strong>Advice from a Caterpillar</a>
  <a href=""><strong>Chapter 6. </strong>Pig and Pepper</a>
  <a href=""><strong>Chapter 7. </strong>A Mad Tea-Party</a>
  <a href=""><strong>Chapter 8. </strong>The Queen's Croquet-Ground</a>
  <a href=""><strong>Chapter 9. </strong>The Mock Turtle's Story</a>
  <a href=""><strong>Chapter 10. </strong>The Lobster Quadrille</a>
  <a href=""><strong>Chapter 11. </strong>Who Stole The Tarts?</a>
  <a href=""><strong>Chapter 12. </strong>Alice's Evidence</a>
</navigation>
</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.

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 unvisited 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 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.

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 {...}
```

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, nondynamic style for the elements receiving dynamic styles (Code 4.7).

2. Style the *hover* state of the element.

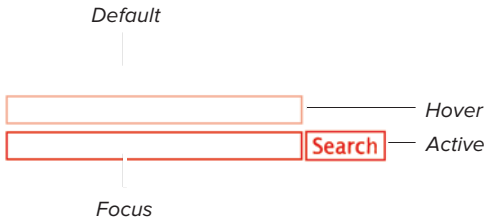
Type the selector (HTML, class, or ID), a colon (:), and then **hover**.

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 are set to change style when the user interacts with them by hovering, selecting (focus), or clicking (active) .

```
<!-- HTML5 -->
<!DOCTYPE html>
<html lang="en">
<head>
<meta http-equiv="Content-Type"
→ content="text/html; charset=UTF-8" />
<title>Alice's Adventures in Wonderland
→ </title>
<style type="text/css" media="all">
  input {
    border: 3px solid rgb(153,153,153);
    background-color: rgb(204,204,204);
    color: rgb(153,153,153);
    padding: 0 5px;
    font-size: 2em; }
  input:hover {
    border-color: rgb(204,153,153);
    color: rgb(102,102,102); }
  input:focus {
    border-color: rgb(255,0,0);
    background-color: rgb(255,255,255);
    color: rgb(0,0,0);
    outline: none; }
  input:active {
    color: rgb(255,0,0);
    border-color: rgb(255,0,0);
    background-color: rgb(0,0,0); }
</style>
</head>
<body>
<form>
  <input type="text" value="First Name">
  <input type="text" class="hover" value="Last
Name">
  <input type="text" class="focus"
  → value="eMail">
  <input type="button" class="active"
  → value="Search">
</form>
</body>
</html>
```



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 type-face, 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="Search">
```

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 nonkeyboard (mouse) interactions with the element. For keyboard-only Web users, `:focus` will apply.

NEW IN CSS3: 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 for it based on location within the list **D**.

```
<!-- HTML5 -->
<!DOCTYPE html>
<html lang="en">
<head>
<meta http-equiv="Content-Type"
→ content="text/html; charset=UTF-8" />
<title>Alice's Adventures in Wonderland
→ </title>
<style type="text/css" media="all">
  li:first-child { font-size: .875em; }
  li:first-of-type { color: red; }
  li:nth-of-type(3) { font-size: 1.5em }
  li:nth-last-of-type(2) { font-size: 2em; }
  li:last-of-type { color: red; }
  li:last-child { font-size: 2.5em; }
</style>
</head>
<body>
<ol>
  <li>Alice</li>
  <li>The White Rabbit</li>
  <li>The Mad Hatter</li>
  <li>The Queen of Hearts</li>
  <li>The Door Mouse</li>
</ol>
</body>
</html>
```

1. Alice

5. The Door Mouse

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.

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.

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.

Code 4.9 Styles are set to turn paragraphs red if they are in French (fr) **E**.

```
<!-- HTML5 -->
<!DOCTYPE html>
<html lang="en">
<head>
<meta http-equiv="Content-Type"
→ content="text/html; charset=UTF-8" />
<title>Alice's Adventure's in Wonderland
→ </title>
<style type="text/css" media="all">
  p:lang(fr) {
    color: red;
    font-style: italic; }
</style>
</head>
<body>
  <p>It sounded an excellent plan,...</p>
  <p lang="fr">On aurait dit un excellent
→ plan,...</p>
</body>
</html>
```

TIP 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.

On aurait dit un excellent plan, sans doute, et fort proprement et simplement disposés, la seule difficulté, c'est qu'elle n'avait pas la moindre idée de comment s'y prendre, et tandis qu'elle regardait avec anxiété à propos parmi les arbres, une petite barque forte juste sur sa tête la faisait paraître en toute hâte.

E The results of Code 4.9 show the paragraph in French rendered in red (with my apologies to French speakers).

Code 4.10 If the element is a paragraph that does *not* use the dialog class, it will be displayed in red and italics **F**.

```
<!-- HTML5 -->
<!DOCTYPE html>
<html lang="en">
<head>
<meta http-equiv="Content-Type"
→ content="text/html; charset=UTF-8" />
<title>Alice's Adventures in Wonderland</
→ </title>
<style type="text/css" media="all">
  p:not(.dialog) {
    color: red;
    font-style: italic; }
</style>
</head>
<body>
<p class='dialog'>"Why?" said the
→ Caterpillar.</p>

<p>Here was another puzzling question,...</p>

<p class='dialog'>"Come back!" the
→ Caterpillar,..."</p>
</body>
</html>
```

NEW IN CSS3: *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**).
2. **The element is not styled if it contains the indicated selector.**

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

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

```
→ <p>...</p>
```

The styles are applied to elements that match the initial selector but *not* the selector in parentheses.

Here was another puzzling question; and as Alice could not think of any good reason, and as the Caterpillar seemed to be in a VERY unpleasant state of mind, she turned away.

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

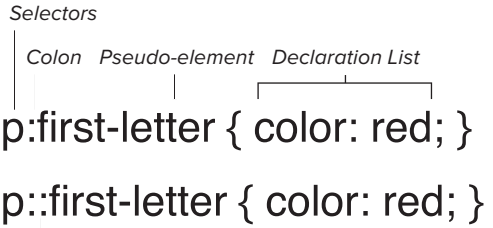
You can access the first letter of any block of text directly using the **:first-letter** pseudo-element. The first line of any block of text can be isolated for style treatment using the **:first-line** pseudo-element.

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.

TABLE 4.5 Pseudo-Elements

Format	Name	Elements Are Styled If...	Compatibility
:first-letter, ::first-letter	the first Letter	first letter in text	IE5.5, FF1, O3.5, S1, CSS1
:first-line, ::first-line	the first line of text	they are the first line of text	IE5.5, FF1, O3.5, S1, CSS1
:after, ::after	After	space immediately before element	IE8, FF1, O5, S1, CSS2
:before, ::before	Before	space immediately after element	IE8, FF1, O5, S1, CSS2

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

```
<!-- HTML5 -->
<!DOCTYPE html>
<html lang="en">
<head>
<meta http-equiv="Content-Type"
→ content="text/html; charset=UTF-8" />
<title>...</title>
<style type="text/css" media="all">
  article p {
    font-size: 16px;
    line-height: 24px;
    color: rgb(102,102,102) }
  article p:first-of-type:first-letter {
    color: red;
    font-size: 3em;
    float: left;
    margin-right: 5px; }
  article p:first-of-type:first-line {
    font-size: 1.25em;
    font-weight: bold;
    color: rgb(0,0,0); }
</style>
</head>
<body>
<article>
<h1>Alice's Adventures in Wonderland</h1>
  <p>The moment Alice appeared,...</p>
  <p>The executioner's argument was,...</p>
  <p>The King's argument was,...</p>
</article>
</body>
</html>
```



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.

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 styling 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.

```
h1 {...}
```

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**.

```
h1:before { content:... }
```

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

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**.

```
h1:after { content:... }
```

Next, declare the **content** property and define what generated content goes after 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**.

```
<!-- HTML5 -->
<!DOCTYPE html>
<html lang="en">
<head>
<meta http-equiv="Content-Type"
→ content="text/html; charset=UTF-8" />
<title>...</title>
<style type="text/css" media="all">
  h1 {
    font-size: 2em;
    color: red;
    font-style: italic; }
  h1:before {
    content: url('../_images/bullet-01.png'); }
  h1:after {
    content: url('../_images/bullet-02.png'); }
</style>
</head>
<body>
<h1>Alice's Adventures in Wonderland</h1>
  <p>The moment Alice appeared,...</p>
  <p>The executioner's argument was,...</p>
  <p>The King's argument was,...</p>
</article>had a head could be beheaded,
→ and that you weren't to talk nonsense.</p>
</article>
</body>
</html>
```



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

Alice's Adventures in Wonderland

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.

Coming Soon! Styling the Selection

Although not implemented in enough (or any) browsers to make it worth even thinking about yet, a great new pseudo-element is coming to CSS3, **::selection**, which will style any element selected by the user.

TIP The pseudo-elements 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 now until all browsers have adopted the syntax.

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

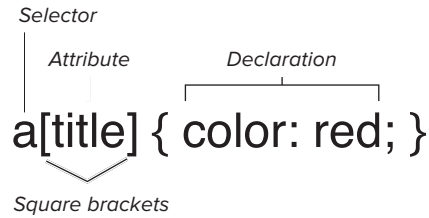
Although style attributes should all be handled by CSS, many HTML tags still have attributes that define how they behave. For example, the image tag, `img`, always includes the `src` attribute to define the source for the image file to be loaded.

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.

TABLE 4.6 Attribute Selectors

Format	Name	Elements Are Styled If That Element:	Compatibility
<code>[attr]</code>	Attribute	has specified attribute	IE7, FF1.5, O5, S2, CSS2
<code>[attr="value"]</code>	Exact value	has specified attribute equal to exact value	IE7, FF1.5, O5, S2, CSS2
<code>[attr~="value"]</code>	Spaced List	has specified attribute equal to exact value within space-separated list	IE7, FF1.5, O5, S2, CSS2
<code>[attr = "value"]</code>	Hyphenated List	has specified attribute equal to exact value within hyphen-separated list	IE7, FF1.5, O5, S2, CSS2
<code>[attr^="value"]</code>	Begins with	has specified attribute equal to exact value at beginning	CSS3
<code>[attr\$="value"]</code>	Ends With	has specified attribute equal to exact value at end	CSS3
<code>[attr*="value"]</code>	Contains	has specified attribute equal to exact value anywhere	CSS3

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

```
<!-- HTML5 -->
<!DOCTYPE html>
<html lang="en">
<head>
<meta http-equiv="Content-Type"
→ content="text/html; charset=UTF-8" />
<title>Alice's Adventures in Wonderland</
→ title>
<style type="text/css" media="all">
  a[title] { display: block; color: rgb(0,0,0);
→ font-size: .8em; }
  a[title="Home"] {color: rgb(51,0,0);
→ font-size: 1em;}
  a[title="email"] { color:rgb(102,0,0);
→ font-size: 1.2em; }
  a[title="resume"] { color: rgb(153,0,0);
→ font-size: 1.4em;}
  a[href^="http://"] {color: rgb(204,0,0);
→ font-size: 1.6em;}
  a[href$=".info"] {color: rgb(235,0,0);
→ font-size: 1.8em;}
  a[href*="speakinginstyles"]
→ {color: rgb(255,0,0); font-size: 2em;}
</style>
</head>
<body>
<navigation>
  <h2>About the Author</h2>
  <a href="" title="Portfolio">Portfolio</a>
  <a href="index.html" title="Home">Home
→ Page</a>
  <a href="" title="contact email
→ link">Email</a>
  <a href="" title="resume-link">RV@sumv@</a>
  <a href="http://www.jasonspeaking.com"
→ title="blog">JasonSpeaking</a>
  <a href="http://www.fluidwebtype.info"
→ title="book">Fluid Web Typography</a>
  <a href="http://www.speakinginstyles.com"
→ title="book">Speaking In Styles</a>
</navigation>
</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.

continues on next page

Portfolio
Home Page
Email
Résumé
JasonSpeaking
Fluid Web Typography
Speaking In Styles

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

- 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.

- 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. **NEW IN CSS3 ★: 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. **NEW IN CSS3 ★: 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.

TIP Values are case sensitive. In other words, 'Alice' and 'alice' are two different values.

NEW IN CSS3: 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 delivers 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 102

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: 600px)
  and (max-width: 980px)"
```

A The general syntax for media queries.

TABLE 4.8 Media Query Properties

Property	Value	Compatibility
aspect-ratio	<ratio>	FF3.5, S1, C1, O9.5, CSS3
max-aspect-ratio	<ratio>	FF3.5, S1, C1, O9.5, CSS3
min-aspect-ratio	<ratio>	FF3.5, S1, C1, O9.5, CSS3
device-aspect-ratio	<ratio>	FF3.5, S1, C1, O9.5, CSS3
max-device-aspect-ratio	<ratio>	FF3.5, S1, C1, O9.5, CSS3
min-device-aspect-ratio	<ratio>	FF3.5, S1, C1, O9.5, CSS3
color	<integer>	FF3.5, S1, C1, O10, CSS3
max-color	<integer>	FF3.5, S1, C1, O10, CSS3
min-color	<integer>	FF3.5, S1, C1, O10, CSS3
color-index	<integer>	FF3.5, S1, C1, O10, CSS3
max-color-index	<integer>	FF3.5, S1, C1, O10, CSS3
min-color-index	<integer>	FF3.5, S1, C1, O10, CSS3
device-height	<length>	FF3.5, S1, C1, O9.5, CSS3
max-device-height	<length>	FF3.5, S1, C1, O9.5, CSS3
min-device-height	<length>	FF3.5, S1, C1, O9.5, CSS3
device-width	<length>	FF3.5, S1, C1, O9.5, CSS3
max-device-width	<length>	FF3.5, S1, C1, O9.5, CSS3
min-device-width	<length>	FF3.5, S1, C1, O9.5, CSS3
height	<length>	FF3.5, S1, C1, O9.5, CSS3
max-height	<length>	FF3.5, S1, C1, O9.5, CSS3
min-height	<length>	FF3.5, S1, C1, O9.5, CSS3
monochrome	<integer>	FF3.5, S1, C1, O10, CSS3
max-monochrome	<integer>	FF3.5, S1, C1, O10, CSS3
min-monochrome	<integer>	FF3.5, S1, C1, O10, CSS3
orientation	portrait, landscape	FF3.5, S1, C1, CSS3
resolution	<resolution>	FF3.5, S1, C1, O10, CSS3
max-resolution	<resolution>	FF3.5, S1, C1, O10, CSS3
min-resolution	<resolution>	FF3.5, S1, C1, O10, CSS3
scan	progressive, interlaced	FF3.5, S1, C1, O10, CSS3
width	<length>	FF3.5, S1, C1, O9.5, CSS3
max-width	<length>	FF3.5, S1, C1, O9.5, CSS3
min-width	<length>	FF3.5, S1, C1, O9.5, CSS3

- **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 media type and include sans-serif fonts, a dark background, and light text.

```
/**/ Default Styles ***/

body {
  background: black url('../_images/AAIW-illos/
  → alic23b.gif') no-repeat 0 0;
  margin: 0 0;
  padding: 200px 0 0 175px; }
h1 {
  color: white;
  font-style: italic; }
h2 {
  color: rgb(153,153,153); }
p {
  font: normal 100%/1.5 Corbel, Helvetica,
  → Arial, Sans-serif;
  color: rgb(204,204,204); }
```

Code 4.15 *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 to match.

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

body {
  background: white url('../_images/AAIW-illos/
  → alic23a.gif') no-repeat 0 0;
  padding: 200px 0 0 175px;
  }
h1 {
  color: black; }
p {
  font: normal 12pt/2 Constantia, palatino,
  → times, "times new roman", serif;
  color: rgb(0,0,0); }
```

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 4.15). You can call the sheet **print.css**, but you might also want to create style sheets specifically for popular mobile devices such as the iPhone (Code 4.16), which you could name **iphone.css**.

continues on next page

Code 4.16 *iphone.css*—These styles are specific for use on an iPhone and are loosely based on that mobile device's look and feel.

```
/**/ iPhone Styles ***/

body {
  -webkit-text-size-adjust:none;
  background: rgb(102,102,102) url('../_images/
  → AAIW-illos/alice23c.gif') no-repeat
  → center 0;
  padding: 120px 20px 20px 20px; }
h1 { color: rgb(153,125,125);
  text-shadow: 0 0 5px rgb(0,0,0); }
p {
  font: normal 1em/1.25em "helvetica neue",
  → Helvetica, Arial, Sans-serif;
  color: rgb(255,255,255); }
```

2. Add the viewport meta tag. In the head of your HTML document (**Code 4.17**), add a meta tag with a name equal to viewport and content, as shown.

```
<meta name="viewport"
→ content="width=device-width;
→ initial-scale=1.0;
→ maximum-scale=1.0;
→ user-scalable=0;">
```

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



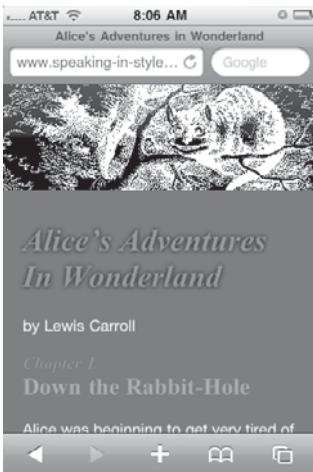
B Code 4.17 output to a computer screen. This version uses a dark background and an inverted version of the *Alice's Adventures in Wonderland* illustration. On an LCD screen, the lightly colored text will look fine.

Code 4.17 The HTML code links to all three of the style sheets, which are displayed in default **B**, Print **C**, and in the iPhone **D**. 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.

```
<!-- HTML5 -->
<!DOCTYPE html>
<html lang="en">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<meta name="viewport" content="width=device-width; initial-scale=1.0; maximum-scale=1.0;
→ user-scalable=0;">
<title>Alice's Adventure's In Wonderland</title>
<link rel="stylesheet" media="all" href="default.css" >
<link rel="stylesheet" media="print" href="print.css">
<link rel="stylesheet" media="screen and (max-device-width: 480px) and (min-device-width: 320px)"
→ href="iphone.css" >
</head>
<body>
<h1>Alice's Adventures In Wonderland</h1>
<p class="byline">by <span class="author">Lewis Carroll</span></p>
<article><!-- Article -->
<header>
<h2><strong>Chapter I.</strong> Down the Rabbit-Hole</h2>
</header>
<p>
Alice was beginning to get very tired of sitting by her sister,...</p>
</article>
</body>
</html>
```



C Code 4.17 output to a printer. The background is white, and the background image is no longer inverted. This works better in print.



D Code 4.17 on an iPhone. A specially tailored version to fit the width of an iPhone uses a custom header of the Cheshire cat.

3. Link to your *default* style sheet. In the head of your HTML document, 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" >
```

4. 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">
```

5. Use a media query to link to a style sheet. Immediately after the previous `<link>` tag, add another `<link>` tag that references the style sheet for a specific media type and then add media queries (Table 4.8) in parentheses connecting multiple queries with **and**.

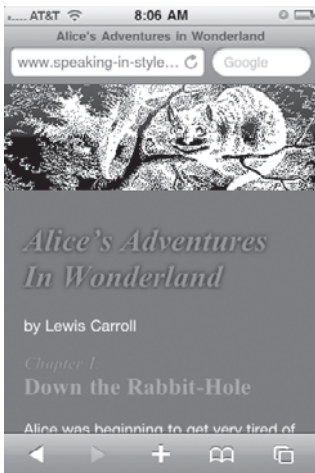
```
<link rel="stylesheet"
→ media="screen and
→ (max-device-width: 480px)
→ and (min-device-width: 320px)"
→ href="iphone.css" >
```

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.

Code 4.19 The HTML code links to the various style sheets for different media types. The big difference between this version and Code 4.16 is that the iPhone-specific code is now embedded in `screen.css`, so I'm not including media queries **F**.

```
<!-- HTML5 -->
<!DOCTYPE html>
<html lang="en">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<meta name="viewport" content="width=device-width; initial-scale=1.0; maximum-scale=1.0;
→ user-scalable=0;">
<title>Alice's Adventures In Wonderland</title>
<link rel="stylesheet" media="all" href="default.css" >
<link rel="stylesheet" media="print" href="print.css">
<link rel="stylesheet" media="screen" href="screen.css">
</head>
<body>
<h1>Alice's Adventures In Wonderland</h1>
<p class="byline">by <span class="author">Lewis Carroll</span></p>
<article><!-- Article -->
<header>
<h2><strong>Chapter I.</strong> Down the Rabbit-Hole</h2>
</header>
<p>
Alice was beginning to get very tired of sitting by her sister,...</p>
</article>
</body>
</html>
```



F Code 4.18 on an iPhone. This looks the same as **D**, but the code is now in an `@media` rule.

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.

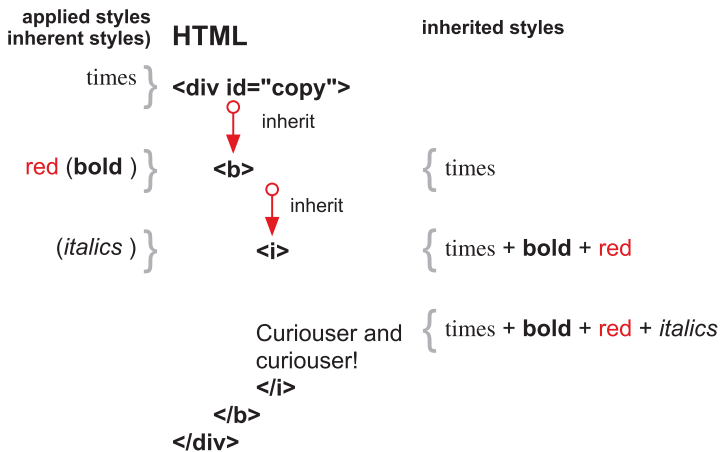
Inheriting Properties from a Parent

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**.

CSS

```
b { color : red; }  
.copy { font-family: times; }
```



RESULT

Curiouser and curiouser!

A The final result of the styles applied and inherited is bold, red, and italicized text in Times font.

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 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 of 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 no style to be applied.
- **auto**—Allows the browser to determine how the element should be displayed based on context.

Selector Declaration *!important*

`h1 { color: red !important; }`

A The general syntax for `!important`.

Code 4.20 The `!important` value has been added to the color property in the first `h1`, but not in the second **B**. Typically, the second `h1` would override the first, but not in this case.

```
<!-- HTML5 -->
<!DOCTYPE html>
<html lang="en">
<head>
<meta http-equiv="Content-Type"
  → content="text/html; charset=UTF-8" />
<title>Alice's Adventures in Wonderland
  → </title>
<style type="text/css" media="all">
  h1 {
    color: red !important;
    font-size: 3em; }...
  h1 {
    color: black;
    font-size: 2em; }
</style>
</head>
<body>
<article>
  <h1>Alice's Adventures in
  → <em>Wonderland</em></h1>
</article>
</body>
</html>
```



B The result of Code 4.20. The style that is most important wins the day, so the text is red rather than black.

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).

`h1 {...}`

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: red !important;`

3. Add other styles.

`font-size: 1em;`

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

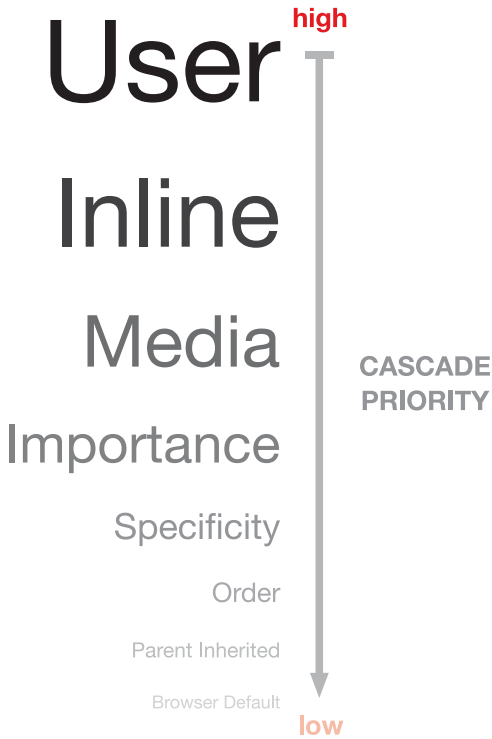
`!important` is a powerful tool, second only to inline styles for determining style cascade. `!important` is great for debugging your CSS; but, because it can interfere with making changes later, it should never be used in the final Web site code.

TIP 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 4.1 specification in which a user-defined style sheet overrides an author-defined style sheet.



A The cascade order from most important to least important.

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 sheet. In principle, these always have precedence over other styles.

continues on next page

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.

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 only worth 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. Parent inherited styles

These styles are inherited from the parent.

8. Browser default styles

These styles are applied by the browser and are the least important.

Putting It All Together

1. **Using the HTML code you created in Chapter 3, style all descendants of the paragraph tag to bold.** This requires using the universal selector.
2. **Style all paragraph siblings.** Turn all paragraphs that follow another paragraph to gray.
3. **Add hypertext links to your page, and then use the link pseudo-classes to style them.** Remember to style all four link states.
4. **Style the first letter and first line of text of the first paragraph in your page.** Start by styling the first line of all paragraphs and then use pseudo-classes to focus on the first paragraph.
5. **Style images with an alt tag value with a black rule and images that do not have a value with a red rule.** This is useful for highlighting images you need to add alt values to.
6. **Style your page for print by hiding navigation and using a white background and dark text.** You want to consider media other than screen for your designs.
7. **Play around with adding more styles and then changing the order to see how the order affects which styles are applied.** Pay careful attention to the cascade order and how specificity affects that.
8. **Play around with adding !important to different styles to see how it affects which styles are applied.** Remember, the !important property is very powerful and should only be used to help you style your page while working. The final design should be able to work without it.

Index

Symbols

- : (colon)
 - CSS tips, 37
 - pseudo-element syntax, 95
- . (period), defining classes, 51, 54
- ;(semicolon)
 - character entities and, 123
 - troubleshooting CSS code, 373
- '...' (single quotes)
 - CSS tips, 37
 - specifying style, 238–239
- / (slash), adding comments to CSS, 66–67
- , (comma), for grouping selectors, 62
- "..." (double quotes)
 - adding generated content, 235
 - CSS tips, 37
 - specifying style, 238–240
- :: (double colon), pseudo-element syntax, 95
- = (equals sign), CSS tips, 37
- # (pound), defining ID selectors, 55
- & (ampersand), as character entity, 123
- * (asterisk)
 - adding comments to CSS, 66–67
 - defining universal selectors, 59
 - styling descendants, 71
- [] (square brackets), in attribute selector syntax, 96
- { } (curly brackets)
 - in declarations, 48
 - troubleshooting CSS code, 373
- ~ (tilde sign), defining general sibling selectors, 78
- > (close angle bracket), styling children, 74

Numbers

- 2D transformations, 308–311
- 3D
 - stacking objects in, 292–293
 - transformations, 311–315

A

- absolute font sizes, 135
- absolute positioning
 - defined, 286
 - setting, 288
 - tip, 289
- accessibility
 - access keys for, 350
 - color and, 195
 - counter-reset** and, 237
- accesskey** attribute, 350
- :active**
 - defined, 82
 - setting contrasting appearances with, 84–85
 - styling for interaction, 86–87
- adjacent sibling elements
 - defined, 70
 - styling based on context, 76–77
- :after**
 - counters, 237
 - defined, 94
 - defining generated content, 234
- Aggregate method, as style sheet strategy, 370–371
- alignment
 - floating elements and, 258–259
 - horizontal text, 162–163
 - vertical text, 164–166, 339

alpha values, 183–184
alphabetical organization, 365
ampersand (&), as character entity, 123
analogic color, 195
anchor tags, styling with pseudo-classes, 82

angle

 Mozilla gradient value, 188
 transform values, 307
animation, 276
appearance, of mouse pointer, 232–233

articles (**<article>**)

 defined, 26
 using HTML5 structure, 29

asides (**<aside>**)

 defined, 26
 using HTML5 structure, 29

aspect-ratio, 100

associations, color, 191

asterisk (*)

 adding comments to CSS, 66–67
 defining universal selectors, 59
 styling descendants, 71

attachment, setting background image, 200, 204

attributes

 adding class, 54
 adding generated content, 235
 adding **id** to HTML tag, 56
 changing **:hover**, 87
 defining styles based on tag, 96–99
 list-style, 219
 vs. selectors, 12

Audio and Video Timed media playback, 25

B

backface-visibility, 3D transformations, 313

background, 208–211

background-attachment, 204

background-clip, 206

background-color, 198–199

background-image, 201, 390

background-origin, 207

background-position, 204–205

background-repeat, 203

backgrounds

 box properties, 246
 color gradients in, 187–190
 color palette for, 192–193
 gradients in, 391
 link styles and, 352
 new in CSS3, 13
 putting it all together, 212
 setting border image, 271–273
 setting color, 198–199
 setting image, 200–207
 shorthand, 208–211
 styling for print, 108

background-size, 205

:before

 counters, 237
 defined, 94
 generated content and, 234

best practices, CSS, 385–392

blink, 174

block, 248

block-level elements

 creating multicolumn layout, 346
 HTML selectors for, 20
 redefining HTML tags, 50

Blueprint, CSS Frameworks, 367

blur

 drop shadow, 160
 setting element shadow, 300

body (**<body>**)

 adding unique class name or ID to body tag
 of every page, 387
 defined, 17
 HTML5 structure for, 28
 not placing style links in, 386
 redefining HTML tags, 50
 setting background color, 192
 setting margins, 262

bolding

 fonts, 142–143
 link styles and, 352

border-bottom, 174

border-collapse, 223

border-radius, 268–270

borders

- adding color with, 184
- box properties, 246
- clipping element, 297
- collapsing between cells, 223–224
- color palette for, 193
- CSS resets for, 339
- new in CSS3, 13
- rounding corners, 268–270
- setting **background-clip**, 206
- setting **background-origin**, 207
- setting element, 265–267
- setting image, 271–273
- setting space between table cells, 222

border-spacing, 222

bottom, setting element position, 290–291

box properties

- coming soon, 276
- controlling overflowing, 254–256
- displaying element, 248–250
- floating elements in window, 257–259
- overview, 241–244
- putting it all together, 277
- rounding border corners, 268–270
- setting border image, 271–273
- setting element border, 265–267
- setting element height and width, 251–253
- setting element margins, 260–262
- setting element outline, 263–264
- setting element padding, 274–275
- understanding element's, 245–247

boxes

- CSS fixes for IE6 box model, 333–334
- new in CSS3, 13

box-shadow, 300–301, 316

box-sizing, 276

break tag clear fix, for floating elements, 340–341

browsers

- 2D transformations, 308–311
- 3D transformations, 314
- attribute selector compatibility, 96
- background color settings, 198–199
- background** shorthand support, 211

cascade order of, 115

color gradients, 187–190

color value compatibility, 181

combinatory selector compatibility, 71

CSS extensions, 11–12

CSS resets for creating consistent browser styles, 336

CSS support, 324

CSS3 and, 14

custom pointer support, 233

default margins, 262

designing for enhanced features of, 325

downloadable Webfonts and formats, 127–128

evolution of CSS and, 6–7

font size for screen vs. print, 135

font-family values, 127

HTML5 work in IE, 30–31

inherited styles, 18

inline styles, 37

link style settings, 85

media querying, 100–102

positioning and, 289

pseudo-class compatibility, 81

pseudo-element compatibility, 92

pseudo-element syntax support, 95

specifying styles with media queries, 103–105

teaching to count, 236–237

transitions, 318

troubleshooting code on different, 375

understanding window and document, 283–284

bullets

creating, 217

positioning, 218

setting style, 216

buttons, dynamic styles for, 388

C

Camino, CSS extension, 12

Canvas element, new in HTML5, 25

capitalization

small caps, 144–145

text properties, 158–159

- caption:**, mimicking visitor styles, 149
- captions, positioning in table, 226
- caption-side**, 226
- cascade
 - determining cascade order, 113–115
 - making declarations **!important**, 111–112
- case, setting text, 158–159
- cells
 - collapsing borders between, 223–224
 - dealing with empty, 225
 - setting space between, 222
- centering element, 262
- character entities, 123
- character sets, specifying, 119–120
- child elements
 - box properties, 245–246
 - floating, 258
 - inheriting properties from parent, 111–112
 - opacity, 299
 - positioning, 291
 - styling based on context, 74–75
 - styling specific with pseudo-classes, 88–90
 - text decoration properties, 174
- choke, 301
- Chrome
 - 3D transformations, 311
 - CSS extension, 12
 - testing code in, 391
 - tools for analyzing/editing code, 376
 - Web Inspector, 379–380
 - Webkit gradients, 189–190
- class selectors
 - basics, 34
 - defining, 51–54
 - overview, 9
 - styling with pseudo-classes, 85
 - troubleshooting CSS code, 373
- classes
 - defining reusable, 51–54
 - mixing and matching, 388
 - using generic names for, 387
- clear**, 258–259
- clear fix, for floating elements, 340–341
- clip**, setting background image, 200, 206
- clipping
 - element's visible area, 296–297
 - positioning type and, 285
- close angle bracket (>), styling children, 74
- code
 - creating minified version of CSS code, 382–384
 - debugging with Firebug, 376–378
 - debugging with Web Inspector, 379–380
 - troubleshooting CSS code, 372–375
 - validating CSS code, 381
- collapsing borders, between table cells, 223–224
- collapsing margins, 262
- colon (:)
 - CSS tips, 37
 - pseudo-element syntax, 95
- color**, 102, 196–197
- color alpha values, 183–184
- Color Palette Generator, 195
- color wheel, 194–195
- Color Wheel Selector, 195
- color-index**, 102
- colors
 - background, 198–199, 209
 - border, 266–267
 - drop shadows and, 161
 - element shadow and, 301
 - glossary of colors used, 362–363
 - gradients in backgrounds, 187–190
 - keywords, 185–187
 - link styles and, 85
 - new in CSS3, 13
 - overview, 179–180
 - palette of, 190–195
 - putting it all together, 212
 - RGB color values, 390
 - styling for print, 108
 - text, 196–197
 - values of, 181–184
- columns, multicolumn layout, 346–349
- combinatory selectors
 - defining styles based on context, 72–79
 - overview, 71

- commas (,) for grouping selectors, 62
- comments
 - adding to CSS, 66–67
 - best practices for, 392
 - troubleshooting CSS code, 373
- compact**, 251
- compatibility
 - attribute selectors, 96
 - child siblings in IE, 78
 - color values, 181
 - combinatory selectors, 71
 - custom pointers, 233
 - font family, 124
 - font-size-adjust** values, 136
 - media queries, 101
 - pseudo-classes, 81
 - pseudo-elements, 92
- complementary colors, 194
- conditional styles, setting up for IE, 328–332
- content
 - adding with CSS, 234–235
 - box properties, 246
 - choosing color palette based on, 193
 - controlling overflowing, 254–256
 - new in CSS3, 14
 - setting **background-clip**, 206
 - setting **background-origin**, 207
 - styling before and after, 94
 - styling for print, 108
 - styling links based on, 350
- contextual selectors
 - defining styles based on, 71–79
 - overview, 70
- contrast
 - color and, 195
 - links and, 82–85
- copying
 - from color palette, 193
 - converting quotes during, 37
- corners, rounding, 268–270
- counter-increment**, 236–237
- counter-reset**, 236–237
- counters, 235
- counting, browser, 236–237
- cropped content, for preventing overflow, 254–256
- CSS (Cascading Style Sheets), basics
 - best practices, 385–392
 - comments, 66–67
 - defined, 3–5
 - embedded styles, 38–40
 - evolution of, 6–7
 - external styles, 41–47
 - grouping, 62–65
 - HTML and, 18
 - HTML selectors for elements, 19–21
 - inline styles, 35–37
 - overview, 33
 - properties with transitions, 317
 - putting it all together, 68
 - Quirks mode and Standards mode, 324
 - redefining HTML tags, 48–50
 - reusable classes, 51–54
 - selectors, 34
 - unique IDs, 55–58
 - universal styles, 59–61
- CSS (Cascading Style Sheets) fixes
 - clear fix for float problem, 340–341
 - correcting box model in IE6, 333–334
 - floating elements and, 340
 - IE6 and, 324
 - overflow fix for float problem, 342
 - setting up conditional styles in IE, 328–332
 - underscore hack for IE6, 325–327
- CSS (Cascading Style Sheets) resets
 - Eric Meyer’s reset, 338
 - overview, 335–336
 - simple example, 336–337
 - starting with clean slate, 391
 - what you should reset, 339
 - Yahoo!’s YUI2 reset, 337
- CSS (Cascading Style Sheets), techniques
 - drop-down menus, 356–359
 - image rollovers added to Web pages, 354–356
 - multicolumn layout, 346–349
 - overview, 343–345
 - sprite technique, 354
 - styling navigation and links, 350–353

- CSS Frameworks, 367
- CSS libraries, 367
- CSS1 (Cascading Style Sheets Level 1), 7
- CSS2 (Cascading Style Sheets Level 2), 7
- CSS3 (Cascading Style Sheets Level 3)
 - CSS defined, 3–5
 - evolution of, 6–7
 - HTML and, 8
 - new in, 13–14
 - overview, 1
 - rule parts, 11–12
 - rules, 9–10
 - style defined, 2
 - W3C and, 10
- CSS-Positioning, 7
- curly brackets ({ })
 - in declarations, 48
 - troubleshooting CSS code, 373
- currentcolor**
 - defined, 182
 - setting background color, 198
- cursive fonts, 121
- cursor**, 232

D

- debugging CSS code
 - with Firebug, 376–378
 - with Web Inspector, 379–380
- decimal color values
 - color keywords, 184
 - defined, 181–182
 - vs. hex color values, 184
- declarations
 - adding for selector grouping, 63
 - adding to classes, 52
 - adding to ID, 56
 - adding to universal selectors, 59
 - defined, 11
 - defining HTML selectors, 48–50
 - Firebug for turning off, editing, or adding, 378
 - !important**, 111–112, 375
 - inline styles, 35–37
 - position, 287
 - tools for turning on/off, 376
 - Web Inspector for turning off, editing, or adding, 380
- decoration, text property, 172–174
- default background, 211
- default design, eliminating, 335
- default font values, 147
- default link state, 84
- default margins, 262
- default styles
 - determining cascade order, 115
 - media queries and, 103
- delay**, setting transition, 319
- deletion, **line-through** text property for, 174
- dependent classes, 51–52
- dependent IDs, 56
- descendant elements
 - defined, 70
 - styling based on context, 71–73
- design enhancement
 - CSS for, 14
 - eliminating design by default, 335
 - notes on, 315
- devices
 - media queries for, 102
 - specifying styles for, 103–105
- dialogs (**<dialog>**), 26
- dingbats, 122
- display
 - display** vs. **visibility**, 295
 - element, 248–250
 - overflow, 255
 - understanding window and document, 283–284
- <div>** tags, HTML, 54
- Divide and Conquer method, style sheet strategy, 369–370
- doctypes (**<!DOCTYPE>**)
 - defined, 17
 - HTML5 structure and, 27
 - reasons for using, 30
 - specifying, 385
- documents
 - editing in HTML5, 25
 - HTML structure (basic), 17

- HTML5 structure, 27–31
 - understanding, 283–284
- double colon (:), pseudo-element syntax, 95
- double quotation marks (“ ”), CSS tips, 37
- double quotes (“ ”), for specifying style, 238–239
- double-spacing text, 157
- drag-and-drop, new in HTML5, 25
- drop cap styled letters, 93
- drop shadow
 - adding to text, 160–161
 - setting element, 300–301
- drop-down menus, 356–359
- duration**, setting transition, 319
- Dynamic method, style sheet strategy, 371
- dynamic pseudo-classes
 - overview, 80–81
 - setting contrasting appearances with, 82–85
 - styling for interaction, 86–87
- dynamic styles, for form elements, button, and interface elements, 388

E

- edge, element, 284
- editing, new in HTML5, 25
- effects, transition, 316–320
- elements
 - alignment of, 166
 - background color settings, 198–199
 - box properties. See *box properties*
 - cascade order of, 113–115
 - default styles for HTML elements, 387
 - defined, 16
 - defining using same styles, 62–65
 - dynamic styles and, 388
 - edge, 284
 - family tree, 70
 - font definition for Web, 128–130
 - font family definition, 124–125
 - font size adjustments, 136–138
 - font size settings, 133–135
 - font-style definition, 139–141
 - inspecting with Firebug, 378

- inspecting with Web Inspector, 380
 - not styling, 91
 - pseudo, 92–95
 - small cap settings, 144–145
 - styling based on attributes, 96–99
 - styling based on context, 71–79
 - styling before and after, 94
 - styling children of, 89
 - tools for highlighting when rolling over, 376
 - transformation, 307–315
 - transitions between element states, 316–320
 - types in HTML, 19–21
 - types new in HTML5, 25
 - visual formatting properties. See *visual formatting properties*
- elliptical corners, 269
- Elastic, CSS Frameworks, 367
- Embedded OpenType (EOT), 127, 129
- embedded styles, adding to Web page, 38–40
- emotions, color association and, 191
- empty cells, dealing with, 225
- end point**, Webkit gradient value, 189
- End User License Agreements (EULA), for fonts, 131
- entities, character, 123
- EOT (Embedded OpenType), 127, 129
- equals sign (=), CSS tips, 37
- errors
 - troubleshooting CSS code, 372–375
 - viewing, 376
- EULA (End User License Agreements), for fonts, 131
- evolution
 - of CSS, 6–7
 - of HTML5, 22–24
- extensions
 - 2D transformations, 308–311
 - 3D transformations, 314
 - CSS browser, 11–12
 - Mozilla gradient, 188
 - transition, 318
 - Webkit gradients, 189–190

- external CSS
 - adding to Web site, 41–47
 - always locating final styles in external files, 386
 - keeping number of external style sheets to minimum, 386

F

- fantasy fonts, 122
- figures (`<figure>`), 26
- file formats, downloadable Webfonts and, 127–128
- file size, style sheet strategies and, 368
- files, creating external CSS, 41–47
- filter**
 - opacity, 298
 - radial gradient, 187
- Firebug
 - debugging CSS code in Firefox, 377–378
 - tools for analyzing/editing code, 376
- Firefox
 - color gradients, 188
 - CSS extension, 12
 - CSS support, 324
 - sizing elements, 253
 - testing code in, 391
 - tools for analyzing/editing code, 376
 - using Firebug add-on, 377
 - W3C box standard, 334

- :first-child**, 88–89

- :first-letter**, 92–93

- :first-line**, 92–93

- :first-of-type**, 88–89

- fixed**, table layout, 220–221

- fixed backgrounds, 211

- fixed positioning
 - defined, 287
 - setting, 288
 - tip, 289

- floating elements (**float**)
 - box properties and, 257–259
 - break tag clear fix, 340–341
 - creating multicolumn layout, 346–349
 - CSS fixes for, 340

- overflow fix, 342
- overview, 257–259

- :focus**, styling for interaction, 86–87

- font families

- defining fonts for Web, 129
- defining multiple font values, 147
- generic, 120–122
- overview, 119
- setting font stack, 124–125

- font properties

- adjusting size for understudy fonts, 136–138
- bolding, 142–143
- finding fonts, 126–130
- Font Squirrel, 131
- italicizing, 139–141
- multiple values, 146–149
- overview, 117–118
- putting it all together, 150
- setting font stack, 124–125
- setting size, 133–135
- small caps, 144–145
- typography on Web, 119–123
- Webfont, 132

- Font Squirrel, 131

- @font-face**, 128, 364

- fonts

- creating glossary of fonts used, 362–363
- new in CSS3, 14
- quoting names, 37
- relative sizing, 389

- font-size**

- defining multiple font values, 148
- scientific notation, 166
- setting, 133–135

- font-size-adjust**, 136–138

- font-stretch**, 139

- font-style**, 139–141, 146, 148

- font-variant**, 144–146, 148

- font-weight**, 142–143, 146, 148

- footers (`<footer>`)

- overview, 26
- using HTML5 structure, 29

- formats
 - custom pointer, 233
 - font difficulties, 131
 - font file, 127–128
 - pseudo-classes, 81
 - visual properties. See visual formatting properties
- forms
 - color palette options for, 193
 - dynamic styles for, 388
- frame tags, in HTML5, 25
- frameworks, CSS Frameworks, 367
- from**, Webkit gradient value, 190

G

- generated content
 - CSS for, 234–235
 - putting it all together, 240
- generic font families
 - overview, 120–122
 - setting font stack, 125
- glyph, 119
- gradients
 - in backgrounds, 187–190, 391
 - color values, 184
- graphics
 - adding mouse pointer, 233
 - as bullets, 217
 - styling for print, 108
- grids, multicolumn layout, 346
- grouping, elements using same styles, 62–65

H

- hacks, 325
- hanging-punctuation**, 176
- head (**<head>**)
 - overview of, 17
 - placing style links in, 386
 - using HTML5 structure, 28
- header (**<header>**)
 - choosing color palette, 193
 - defined, 17
 - overview of, 26
 - using HTML5 structure, 28

- height
 - adjusting line, 156–157
 - box properties, 246–247
 - height** property, 100
 - setting element, 251–253
 - window and document, 283–284
- hex color values
 - color keywords, 185–187
 - defined, 181–182
 - vs. decimal color values, 184
- hiding
 - cursor, 232
 - elements, 250–251, 294–295
 - empty table cells, 225
 - overflow, 255
- hierarchy, organizing based on CSS rules, 366
- horizontal alignment, of text, 162–163
- :hover**
 - overview of, 82
 - setting contrasting appearances with, 84–87
- HSL values, 183–184
- HTML (HyperText Markup Language)
 - box properties. See box properties
 - character entities, 123
 - class attributes added to, 54
 - CSS and, 8
 - defined, 16–18
 - id** added to, 56
 - inline styles added to, 35–37
 - redefining tags, 48–50
 - text control, 119
- HTML selectors
 - basics, 34
 - defining, 48–50
 - overview of, 9
- HTML5
 - applying structure, 27–31
 - basic HTML defined, 16–18
 - elements, 19–21
 - evolution of, 22–24
 - new in, 25
 - overview, 15
 - putting it all together, 32
 - structure, 26

hue, 183

hypertext links. See links

I

icon:, mimicking visitor styles, 149

ID selectors

basics, 34

defining, 55–56

overview of, 10

styling with pseudo-classes, 85

troubleshooting CSS code, 373

IDs

defining unique, 55–58, 387

rules, 10

iframes, 283

image rollovers, added to Web pages,
354–356

images

custom pointer, 233

resizing, 253

setting background, 200–207, 209–210

setting border, 271–273

using as bullets, 217

@import

Aggregate method and, 370

Dynamic method and, 371

favoring `<link>` over, 392

importing external CSS files, 46

placing at top of CSS, 364

!important

adding to declarations, 375

cascade order of, 114

making declarations, 111–112

removing before site goes live, 389

importing

CSS files, 42

external CSS files, 46

external CSS to Web pages, 43

style sheet strategies and, 368

indentation, text paragraphs, 167–168

inheritance

anchor link styles, 85

bullet positioning, 218

cascade order and, 115

element display, 250

element opacity, 299

element visibility, 294

generated content and, 235

position type, 288–289

of properties from parent, 109–110

setting space between table cells, 222

inline elements

displaying, 248

HTML selectors for, 19–20

inline styles

adding to HTML tag, 35–37

cascade order of, 114

inline-block, 250

insetting element shadow, 301

inside, bullet positioning, 218

interface chrome, background images or styles
for, 390

interface elements, dynamic styles for, 388

Internet Explorer (IE)

box model and, 247

child sibling support, 78

color gradients, 187

CSS extensions, 12

CSS3 support in IE 9, 14, 324

element opacity settings, 298–299

fixing code in, after testing in other
browsers, 391

HTML5 in, 30–31

pseudo-element syntax support, 95

universal selector support, 59

Internet Explorer (IE), CSS fixes for

correcting box model in IE6, 333–334

overview, 324

setting up conditional styles in IE, 328–332

underscore hack, 325–327

introduction section, adding at top of CSS, 362

iPhone, styles tailored for, 103–105

ISO 8859-1 character set, 120

italics

fonts, 139–141

link styles and, 352

J

JavaScript

CSS and, 8

making HTML5 work in IE, 30–31

justifying text, 175

K

kerning

text properties that are coming soon, 176

vs. tracking, 154

keywords, color, 181–182, 185–187

kludges, 153

Kuler, 195

L

:lang(), 89–90

languages

specifying character sets, 119–120

styling for, 89–90

:last-child, 88–89

:last-of-type, 88–89

layout

bullets, 217

multi-column boxes, 276

tables, 220–221

leading

adjusting, 156–157

overview of, 153

left, setting element position, 290–291

left alignment, of floating elements, 258–259

legal issues, fonts and, 131

length

of background position, 204

of element shadow, 300–301

of indentation, 167

transform values and, 307

letterforms, 117

letters

spacing, 153–154

styling first, 92–93

letter-spacing, 153–154

libraries, CSS libraries, 367

licensing fonts, 131

lightness, as color value, 183

line height

adjusting, 156–157

CSS resets for, 339

font values, 147

linear gradients, 187, 189

lines

styling first, 92–93

text decoration, 176

line-through, 172

:link

overview, 82

setting contrasting appearances with,
84–85

links (**<link>**)

adding styles to HTML documents, 386

color palette options for, 193

to external CSS, 43–45

external CSS to HTML files, 44–45

external CSS to Web pages, 43

favoring link over **@import**, 392

media queries and, 105

pseudo-classes, 80

style sheet strategies and, 368

style sheets to HTML files, 44

styling, 350–353

styling link state, 391

styling with pseudo-classes, 82–85

troubleshooting CSS code, 373–374

underlining, 88, 174

list properties

bullet style, 216, 218

creating bullets, 217

putting it all together, 227

setting multiple styles, 219

list-item, element display, 250

lists

color palette for, 193

links, 352

list-style, 219

list-style-image, 217

list-style-position, 218

list-style-type, 216

local source decoys, 129

lowercase text, 158

M

margins

- box properties, 247
- clipping element, 297
- CSS resets for, 339
- favoring over padding, 391
- positioning and, 289
- setting element, 260–262

markup languages, 17

matrix()

- 2D transformations, 310
- 3D transformations, 315

max-height, 253

max-width, 253

media

- determining cascade order, 114
- new in CSS3, 14

@media

- placing @ rules at top of CSS, 364
- tailoring Web pages to devices, 388

media query

- overview, 100–107
- tailoring Web pages to devices, 388

menu:, mimicking visitor styles, 149

menus, drop-down, 356–359

message-box:, mimicking visitor styles, 149

Meyer, Eric, 338

min-height, 253

mini-caps, 144–145

Minify CSS

- applying before launch, 392
- CSS Compressor & Minifier, 382
- minifying CSS code with, 383–384

min-width, 253

monochrome, 102, 194

monospace fonts, 121

mouse pointer, styling appearance of, 232–233

Mozilla

- color gradients, 188
- CSS extensions, 12
- rounding border corners, 270

multicolumn layout

- boxes for, 276
- overview, 346–349

multiline comment tags, 373

N

naming

- classes, 52, 56
- ID selectors, 55
- IDs, 56

navigation (<nav>)

- color palette for, 193
- defined, 26
- displaying, 295
- preventing navigation noise, 359
- styling, 350–353
- styling for print, 108
- using HTML5 structure, 28

navigation links, styling, 352

negation selector, 91

negative margins, setting, 260

negative values, setting element position, 291

nested tags

- box properties, 245
- overview of, 70

nesting comments, 67

newspaper style, 162

normal flow, 284

normal white-space, 169–170

:not, 91

nowrap white-space, 169–170

:nth-child(#), 88–89

:nth-last-of-type(#), 88–89

:nth-of-type(#), 88–89

number, transform values, 307

numbering, counting with browsers, 236–237

O

objects, stack in 3D, 292–293

oblique fonts, 139–141

offsetting outlines, 264

One For All method, style sheet strategy, 368–369

- opacity
 - color, 184
 - element settings, 298–299
 - new in CSS3, 14
- OpenType fonts (OTF), 127, 130
- Opera
 - CSS extension, 12
 - CSS support, 324
 - W3C box standard, 334
- operating system, troubleshooting code on different, 375
- order, determining cascade, 113–115
- organization scheme, choosing consistent pattern for CSS, 365
- orientation**, 102
- origin**, setting background image, 200, 207
- OTF (OpenType fonts), 127, 130
- outline-offset**, 264
- outlines
 - adding color with, 184
 - box properties, 247
 - CSS resets for, 339
 - setting element, 263–264
 - text, 175
- outside**, bullet positioning, 218
- overflow
 - controlling element, 254–256
 - fix for floating elements, 342
 - tip, 253
- overlapping text, 260
- overline**, 172

P

- padding
 - box properties, 246, 274–275
 - clipping element, 297
 - CSS resets for, 339
 - favoring margins over, 391
 - setting **background-clip**, 206
 - setting **background-origin**, 207
- page breaks, styling for print, 108
- page structure, organization based on, 365
- palette, color, 190–195
 - Color Palette Generator, 195
 - overview of, 190–192
 - selecting, 192–194
- paragraphs
 - indentation, 167–168
 - links, 352
- parent elements
 - box properties, 245
 - defining styles based on context, 71–79
 - determining cascade order, 115
 - floating, 258
 - inheriting properties from, 109–110
 - overview of, 70
- pasting, converting quotes when, 37
- percentage values
 - background position settings, 205
 - color settings, 183
- period (.), defining classes, 51, 54
- perspective**, 3D transformations, 312, 314
- perspective-origin**, 3D transformations, 313
- point
 - Mozilla gradient, 188
 - overview of, 135
- position**
 - Mozilla gradient value, 188
 - setting background image, 200, 204–205
- positioning
 - background images, 200, 204–205, 210–211
 - bullets, 218
 - captions in table, 226
 - elements, 290–291
 - objects in 3D, 292–293
 - type, 285–289
- pound (#), defining ID selectors, 55
- pre white-space**, 169–170
- preceding sibling elements, 70
- prefixes, browser, 12
- presentation, focusing on structure prior to presentation, 385
- presentation tags, in HTML5, 25
- Presto, CSS extension, 12
- primary fonts, 124

- print
 - font size for, 135
 - paragraph indentation, 167–168
 - selective styling for, 108
 - styles tailored for, 103
- progressive enhancements, designing for
 - enhanced browser features, 325
- properties
 - applying CSS to HTML elements, 21
 - box. See *box* properties
 - color. See *colors*
 - CSS, 4
 - CSS rules, 11–12
 - font. See *font* properties
 - grouping selectors, 63
 - HTML, 17–18
 - HTML selector, 48–50
 - inheriting, 109–110
 - inline style, 35
 - language, 89–90
 - link state, 82
 - list. See *list* properties
 - making declarations **important**, 111–112
 - media query, 100–102
 - shorthand, 390
 - style, 2, 96–99
 - styling before and after **content**, 94
 - table. See *table* properties
 - text. See *text* properties
 - transformation. See *transformation* properties
 - transition. See *transition* properties
 - troubleshooting CSS code, 373
 - user interface. See *user interface* properties
 - visual formatting. See *visual formatting* properties
- pseudo-classes
 - counters, 237
 - defining generated content, 234
 - styling for interaction with, 86–87
 - styling links with, 82–85
 - styling specific children with, 88–90
 - working with, 80–81
- pseudo-elements, 92–95
- punctuation, hanging, 176

- punctuation-trim**, 176
- purpose, organization based on, 365

Q

- querying media. See *media query*
- Quirks mode
 - CSS support and, 324
 - overview of, 30
- quotation marks (“ ”)
 - adding generated content, 235
 - CSS tips, 37
 - specifying style, 238–240

R

- radial gradients
 - overview of, 187
 - Webkit, 189
- radius**, Webkit gradient value, 189
- readability, creating readable style sheets, 362–366
- rectangular clipping, 296–297
- relative font sizes, 135, 389
- relative positioning, 286, 288
- rendering engines, 4
- repeat**
 - background image, 200, 203
 - border image, 273
- repertoire of characters, 119
- repetition, avoiding unnecessary, 389
- resets, CSS
 - Eric Meyer’s reset, 338
 - overview, 335–336
 - simple example, 336–337
 - what you should reset, 339
 - Yahoo!’s YUI2 reset, 337
- resize**, 276
- resolution**, 102
- reusable classes, 51–54
- RGB color values, 181–182, 390
- right alignment
 - floating elements, 258–259
 - setting element position, 290–291
- rotate()**
 - 2D transformations, 310
 - 3D transformations, 314

round, setting border image, 273

rules

- adding to text files, 42–43
- background**, 208–211
- combining into selector lists, 388
- CSS, 11–12, 34
- CSS3, 9–10
- deleting and retyping when troubleshooting, 375
- @font-face**, 128
- @media**, 106–107
- organizational hierarchy based on, 366
- placing at top of CSS, 364
- <style>**, 40
- tools for viewing, 376
- troubleshooting CSS code, 374

run-in, element display, 250

S

Safari

- 3D transformations, 311
- CSS extension, 12
- CSS support, 324
- testing code in, 391
- tools for analyzing/editing code, 376
- W3C box standard, 334
- Web Inspector, 379–380
- Webkit gradients, 189–190

sans-serif fonts, 121

saturation, color values, 183

Scalable Vector Graphics (SVG)

- defined, 18
- defining source, 130
- fonts and file formats, 128

scale()

- 2D transformations, 310
- 3D transformations, 315

scientific notation, text alignment, 166

screen, font size for, 135

scrolling

- background images, 210
- overflow, 256

searches, finding fonts, 126–130

sections (**<section>**)

- for keeping CSS organized, 364
- overview of, 26

::selection, 95

selective styling

- cascade order in, 113–115
- context determining, 71–79
- element family tree, 70
- !important** declarations, 111–112
- inheritance and, 109–110
- media query and, 100–107
- not styling an element, 91
- overview, 69
- for print, 108
- pseudo-class interactions, 86–87
- pseudo-class links, 82–85
- pseudo-elements and, 92–95
- putting it all together, 116
- specific children with pseudo-classes, 88–90

- tag attributes determining, 96–99
- working with pseudo-classes, 80–81

selector lists, combining rules into, 388

selectors

- attribute, 96
- basic CSS, 34
- class, 51–54
- combinatory, 71–79
- in CSS rules, 11
- grouping, 62–63
- HTML, 48–50
- HTML for block-level elements, 20
- HTML for inline elements, 19
- ID, 55–56
- negation, 91
- organization based on selector type, 365
- overview, 9–10
- universal, 59

semicolon (;)

- character entities, 123
- troubleshooting CSS code and, 373

serif fonts, 120

service bureaus, Webfont, 132

- SGML (Standard Generalized Markup Language), 17
- shadows
 - adding to text, 160–161
 - visual formatting properties, 300–301
- shape**, Mozilla gradient value, 188
- shorthand
 - background, 208–211
 - properties, 390
 - setting multiple list styles, 219
- sibling elements
 - defined, 70
 - opacity of, 299
 - styling based on context, 76–79
- single quotes (' ')
 - CSS tips, 37
 - specifying style, 238–239
- size**
 - Mozilla gradient value, 189
 - setting background image, 200, 205
- sizing
 - adjusting understudy fonts, 136–138
 - background images, 200, 205
 - boxes, 276
 - choosing fonts with similar size, 125
 - defining multiple font values, 147–148
 - setting elements, 252–253
 - setting fonts, 133–135
- skew()**, 2D transformations, 310
- slash (/), adding comments to CSS, 66–67
- small caps, 144–145
- small-caption:**, mimicking visitor styles, 149
- spacing
 - between cells, 222
 - text, 153–157
- ** tags, creating HTML tags, 54
- specificity
 - best practices for, 388–389
 - determining cascade order, 115
 - organization of code and, 392
- spread, setting element shadow, 301
- sprite technique
 - best practices for, 390
 - overview of, 354
- square brackets ([]), in attribute selector syntax, 96
- stack, font, 124–125
- stacking objects in 3D, 292–293
- stacking order, setting positioning type, 285
- Standard Generalized Markup Language (SGML), 17
- standards, evolution of CSS and, 6–7
- Standards mode, CSS support and, 324
- start point**, Webkit gradient value, 189
- states
 - pseudo-class, 80
 - transitions between element, 316–320
- static positioning, 285, 288
- status-bar:**, mimicking visitor styles, 149
- stop**
 - Mozilla gradient value, 189
 - Webkit gradient value, 190
- strategies, style sheet
 - Aggregate method, 370–371
 - Divide and Conquer method, 369–370
 - Dynamic method, 371
 - One For All method, 368–369
 - overview, 368
- stretch**, to fill borders, 273
- stretching fonts, 139
- stretching images, 271–273
- Strict mode, CSS support and, 324
- strings, defining styles based on attributes, 97–99
- structural pseudo-classes, 80–81
- structure
 - applying HTML5 now, 27–31
 - focusing on structure prior to presentation, 385
 - HTML, 16–17
 - HTML5, 25–26
- style sheets
 - Aggregate method, 370–371
 - best practices, 385–392
 - comments in, 66–67
 - creating minified version of CSS code, 382–384
 - creating readable, 362–366
 - CSS Libraries and Frameworks and, 367

- debugging with Firebug, 376–378
- debugging with Web Inspector, 379–380
- Divide and Conquer method, 369–370
- Dynamic method, 371
- keeping number of external style sheets to minimum, 386
- One For All method, 368–369
- overview, 361
- strategies for, 368
- troubleshooting CSS code, 372–375
- validating CSS code, 381

styles (**<style>**)

- 3D transformations, 312
- browser inherited, 18
- bullets, 216
- embedding, 38–40
- font, 139–141
- for interface chrome, 390
- link state, 391
- media queries for specifying, 105
- multiple font values, 146
- multiple lists, 219
- navigation and links, 350–353
- overview of, 2
- placing style links in **<head>** section, 386
- resetting CSS to reduce bad styles, 335
- selective. *See* selective styling
- troubleshooting CSS code, 374
- universal, 59–61

subscript, 166

superscript, 166

SVG. *See* Scalable Vector Graphics (SVG)

syntax

- attribute selector, 96
- child selector, 74
- CSS rule, 11
- descendant selector, 71
- HTML5, 26
- !important**, 111
- @media**, 106
- media query, 100
- pseudo-class, 80
- pseudo-element, 92
- sibling selector, 76

T

table of contents (TOC), 362–363

table properties

- bullet positions, 218
- bullet style, 216
- caption positioning, 226
- collapsing borders between cells, 223–224
- creating bullets, 217
- dealing with empty cells, 225
- layout, 220–221
- overview, 213–215
- putting it all together, 227
- space between cells, 222

table-layout property, 220–221

tables

- color palette, 193
- element display, 250
- links, 352
- separating borders between table cells, 223–224
- showing/hiding empty table cells, 225

tags

- adding inline styles to HTML, 35–37
- applying HTML5 structure, 27–31
- defining styles based on attributes, 96–99
- embedding styles, 38–40
- HTML, 16–18
- HTML5, 26
- language, 90
- redefining HTML, 48–50
- styling anchor tags with pseudo-classes, 82
- sub** and **sup**, 166

tetrad color, 194

tex-overflow, 276

text

- bulleting. *See* bullets
- color palette for, 193
- color settings, 196–197
- CSS resets for underlines, 339
- external CSS and, 42
- font settings. *See* font properties
- negative margins, 260
- new in CSS3, 13
- overflow, 256

text (*continued*)

- styling for print, 108
- underlining tip, 88

text properties

- case, 158–159
- decoration, 172–174
- drop shadow, 160–161
- features coming soon, 175–176
- horizontal alignment, 162–163
- overview, 151–152
- paragraph indentation, 167–168
- putting it all together, 177
- spacing, 153–157
- vertical alignment, 164–166
- white space, 169–171

text-align, 162–163

text-decoration, 172–174

text-indent, 167–168

text-justify, 175

text-outline, 175

text-shadow, 160–161, 184

text-transform, 158–159

text-wrap, 175

tilde sign (“~”), defining general sibling selectors, 78

tiling images, to fill borders, 271–273

timing, transition values, 319–320

to, Webkit gradient value, 190

TOC (table of contents), 362–363

top, setting element position, 290–291

tracking

- overview of, 153
- vs. kerning, 154

transformation properties

- element, 307–315
- overview of, 303–306
- putting it all together, 320

transformations, new in CSS3, 13

transition properties

- between element states, 316–320
- overview, 303–306
- putting it all together, 320

transitions, new in CSS3, 13

translate()

- 2D transformations, 310
- 3D transformations, 315

transparent

- overview of, 182
- setting background color, 198–199

transparent colors, styling for print, 108

triad color, 194

Trident, CSS extension, 12

troubleshooting CSS code, 372–375

TrueType fonts (TTF)

- defining source, 130
- fonts and file formats, 127

TTF (TrueType fonts)

- defining source, 130
- fonts and file formats, 127

type family, 119

typefaces

- adding overrides, 125
- creating glossary of fonts used, 362–363

typography

- Eric Meyer’s reset for, 338
- overview of, 117
- text and, 151
- on Web, 119–123

Typogridphy, CSS Frameworks, 367

typos, troubleshooting CSS code, 373

U

underlining

- CSS resets for, 339
- link styles and, 352
- text decoration properties, 172, 174
- text tip, 88
- vs. italicizing, 141

underscore hack

- overview of, 325–326
- using the IE underscore hack, 325–326

understudy fonts

- adjusting size for, 136–138
- overview of, 125

Unicode Transformation Format-8 bit (UTF-8), 120

- universal selectors
 - defining, 59
 - sibling selectors, 78
 - simple CSS reset, 336
 - styling descendants, 71, 73
- universal styles, 59–61
- uppercase text, 158
- URLs, adding generated content, 235
- user interface properties
 - generated content, 234–235
 - mouse pointer appearance, 232–233
 - overview, 229–231
 - putting it all together, 240
 - specifying quotation style, 238–239
 - teaching browsers to count, 236–237
- user styles, cascade order of, 113
- UTF-8 (8-bit Unicode Transformation Format), 120

V

- validating CSS code, 381
- values
 - 3D transformation, 314
 - backface-visibility** and **perspective-origin**, 313
 - background**, 208
 - background-attachment** and **background-position**, 204
 - background-clip**, 206
 - background-color**, 199
 - background-image**, 201
 - background-origin**, 207
 - background-repeat**, 203
 - background-size**, 206
 - border**, 265
 - border-collapse**, 223
 - border-image**, 271
 - border-radius**, 268
 - border-spacing**, 222
 - border-style**, 267
 - box-shadow**, 300
 - caption-side**, 226
 - choosing color values, 181–184
 - clip**, 296
 - color**, 197
 - content**, 234
 - counter**, 236
 - in CSS rules, 11
 - cursor**, 232
 - defining styles based on attributes, 96–99
 - display**, 247
 - element width and height, 252–253
 - empty-cells**, 225
 - float** and **clear**, 258
 - font-size-adjust**, 136
 - font-weight**, 142
 - inherit**, 110
 - letter-spacing**, 153
 - line-height**, 156
 - list-style**, 219
 - list-style-image**, 217
 - list-style-position**, 218
 - list-style-type**, 216
 - margin**, 260
 - media, 100
 - Mozilla gradient, 188
 - opacity**, 298
 - outline**, 263
 - outline-offset**, 264
 - overflow**, 255
 - padding**, 274
 - perspective** and **transform-style**, 312
 - positioning, 285
 - quotes**, 238
 - setting multiple font, 146–149
 - specifying units for, 385
 - table-layout**, 220
 - text-align**, 162
 - text-decoration**, 172
 - text-indent**, 167
 - text-shadow**, 160–161
 - text-transform**, 158–159
 - top**, **left**, **bottom**, **right**, 290
 - transformation, 307
 - transition**, 318–319
 - vertical-align**, 164
 - visibility**, 294
 - white-space**, 169–171
 - word-spacing**, 155
 - z-index**, 292

- variants
 - defining multiple font values, 146
 - setting small caps, 144–145
- versions, CSS, 7
- vertical alignment
 - CSS resets for, 339
 - of text, 164–166
- vertical-align**, 164–166
- viewports
 - media queries for, 102, 104
 - understanding, 283–284
- visibility
 - adding **backface-visibility**, 313
 - overflow, 255
 - setting element, 294–297
 - setting positioning type, 285
 - vs. **display**, 251
- :visited**
 - overview of, 82
 - setting contrasting appearances with, 84–85
- visitors, mimicking styles, 149
- visual formatting properties
 - clipping element's visible area, 296–297
 - element opacity, 298–299
 - element position, 290–291
 - element shadow, 300–301
 - element visibility, 294–295
 - overview, 279–282
 - positioning type, 285–289
 - putting it all together, 302
 - stacking objects in 3D, 292–293
 - understanding window and document, 283–284

W

- W3C (World Wide Web Consortium)
 - boxes and, 333–334
 - CSS validator, 381
 - evolution of CSS, 6–7
 - evolution of HTML5, 22–24
 - overview of, 10

- Web
 - defining fonts for, 128–130
 - safe fonts for, 126–127
 - typography on, 119–123
- Web fonts, new in CSS3, 14
- Web forms, new in HTML5, 25
- Web HyperText Application Technology Working Group (WHATWG), 23
- Web Inspector
 - debugging CSS code with, 379–380
 - tools for analyzing/editing code, 376
- Web Open Font Format (WOFF), 128, 130
- Web pages
 - CSS in, 3–5
 - embedded styles on, 38–40
 - image rollovers added to, 354–356
 - understanding window and document, 283–284
- Web sites
 - adding styles to, 41–47
 - danger of using inline styles, 37
- Webfonts
 - defined, 123
 - defining for element, 128–130
 - downloadable, 127–128
 - overview of, 117
 - service bureaus, 132
- Webkit
 - CSS extension, 12
 - rounding border corners, 270
- weight, font values, 142–143, 146
- Western color associations, 191
- WHATWG (Web HyperText Application Technology Working Group), 23
- white space, controlling text, 169–171
- width
 - border, 265–267
 - box properties, 246
 - column, 349
 - element, 251–253
 - width** property, 100
 - window and document, 283–284

window

floating elements in, 257–259

understanding, 283–284

WOFF (Web Open Font Format), 128, 130

word-spacing, 155

wrapping text, 175

X

XHTML, 22

XHTML2, 23

XHTML5, 24

Y

Yahoo!

YUI grids, 367

YUI2 reset, 337

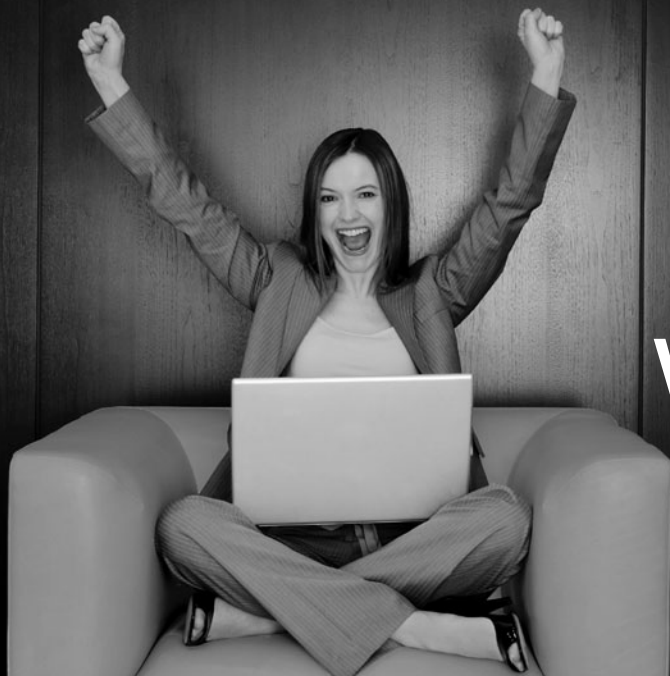
YUI grids, CSS Frameworks, 367

YUI2 reset, 337

Z

zebra striping, 193

z-index, 292–293



WATCH
READ
CREATE

Meet Creative Edge.

A new resource of unlimited books, videos and tutorials for creatives from the world's leading experts.

Creative Edge is your one stop for inspiration, answers to technical questions and ways to stay at the top of your game so you can focus on what you do best—being creative.

All for only \$24.99 per month for access—any day any time you need it.

creative
edge

peachpit.com/creativeedge