VISUAL QUICKSTART GUIDE

Get up and running in no time!

Register your book for free video training!
www.peachpit.com/register

Downloadable examples from the companion Web site!

HTML5 and CSS3

Seventh Edition

ELIZABETH CASTRO • BRUCE HYSLOP

LEARN THE QUICK AND EASY WAY!

VISUAL QUICKSTART GUIDE

HTML5 and CSS3

Seventh Edition

ELIZABETH CASTRO • BRUCE HYSLOP

Peachpit Press

HTML5 and CSS3, Seventh Edition: Visual QuickStart Guide
Elizabeth Castro and Bruce Hyslop

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.

Editor: Clifford Colby
Development editor: Robyn G. Thomas
Production editor: Cory Borman
Compositor: David Van Ness
Copyeditor: Scout Festa
Proofreader: Nolan Hester
Technical editors: Michael Bester and Chris Casciano
Indexer: Valerie Haynes Perry
Cover design: RHDG/Riezebos Holzbaur Design Group, Peachpit Press
Interior design: Peachpit Press
Logo design: MINE™ www.minesf.com

ISBN-13: 978-0-321-71961-4
ISBN-10: 0-321-71961-1

9 8 7 6 5 4 3 2 1

Printed and bound in the United States of America

To family.

Acknowledgments

Writing the acknowledgments is one of the most daunting challenges of working on a book, because you want to be sure to convey your appreciation of everyone properly. This book is the result of the support, tireless work, and good spirits of a lot of people. I hope to do them all justice, and I hope that you'll indulge me for a bit while I thank them.

A most sincere thank you goes out to:

Nancy Aldrich-Ruenzel and Nancy Davis, for entrusting me with this edition of a book that has been important to Peachpit for many years.

Cliff Colby, for recommending me and making this possible; for his confidence in me and his patience, flexibility, and guidance; and for countless conversations and lots of laughs.

Robyn Thomas, for her tremendous effort in keeping us all on track, wrangling countless documents, making thoughtful edits and suggestions, and providing regular words of encouragement, which were always a boost.

Michael Bester, for all the spot-on feedback and suggestions, catching technical errors and omissions, and helping us get the right message across to readers. It was a real pleasure working with him on another book.

Chris Casciano, in the same vein, for all your technical expertise, suggestions, and crucial feedback. I really appreciated your joining us in the final weeks; we were lucky to have you.

Cory Borman, for expertly overseeing the production of the book and creating diagrams in a pinch, and for his good humor.

Scout Festa, for carefully correcting grammar and punctuation, tightening up language, ensuring the accuracy of figure and chapter references, and, overall, providing an all-important level of polish.

David Van Ness, for his great care laying out the pages and for his proficiency and attention to detail.

Nolan Hester, for lending his expertise to the effort of reviewing the laid-out pages.

Valerie Haynes Perry, for handling the critical task of creating an effective index on which readers will rely time and again.

The numerous marketing, sales, and other folks at Peachpit for working behind the scenes to make the book successful.

My family and friends, for checking in on my progress and providing occasional, welcome breaks from writing. Thanks to those friends in particular who probably tired of hearing me say often that I couldn't get together, but who kept asking anyway.

Robert Reinhardt, as always, for getting me started in writing books and for his guidance as I was embarking on this one.

The Web community, for your innovations and for sharing your knowledge so that others may benefit (I've cited many of you throughout the book).

To you readers, for your interest in learning about HTML and CSS and for selecting this book; I know you have a lot of others from which to choose. I hope the book serves you well.

Thank you so much to the following contributing authors. Readers have a more valuable book because of your efforts, for which I'm grateful. I'd also like to extend my apologies to Erik Vorhes that we weren't able to fit Appendixes A and B in the book. Readers who see them on the book's site will surely appreciate your work.

In alphabetical order by last name, the contributing authors are:

Scott Boms (Chapter 14)
Scott is an award-winning designer, writer, and speaker who has partnered with organizations such as PayPal, HSBC, Hyundai, DHL, XM Radio, *Toronto Life* magazine, and Masterfile during his more than 15 years of working on the Web. When he's away from the computer, you might find him shooting Polaroids; playing drums with his band, George; or enjoying time with his wonderful wife and two children. He's @scottboms on Twitter.

Ian Devlin (Chapter 17)
Ian Devlin is an Irish Web developer, blogger, and author who enjoys coding and writing about emerging Web technologies such as HTML5 and CSS3. In addition to front-end development, Ian also builds solutions with back-end technologies such as .NET and PHP. He has recently written a book, *HTML5 Multimedia: Develop and Design* (Peachpit Press, 2011).

Seth Lemoine (Chapters 5 and 16)
Seth Lemoine is a software developer and teacher in Atlanta. For over ten years, he's worked on challenging projects to see what's possible, with technologies from HTML, JavaScript, and CSS to Objective-C and Ruby. Whether it's finding innovative ways to teach HTML5 and CSS to his students or perfecting a Schezuan recipe in his outdoor wok, being creative is his passion.

Erik Vorhes (Appendixes A and B, available on the book's Web site)
Erik Vorhes creates things for the Web with VSA Partners and is managing editor for Typedia (http://typedia.com/). He lives and works in Chicago.

Brian Warren (Chapter 13)
Brian Warren is a senior designer at Happy Cog in Philadelphia. When he's not writing or designing, he spends his time playing with his beautiful family, listening to music, and brewing beer. He blogs, intermittently, at http://begoodnotbad.com.

And, finally, I'd like to extend a special thank you to Elizabeth Castro. She created the first edition of this book more than 15 years ago and nurtured her audience with each edition that followed. Her style of teaching has resonated with literally hundreds of thousands of readers over the years. I'm extremely grateful for the opportunity to be part of this book, and I was very mindful of doing right by both it and readers while working on this edition.

—Bruce

Contents at a Glance

	Acknowledgments	iv
	Introduction	xv
Chapter 1	Web Page Building Blocks	1
Chapter 2	Working with Web Page Files	25
Chapter 3	Basic HTML Structure	41
Chapter 4	Text	99
Chapter 5	Images	147
Chapter 6	Links	165
Chapter 7	CSS Building Blocks	179
Chapter 8	Working with Style Sheets	197
Chapter 9	Defining Selectors	213
Chapter 10	Formatting Text with Styles	241
Chapter 11	Layout with Styles	275
Chapter 12	Style Sheets for Mobile to Desktop	327
Chapter 13	Working with Web Fonts	353
Chapter 14	Enhancements with CSS3	371
Chapter 15	Lists	397
Chapter 16	Forms	417
Chapter 17	Video, Audio, and Other Multimedia	449
Chapter 18	Tables	489
Chapter 19	Working with Scripts	497
Chapter 20	Testing & Debugging Web Pages	505
Chapter 21	Publishing Your Pages on the Web	521
	Index	529

Table of Contents

Acknowledgments iv

Introduction xv

HTML and CSS in Brief xvi
Progressive Enhancement: A Best Practice xviii
Is This Book for You? xx
How This Book Works. xxii
Companion Web Site xxiv

Chapter 1 Web Page Building Blocks 1

A Basic HTML Page 3
Semantic HTML: Markup with Meaning 6
Markup: Elements, Attributes, and Values 13
A Web Page's Text Content 16
Links, Images, and Other Non-Text Content 17
File Names 19
URLs 20
Key Takeaways. 24

Chapter 2 Working with Web Page Files 25

Planning Your Site 26
Creating a New Web Page 28
Saving Your Web Page 30
Specifying a Default Page or Homepage 33
Editing Web Pages 35
Organizing Files 36
Viewing Your Page in a Browser 37
The Inspiration of Others 39

Chapter 3 Basic HTML Structure 41

Starting Your Web Page 43
Creating a Title. 46
Creating Headings 48
Understanding HTML5's Document Outline 50
Grouping Headings 58
Common Page Constructs 60

Creating a Header 61
Marking Navigation 64
Creating an Article 68
Defining a Section 72
Specifying an Aside 75
Creating a Footer 80
Creating Generic Containers 84
Improving Accessibility with ARIA 88
Naming Elements with a Class or ID 92
Adding the Title Attribute to Elements 95
Adding Comments 96

Chapter 4 Text 99

Starting a New Paragraph 100
Adding Author Contact Information 102
Creating a Figure 104
Specifying Time 106
Marking Important and Emphasized Text 110
Indicating a Citation or Reference 112
Quoting Text 113
Highlighting Text 116
Explaining Abbreviations 118
Defining a Term 120
Creating Superscripts and Subscripts 121
Noting Edits and Inaccurate Text 124
Marking Up Code 128
Using Preformatted Text 130
Specifying Fine Print 132
Creating a Line Break 133
Creating Spans 134
Other Elements 136

Chapter 5 Images 147

About Images for the Web 148
Getting Images 152
Choosing an Image Editor 153
Saving Your Images 154
Inserting Images on a Page 156
Offering Alternate Text 157
Specifying Image Size 158
Scaling Images with the Browser 160

Scaling Images with an Image Editor 161
Adding Icons for Your Web Site 162

Chapter 6 Links 165

The Anatomy of a Link 166
Creating a Link to Another Web Page. 167
Creating Anchors 172
Linking to a Specific Anchor 174
Creating Other Kinds of Links 175

Chapter 7 CSS Building Blocks 179

Constructing a Style Rule. 181
Adding Comments to Style Rules 182
The Cascade: When Rules Collide. 184
A Property's Value. 188

Chapter 8 Working with Style Sheets 197

Creating an External Style Sheet 198
Linking to External Style Sheets 200
Creating an Embedded Style Sheet. 202
Applying Inline Styles. 204
The Importance of Location 206
Using Media-Specific Style Sheets 208
Offering Alternate Style Sheets 210
The Inspiration of Others: CSS. 212

Chapter 9 Defining Selectors 213

Constructing Selectors 214
Selecting Elements by Name 216
Selecting Elements by Class or ID. 218
Selecting Elements by Context 221
Selecting Part of an Element. 227
Selecting Links Based on Their State 230
Selecting Elements Based on Attributes 232
Specifying Groups of Elements 236
Combining Selectors 238
Selectors Recap 240

Chapter 10 Formatting Text with Styles 241

Choosing a Font Family 243
Specifying Alternate Fonts 244
Creating Italics 246
Applying Bold Formatting 248
Setting the Font Size 250
Setting the Line Height 255
Setting All Font Values at Once 256
Setting the Color 258
Changing the Text's Background 260
Controlling Spacing 264
Adding Indents 265
Setting White Space Properties 266
Aligning Text 268
Changing the Text Case 270
Using Small Caps 271
Decorating Text 272

Chapter 11 Layout with Styles 275

Considerations When Beginning a Layout 276
Structuring Your Pages 279
Styling HTML5 Elements in Older Browsers 286
Resetting or Normalizing Default Styles 290
The Box Model 292
Changing the Background 294
Setting the Height or Width for an Element 298
Setting the Margins around an Element 302
Adding Padding around an Element 304
Making Elements Float 306
Controlling Where Elements Float 308
Setting the Border 311
Offsetting Elements in the Natural Flow 314
Positioning Elements Absolutely 316
Positioning Elements in 3D 318
Determining How to Treat Overflow 320
Aligning Elements Vertically 322
Changing the Cursor 323
Displaying and Hiding Elements 324

Chapter 12 Style Sheets for Mobile to Desktop 327

Mobile Strategies and Considerations 328
Understanding and Implementing Media Queries. . . 333
Building a Page that Adapts with Media Queries . . . 340

Chapter 13 Working with Web Fonts 353

What Is a Web Font?. 354
Where to Find Web Fonts. 356
Downloading Your First Web Font. 358
Working with **`@font-face`** 360
Styling Web Fonts and Managing File Size. 365

Chapter 14 Enhancements with CSS3 371

Understanding Vendor Prefixes 373
A Quick Look at Browser Compatibility. 375
Using Polyfills for Progressive Enhancement 376
Rounding the Corners of Elements 378
Adding Drop Shadows to Text 382
Adding Drop Shadows to Other Elements 384
Applying Multiple Backgrounds 388
Using Gradient Backgrounds 390
Setting the Opacity of Elements. 394

Chapter 15 Lists 397

Creating Ordered and Unordered Lists. 398
Choosing Your Markers. 401
Choosing Where to Start List Numbering. 403
Using Custom Markers 404
Controlling Where Markers Hang 406
Setting All List-Style Properties at Once 407
Styling Nested Lists 408
Creating Description Lists 412

Chapter 16 Forms 417

Creating Forms 419
Processing Forms 421
Sending Form Data via Email. 424
Organizing the Form Elements. 426
Creating Text Boxes. 428

Creating Password Boxes 431
Creating Email, Telephone, and URL Boxes 432
Labeling Form Parts 434
Creating Radio Buttons 436
Creating Select Boxes 438
Creating Checkboxes 440
Creating Text Areas 441
Allowing Visitors to Upload Files 442
Creating Hidden Fields 443
Creating a Submit Button 444
Using an Image to Submit a Form 446
Disabling Form Elements 447
New HTML5 Features and Browser Support 448

Chapter 17 Video, Audio, and Other Multimedia 449

Third-Party Plugins and Going Native 451
Video File Formats 452
Adding a Single Video to Your Web Page 453
Exploring Video Attributes 454
Adding Controls and Autoplay to Your Video 455
Looping a Video and Specifying a Poster Image . . . 457
Preventing a Video from Preloading 458
Using Video with Multiple Sources 459
Multiple Media Sources and the Source Element . . . 460
Adding Video with Hyperlink Fallbacks 461
Adding Video with Flash Fallbacks 463
Providing Accessibility 467
Adding Audio File Formats 468
Adding a Single Audio File to Your Web Page 469
Adding a Single Audio File with Controls to Your Web Page 470
Exploring Audio Attributes 471
Adding Controls and Autoplay to Audio in a Loop . . . 472
Preloading an Audio File 473
Providing Multiple Audio Sources 474
Adding Audio with Hyperlink Fallbacks 475
Adding Audio with Flash Fallbacks 476
Adding Audio with Flash and a Hyperlink Fallback . . . 478
Getting Multimedia Files 480
Considering Digital Rights Management (DRM) 481
Embedding Flash Animation 482

Embedding YouTube Video 484
Using Video with Canvas 485
Coupling Video with SVG 486
Further Resources 487

Chapter 18 Tables 489

Structuring Tables 490
Spanning Columns and Rows 494

Chapter 19 Working with Scripts 497

Loading an External Script 499
Adding an Embedded Script 502
JavaScript Events 503

Chapter 20 Testing & Debugging Web Pages 505

Trying Some Debugging Techniques 506
Checking the Easy Stuff: General 508
Checking the Easy Stuff: HTML 510
Checking the Easy Stuff: CSS 512
Validating Your Code 514
Testing Your Page 516
When Images Don't Appear 519
Still Stuck? 520

Chapter 21 Publishing Your Pages on the Web 521

Getting Your Own Domain Name 522
Finding a Host for Your Site 523
Transferring Files to the Server 525

Index 529

Bonus chapters mentioned in this eBook are available after the index.

Appendix A A1

Appendix B B1

This page intentionally left blank

Introduction

Whether you are just beginning your venture into building Web sites or have built some before but want to ensure that your knowledge is current, you've come along at a very exciting time in the industry.

How we code and style pages, the browsers in which we view the pages, and the devices on which we view the browsers have all advanced substantially the past few years. Once limited to browsing the Web from our desktop computers or laptops, we can now take the Web with us on any number of devices: phones, tablets, and, yes, laptops and desktops, and more.

Which is as it should be, because the Web's promise has always been the dissolution of boundaries—the power to share and access information freely from any metropolis, rural community, or anywhere in between, from any Web-enabled device. In short, the Web's promise lies in its universality. And the Web's reach continues to expand as technology finds its ways to communities that were once shut out.

Adding to the Web's greatness is that anyone is free to create and launch a site. This book shows you how. It is ideal for the beginner with no knowledge of HTML or CSS who wants to begin to create Web pages. You'll find clear, easy-to-follow instructions that take you through the process of creating pages step by step. Lastly, the book is a helpful guide to keep handy. You can look up topics in the table of contents or index and consult just those subjects about which you need more information.

HTML and CSS in Brief

At the root of the Web's success is a simple, text-based markup language that is easy to learn and that any device with a basic Web browser can read: HTML. Every Web page requires at least some HTML; it wouldn't be a Web page without it.

As you will learn in greater detail as you move through this book, HTML is used to define your content's meaning, and CSS is used to define how your content and Web page will look. Both HTML pages and CSS files (*style sheets*) are text files, making them easy to edit. You can see snippets of HTML and CSS in "How This Book Works," near the end of this introduction.

You'll dive into learning a basic HTML page right off the bat in Chapter 1, and you'll begin to learn how to style your pages with CSS in Chapter 7. See "What this book will teach you" for an overview of all the chapters and a summary of the topics covered.

What is HTML5?

It helps to know some basics about the origins of HTML in order to understand HTML5. HTML began in the early 1990s as a short document that detailed a handful of elements used to build Web pages. Many of those elements were for describing Web page content such as headings, paragraphs, and lists. HTML's version number has increased as the language has evolved with the introduction of other elements and adjustments to its rules. The most current version is HTML5.

HTML5 is a natural evolution of earlier versions of HTML and strives to reflect the needs of both current and future Web sites. It inherits the vast majority of features from its predecessors, meaning that if you coded HTML before HTML5 came on the scene, you already know a lot of HTML5. This also means that much of HTML5 works in both old and new browsers; being backward compatible is a key design principle of HTML5 (see www.w3.org/TR/html-design-principles/).

HTML5 also adds a bevy of new features. Many are straightforward, such as additional elements (**`article`**, **`section`**, **`figure`**, and many more) that are used to describe content. Others are quite complex and aid in creating powerful Web applications. You'll need to have a firm grasp of creating Web pages before you can graduate to the more complicated features that HTML5 provides. HTML5 also introduces native audio and video playback to your Web pages, which the book also covers.

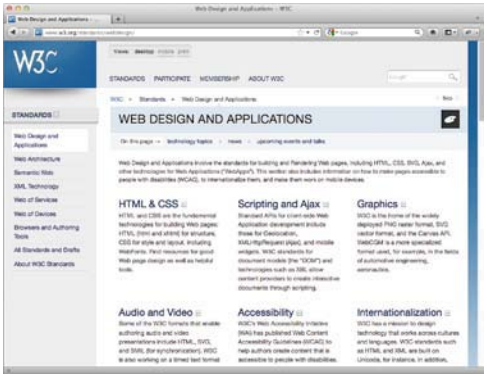
What is CSS3?

The first version of CSS didn't exist until after HTML had been around for a few years, becoming official in 1996. Like HTML5 and its relationship to earlier versions of HTML, CSS3 is a natural extension of the versions of CSS that preceded it.

CSS3 is more powerful than earlier versions of CSS and introduces numerous visual effects, such as drop shadows, text shadows, rounded corners, and gradients. (See "What this book will teach you" for details of what's covered.)

Web standards and specifications

You might be wondering who created HTML and CSS in the first place, and who continues to evolve them. The World Wide Web Consortium (W3C)—directed by the inventor of the Web and HTML, Tim Berners-Lee—is the organization responsible for shepherding the development of Web standards. *Specifications* (or *specs*, for short) are documents that define the parameters



A The W3C site is the industry's primary source of Web-standards specifications.

of languages like HTML and CSS. In other words, specs standardize the rules. Follow the W3C's activity at www.w3.org A.

For a variety of reasons, another organization—the Web Hypertext Application Technology Working Group (WHATWG, found at www.whatwg.org)—is developing the HTML5 specification. The W3C incorporates WHATWG's work into its official version of the in-progress spec.

With standards in place, we can build our pages from the agreed-upon set of rules, and browsers—like Chrome, Firefox, Internet Explorer (IE), Opera, and Safari—can be built to display our pages with those rules in mind. (On the whole, browsers implement the standards well. Older versions of IE, especially IE6, have some issues.

Specifications go through several stages of development before they are considered final, at which point they are dubbed a *Recommendation* (www.w3.org/2005/10/Process-20051014/tr).

Parts of the HTML5 and CSS3 specs are still being finalized, but that doesn't mean you can't use them. It just takes time (literally years) for the standardization process to run its course. Browsers begin to implement a spec's features long before it becomes a Recommendation, because that informs the spec development process itself. So browsers already include a wide variety of features in HTML5 and CSS3, even though they aren't Recommendations yet.

On the whole, the features covered in this book are well entrenched in their respective specs, so the risk of their changing prior to becoming a Recommendation is minimal. Developers have been using many HTML5 and CSS3 features for some time. So can you.

Progressive Enhancement: A Best Practice

I began the introduction by speaking of the universality of the Web—the notion that information on the Web should be accessible to all. *Progressive enhancement* helps you build sites with universality in mind. It is not a language, rather it's an approach to building sites that Steve Champeon created in 2003 (http://en.wikipedia.org/wiki/Progressive_enhancement).

The idea is simple but powerful: Start your site with HTML content and behavior that is accessible to all visitors **A**. To the *same* page, add your design with CSS **B** and add additional behavior with JavaScript, typically loading them from external files (you'll learn how to do this).

The result is that devices and browsers capable of accessing basic pages will get the simplified, default experience; devices and browsers capable of viewing more-robust sites will see the enhanced version. The experience on your site doesn't have to be the same for everyone, as long as your content is accessible. In essence, the idea behind progressive enhancement is that everyone wins.

A A basic HTML page with no custom CSS applied to it. This page may not look great, but the information is accessible—and that's what's important. Even browsers from near the inception of the Web more than 20 years ago can display this page; so too can the oldest of mobile phones with Web browsers. And *screen readers*, software that reads Web pages aloud to visually impaired visitors, will be able to navigate it easily.

B The same page as viewed in a browser that supports CSS. It's the same information, just presented differently. Users with more capable devices and browsers get an enhanced experience when visiting the page.

This book teaches you how to build progressively enhanced sites even if it doesn't always explicitly call that out while doing so. It's a natural result of the best practices imparted throughout the book.

However, Chapters 12 and 14 do address progressive enhancement head on. Take an early peek at those if you're interested in seeing how the principle of progressive enhancement helps you build a site that adapts its layout based on a device's screen size and browser capabilities, or how older browsers will display simplified designs while modern browsers will display ones enhanced with CSS3 effects.

Progressive enhancement is a key best practice that is at the heart of building sites for everyone.

Is This Book for You?

This book assumes no prior knowledge of building Web sites. So in that sense, it is for the absolute beginner. You will learn both HTML and CSS from the ground up. In the course of doing so, you will also learn about features that are new in HTML5 and CSS3, with an emphasis on the ones that designers and developers are using today in their daily work.

But even if you *are* familiar with HTML and CSS, you still stand to learn from this book, especially if you want to get up to speed on much of the latest in HTML5, CSS3, and best practices.

What this book will teach you

We've added approximately 125 pages to this book since the previous edition in order to bring you as much material as possible. (The very first edition of the book, published in 1996, had 176 pages *total*.) We've also made substantial updates to (or done complete rewrites of) nearly every previous page. In short, this Seventh Edition represents a major revision.

The chapters are organized like so:

- Chapters 1 through 6 and 15 through 18 cover the principles of creating HTML pages and the range of HTML elements at your disposal, clearly demonstrating when and how to use each one.
- Chapters 7 through 14 dive into CSS, all the way from creating your first style rule to applying enhanced visual effects with CSS3.
- Chapter 19 shows you how to add prewritten JavaScript to your pages.
- Chapter 20 tells you how to test and debug your pages before putting them on the Web.
- Chapter 21 explains how to secure your own domain name and then publish your site on the Web for all to see.

Expanding on that, some of the topics include:

- Creating, saving, and editing HTML and CSS files.
- What it means to write semantic HTML and why it is important.
- How to separate your page's content (that is, your HTML) from its presentation (that is, your CSS)—a key aspect of progressive enhancement.
- Structuring your content in a meaningful way by using HTML elements that have been around for years and ones that are new in HTML5.
- Improving your site's accessibility with ARIA landmark roles and other good coding practices.
- Adding images to your pages and optimizing them for the Web.
- Linking from one Web page to another page, or from one part of a page to another part.
- Styling text (size, color, bold, italics, and more); adding background colors and images; and implementing a fluid, multi-column layout that can shrink and expand to accommodate different screen sizes.

- Leveraging new selectors in CSS3 that allow you to target your styles in a wider range of ways than was previously possible.
- Learning your options for addressing visitors on mobile devices.
- Building a single site for all users—whether they are using a mobile phone, tablet, laptop, desktop computer, or other Web-enabled device—based on many of the principles of responsive web design, some of which leverage CSS3 media queries.
- Adding custom Web fonts to your pages with **`@font-face`**.
- Using CSS3 effects such as opacity, background alpha transparency, gradients, rounded corners, drop shadows, shadows inside elements, text shadows, and multiple background images.
- Building forms to solicit input from your visitors, including using some of the new form input types in HTML5.
- Including media in your pages with the HTML5 **`audio`** and **`video`** elements.

And more.

These topics are complemented by many dozens of code samples that show you how to implement the features based on best practices in the industry.

What this book *won't* teach you

Alas, even after adding so many pages since the previous edition, there is so much to talk about when it comes to HTML and CSS that we had to leave out some topics.

With a couple of exceptions, we stuck to omitting items that you would have fewer occasions to use, are still subject to change, lack widespread browser support, require JavaScript knowledge, or are advanced subjects.

Some of the topics not covered include:

- The HTML5 **`details`**, **`summary`**, **`menu`**, **`command`**, and **`keygen`** elements.
- The HTML5 **`canvas`** element, which allows you to draw graphics (and even create games) with JavaScript.
- The HTML5 APIs and other advanced features that require JavaScript knowledge or are otherwise not directly related to the new semantic HTML5 elements.
- CSS sprites. This technique involves combining more than one image into a single image, which is very helpful in minimizing the number of assets your pages need to load. See www.bruceontheloose.com/sprites/ for more information.
- CSS image replacement. These techniques are often paired with CSS sprites. See www.bruceontheloose.com/ir/ for more information.
- CSS3 transforms, animations, and transitions.
- CSS3's new layout modules.

How This Book Works

Nearly every section of the book contains practical code examples that demonstrate real-world use (A and B). Typically, they are coupled with screen shots that show the results of the code when you view the Web page in a browser C.

Most of the screen shots are of the latest version of Firefox that was available at the time. However, this doesn't imply a recommendation of Firefox over any other browser. The code samples will look very similar in any of the latest versions of Chrome, Internet Explorer, Opera, or Safari. As you will learn in Chapter 20, you should test your pages in a wide range of browsers before putting them on the Web, because there's no telling what browsers your visitors will use.

The code and screen shots are accompanied by descriptions of the HTML elements or CSS properties in question, both to give the samples context and to increase your understanding of them.

In many cases, you may find that the descriptions and code samples are enough for you to start using the HTML and CSS features. But if you need explicit guidance on how to use them, step-by-step instructions are always provided.

Finally, most sections contain tips that relay additional usage information, best practices, references to related parts of the book, links to relevant resources, and more.

A You'll find a snippet of HTML code on many pages, with the pertinent sections highlighted. An ellipsis (...) represents additional code or content that was omitted for brevity. Often, the omitted portion is shown in a different code figure.

```
...
<body>
<header role="banner">
    ...
        <nav role="navigation">
            <ul class="nav">
                <li><a href="/" class="current">home</a></li>
                <li><a href="/about/">about</a></li>
                <li><a href="/resources/">resources</a></li>
                <li><a href="/archives/">archives</a></li>
            </ul>
        </nav>
    ...
</header>

...

</body>
</html>
```

B If CSS code is relevant to the example, it is shown in its own box, with the pertinent sections highlighted.

```
/* Site Navigation */
.nav li {
    float: left;
    font-size: .75em; /* makes the
    → bullets smaller */
}

.nav li a {
    font-size: 1.5em;
}

.nav li:first-child {
    list-style: none;
    padding-left: 0;
}
```

C Screen shots of one or more browsers demonstrate how the code affects the page.

Conventions used in this book

The book uses the following conventions:

- The word *HTML* is all encompassing, representing the language in general. *HTML5* is used when referring to that specific version of HTML, such as when discussing a feature that is new in HTML5 and doesn't exist in previous versions of HTML. The same approach applies to usage of the terms *CSS* (general) and *CSS3* (specific to CSS3).
- Text or code that is a placeholder for a value you would create yourself is italicized. Most placeholders appear in the step-by-step instructions. For example, "Or type **`#rrggbb`**, where ***`rrggbb`*** is the color's hexadecimal representation."
- Code that you should actually type or that represents HTML or CSS code appears in **`this font`**.
- An arrow (→) in a code figure indicates a continuation of the previous line—the line has been wrapped to fit in the book's column B. The arrow is not part of the code itself, so it's not something you would type. Instead, type the line continuously, as if it had not wrapped to another line.
- The first occurrence of a word is italicized when it is defined.
- *IE* is often used as a popular abbreviation of *Internet Explorer*. For instance, IE9 is synonymous with Internet Explorer 9.
- Whenever a plus sign (+) follows a browser version number, it means the version listed plus subsequent versions. For instance, Firefox 8+ refers to Firefox 8.0 and all versions after it.

Companion Web Site

The book's site, at www.bruceontheloose.com/htmlcss/, contains the table of contents, every complete code example featured in the book (plus some additional ones that wouldn't fit), links to resources cited in the book (as well as additional ones), information about references used during writing, a list of errata, and more.

The site also includes reference sections (Appendixes A and B) that we didn't have room to include in the book. These are handy for quickly looking up HTML elements and attributes or CSS properties and values. (They also contain some information not covered in the book.)

You can find the code examples at www.bruceontheloose.com/htmlcss/examples/. You can browse them from there or download them to your computer—all the HTML and CSS files are yours for the taking.

In some cases, I've included additional comments in the code to explain more about what it does or how to use it. A handful of the code samples in the book are truncated for space considerations, but the complete versions are on the book's Web site. Please feel free to use the code as you please, modifying it as needed for your own projects.

The URLs for some of the key pages on the book's site follow:

- Home page: www.bruceontheloose.com/htmlcss/
- Code samples: www.bruceontheloose.com/htmlcss/examples/
- Appendix A: HTML Reference: www.bruceontheloose.com/ref/html/
- Appendix B: CSS Properties and Values: www.bruceontheloose.com/ref/css/

I hope you find the site helpful.

Video Training

Visual QuickStart Guides are now even more visual: Building on the success of the top-selling Visual QuickStart Guide books, Peachpit now offers Video QuickStarts. As a companion to this book, Peachpit offers more than an hour of short, task-based videos that will help you master HTML5's top features and techniques; instead of just reading about how to use HTML5, you can watch it in action. It's a great way to learn all the basics and some of the newer or more complex features of HTML5. Log on to the Peachpit site at www.peachpit.com/register to register your book, and you'll find a free streaming sample; purchasing the rest of the material is quick and easy.

This page intentionally left blank

Text

Unless a site is heavy on videos or photo galleries, most content on Web pages is text. This chapter explains which HTML semantics are appropriate for different types of text, especially (but not solely) for text within a sentence or phrase.

For example, the **em** element is specifically designed for indicating emphasized text, and the **cite** element's purpose is to cite works of art, movies, books, and more.

Browsers typically style many text elements differently than normal text. For instance, both the **em** and **cite** elements are italicized. Another element, **code**, which is specifically designed for formatting lines of code from a script or program, displays in a monospace font by default.

How content will look is irrelevant when deciding how to mark it up. So, you shouldn't use **em** or **cite** just because you want to italicize text. That's the job of CSS.

Instead, focus on choosing HTML elements that describe the content. If by default a browser styles it as you would yourself with CSS, that's just a bonus. If not, just override the default formatting with your own CSS.

In This Chapter

Starting a New Paragraph	100
Adding Author Contact Information	102
Creating a Figure	104
Specifying Time	106
Marking Important and Emphasized Text	110
Indicating a Citation or Reference	112
Quoting Text	113
Highlighting Text	116
Explaining Abbreviations	118
Defining a Term	120
Creating Superscripts and Subscripts	121
Noting Edits and Inaccurate Text	124
Marking Up Code	128
Using Preformatted Text	130
Specifying Fine Print	132
Creating a Line Break	133
Creating Spans	134
Other Elements	136

Starting a New Paragraph

HTML does not recognize the returns or other extra white space that you enter in your text editor. To start a new paragraph in your Web page, you use the **p** element (A and B).

To begin a new paragraph:

1. Type ** **.
2. Type the contents of the new paragraph.
3. Type ** ** to end the paragraph.

A Not surprisingly, **p** is one of the most frequently used HTML elements.

```
...
<body>

<article>
    <h1>Antoni Gaudí</h1>
    <p>Many tourists are drawn to
    → Barcelona to see Antoni Gaudí's
    → incredible architecture.</p>

    <p>Barcelona celebrated the 150th
    → anniversary of Gaudí's birth in
    → 2002.</p>

    <h2>La Casa Milà</h2>
    <p>Gaudí's work was essentially useful.
    → <span lang="es">La Casa Milà</span> is
    → an apartment building and real people
    → live there.</p>

    <h2>La Sagrada Família</h2>
    <p>The complicatedly named and curiously
    → unfinished Expiatory Temple of the
    → Sacred Family is the most visited
    → building in Barcelona.</p>
</article>

</body>
</html>
```

B Here you see the typical default rendering of paragraphs. As with all content elements, you have full control over the formatting with CSS.

TIP You can use styles to format paragraphs with a particular font, size, or color (and more). For details, consult Chapter 10.

TIP To control the amount of space between lines, consult "Setting the Line Height" in Chapter 10. To control the amount of space after a paragraph, consult "Setting the Margins around an Element" or "Adding Padding around an Element," both of which are in Chapter 11.

TIP You can justify paragraph text or align it to the left, right, or center with CSS (see "Aligning Text" in Chapter 10).

Adding Author Contact Information

You might think the **address** element is for marking up a postal address, but it isn't (except for one circumstance; see the tips). In fact, there isn't an HTML element explicitly designed for that purpose.

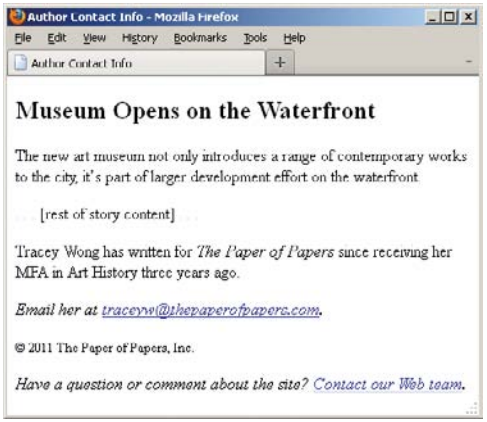
Instead, **address** defines the contact information for the author, people, or organization relevant to an HTML page (usually appearing at the end of the page, if at all) or part of a page, such as within a report or a news article (A and B).

To provide the author's contact information:

1. If you want to provide author contact information for an **article**, place the cursor within that **article**. Alternatively, place the cursor within the **body** (or, more commonly, the page-level **footer**) if you want to provide author contact information for the page at large.
2. Type **<address>**.
3. Type the author's email address, a link to a page with contact information, and so on.
4. Type **</address>**.

A This page has two **address** elements: one for the **article**'s author and the other in a page-level **footer** for the people who maintain the whole page. Note that the **address** for the **article** contains contact information only. Although the background information about Tracey Wong is also in the **article**'s **footer**, it's outside the **address** element.

```
...
<body>

<article>
    <h1>Museum Opens on the Waterfront</h1>
    <p>The new art museum not only introduces
    → a range of contemporary works to the
    → city, it's part of larger development
    → effort on the waterfront.</p>
    ... [rest of story content] ...

    <!-- the article's footer with address
    → information for the article -->
    <footer>
        <p>Tracey Wong has written for <cite>
        → The Paper of Papers</cite> since
        → receiving her MFA in art history
        three years ago.</p>
        <address>
        Email her at <a href="mailto:
        → traceyw@thepaperofpapers.com">
        → traceyw@thepaperofpapers.com
        → </a>.
        </address>
    </footer>
</article>

<!-- the page's footer with address
→ information for the whole page -->
<footer>
    <p><small>&copy; 2011 The Paper of
    → Papers, Inc.</small></p>

    <address>
    Have a question or comment about the
    → site? <a href="site-feedback.html">
    → Contact our Web team</a>.
    </address>
</footer>

</body>
</html>
```



B The `address` element renders in italics by default.

TIP **Most of the time, contact information takes the form of the author's email address or a link to a page with more contact information. The contact information could very well be the author's postal address, in which case marking it up with `address` would be valid. But if you're creating the Contact Us page for your business and want to include your office locations, it would be incorrect to code those with `address`.**

TIP **`address` pertains to the nearest `article` element ancestor, or to the page's `body` if `address` isn't nested within an `article`. It's customary to place `address` in a `footer` element when noting author contact information for the page at large A.**

TIP **An `address` in an `article` provides contact information for the author of that `article` A, not for any `articles` nested within that `article`, such as user comments.**

TIP **`address` may contain author contact information only, not anything else such as the document or `article`'s last modified date A. Additionally, HTML5 forbids nesting any of the following elements inside `address`: `h1`–`h6`, `article`, `address`, `aside`, `footer`, `header`, `hgroup`, `nav`, and `section`.**

TIP **See Chapter 3 to learn more about the `article` and `footer` elements.**

Creating a Figure

As you well know, it's a common convention in the print world to associate figures with text. A figure may be a chart, a graph, a photo, an illustration, a code segment, and so on. You've seen these at play in newspapers, magazines, reports, and more. Why, this very book has figures on most pages.

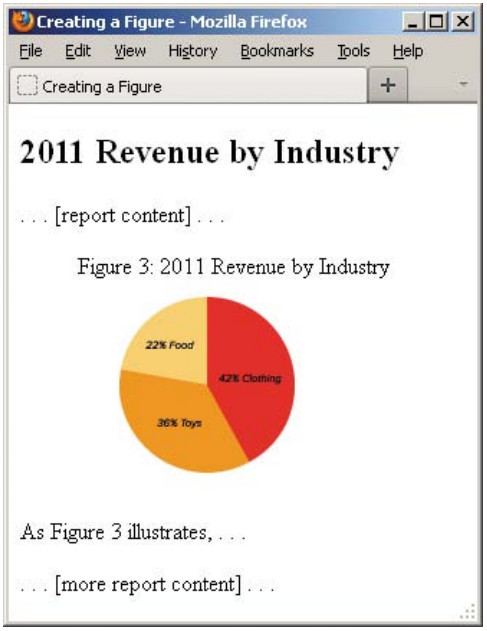
Prior to HTML5, there wasn't an element designed for this purpose, so developers cobbled together solutions on their own, often involving the less-than-ideal, non-semantic **div** element. HTML5 changes that with **figure** and **figcaption**. By definition, a **figure** is a self-contained piece of content (with an optional caption) that is referred to by the main content of your document (A and B). The optional **figcaption** is a **figure**'s caption or legend and may appear either at the beginning or at the end of a **figure**'s content A.

To create a figure and figure caption:

1. Type **<figure>**.
2. Optionally, type **<figcaption>** to begin the figure's caption.
3. Type the caption text.
4. Type **</figcaption>** if you created a caption in steps 2 and 3.
5. Create your figure by adding code for images, videos, data tables, and so on.
6. If you didn't include a **figcaption** before your **figure**'s content, optionally follow steps 2–4 to add one after the content.
7. Type **</figure>**.

A This **figure** has a chart image, though more than one image or other types of content (such as a data table or video) are allowed as well. The **figcaption** element isn't required, but it must be the first or last element in a **figure** if you do include it. A **figure** doesn't have a default styling aside from starting on its own line in modern browsers B.

```
...
<body>

<article>
    <h1>2011 Revenue by Industry</h1>
    ... [report content] ...

    <figure>
       <figcaption>Figure 3: 2011 Revenue
       → by Industry</figcaption>

       <img src="chart-revenue.png"
       → width="180" height="143" alt=
       → "Revenue chart: Clothing 42%,
       → Toys 36%, Food 22%" />
    </figure>

    <p>As Figure 3 illustrates, ... </p>

    ... [more report content] ...
</article>

</body>
</html>
```



B The `figure` with the chart and caption appears within the `article` text. It would be simple to style the `figure` with CSS so, for example, it has a border and so the article text wraps around it.

TIP **Typically, `figure` is part of the content that refers to it A, but it could also live elsewhere on the page or on another page, such as in an appendix.**

TIP **The `figure` element may include multiple pieces of content. For instance, A could include two charts: one for revenue and another for profits. Keep in mind, though, that regardless of how much content a `figure` has, only one `figcaption` is allowed.**

TIP **Don't use `figure` simply as a means to embed all instances of self-contained content within text. Oftentimes, the `aside` element may be appropriate instead (see "Specifying an Aside" in Chapter 3).**

TIP **The `figure` element is known as a sectioning root in HTML5, which means it can have h1–h6 headings (and thus, its own outline), but they don't contribute to the document outline. This is very different than sectioning *content*. Please see "Understanding HTML5's Document Outline" in Chapter 3.**

TIP **You can't use the optional `figcaption` element unless it's in a `figure` with other content.**

TIP **`figcaption` text doesn't have to begin with "Figure 3" or "Exhibit B." It could just as well be a brief description of the content, like a photo caption.**

TIP **If you include a `figcaption`, it must be either the first or last element of the `figure`.**

Specifying Time

You can mark up a precise time or calendar date with the **time** element. This element is new in HTML5. (See the sidebar "Understanding the **datetime** Format" for more specifics about the calendar date system.)

One of the most common uses of **time** is to indicate the publication date of an **article** element. To do so, include the **pubdate** attribute. The **time** element with **pubdate** represents the publication date of the nearest ancestor **article** element (A). You could also time-stamp an **article**'s reader-submitted comments with **time**, **datetime**, and **pubdate**, assuming each comment is wrapped in an **article** element that is nested in the **article** to which the comment relates (see Example 2 of the sidebar in "Creating an Article" in Chapter 3).

You can represent time with the **time** element in a variety of ways ((A) and (C)). The optional text content inside **time** (that is, **<time>text</time>**) appears on the screen as a human-readable version ((B) and (D)) of the optional, machine-readable **datetime** value.

(A) As is proper, the **datetime** attribute and the **time** element's text reflect the same date, though they can be written differently than one another (see (C) for more examples). This **time** element represents the date the article was published, because the **pubdate** attribute is included.

```
...
<body>

<article>
    <header>
        <h1>Cheetah and Gazelle Make Fast
        → Friends</h1>
        <p><time datetime="2011-10-15"
        → pubdate="pubdate">October 15,
        → 2011</time></p>
    </header>

    ... [article content] ...
</article>

</body>
</html>
```

(B) The **article**'s publication date appears underneath its heading. The text content version of the **time** element displays, not the **datetime** value.

C The `time` element can be utilized several ways. The simplest form (the first example) lacks a `datetime` attribute. But it *does* provide the date and times in the valid format as required when `datetime` is omitted. The top three examples contain time and/or date with text inside `time`, which will display on the screen D. I suggest you always include this human-readable form of the time, since, currently, browsers won't display a value otherwise B.

```
...
<body>

<p>The train arrives at <time>08:45</time>
→ and <time>16:20</time> on
→ <time>2012-04-10</time>.</p>

<p>We began our descent from the peak of
→ Everest on <time datetime="1952-06-12T11:
→ 05:00">June 12, 1952 at 11:05 a.m.
→ </time></p>

<p>They made their dinner reservation
→ for <time datetime="2011-09-20T18:
→ 30:00">tonight at 6:30</time>.</p>

<p>The record release party is on <time
→ datetime="2011-12-09"></time>.</p>

</body>
</html>
```

D The first three paragraphs show a time. The last does not (see the last tip).

To specify a precise time, calendar date, or both:

1. Type **`<time`** to begin a **`time`** element.
2. If desired, type **`datetime="`*`time`*`"`** where ***`time`*** is represented in the approved format (see the "Understanding the **`datetime`** Format" sidebar).
3. If the time represents the publication date of an **`article`** or the whole page, type either **`pubdate="pubdate"`** or **`pubdate`**.
4. Type **`>`** to complete the start tag.
5. If you want the time to display in the browser, type text that reflects the time, the date, or both (see the first tip about the allowed text format).
6. Type **`</time>`**.

TIP If you omit the `datetime` attribute, the text content must conform to the valid date or time format. In other words, the first example in C could not be coded as `<p>The train arrives at <time>8:45 a.m.</time> and <time>4:20 p.m.</time> on <time>October 4th, 2012</time>.</p>`. However, when you do include `datetime`, you're free to represent the date or time in the text content as you wish, as seen in the second and third examples of C.

TIP Don't use `time` to mark up imprecise dates or times, such as "the mid-1900s," "just after midnight," "the latter part of the Renaissance," or "early last week."

TIP Always include a text version of the time and date inside the `time` element if you want it to show in your page. If it's missing, browsers are supposed to display text that is based on `datetime`'s value, but support is lacking greatly at the time of this writing D.

continues on page 109

Understanding the `datetime` Format

The **`time`** element's time is based on a 24-hour clock with an optional time-zone offset from UTC (Coordinated Universal Time). The **`datetime`** attribute provides the date and time in a machine-readable format, which I've simplified for this initial example:

`YYYY-MM-DDThh:mm:ss`

For example (local time):

`2011-11-03T17:19:10`

This means "November 3, 2011, at 10 seconds after 5:19 p.m. local time." **`T`** separates the date (**`YYYY-MM-DD`**) and time (**`hh:mm:ss`**), and if you include a time, the seconds are optional. (You may also provide time with milliseconds in the format of **`hh:mm.sss`**. Note the period before the milliseconds.)

If you'd like, you can represent your time in a global context instead. Add a **`Z`** at the end, and the time zone is UTC.

For example (global time in UTC):

`2011-11-03T17:19:10Z`

Or, you can specify a time-zone offset from UTC by omitting **`Z`** and preceding the offset with **`–`** (minus) or **`+`** (plus).

For example (global time with offset from UTC):

`2011-11-03T17:19:10-03:30`

This means "November 3, 2011, at 10 seconds after 5:19 p.m. Newfoundland standard time (it's minus three and a half hours from UTC)." A list of time zones by UTC offsets is available at http://en.wikipedia.org/wiki/List_of_time_zones_by_UTC_offset.

If you do include **`datetime`**, it doesn't require the full complement of information I just described, as the examples in C show. Technically speaking, dates in the **`time`** element are based on the proleptic Gregorian calendar (as you may know, the Gregorian calendar is the internationally accepted civil calendar system in common use today). As such, HTML5 recommends you don't use it for pre-Gregorian dates (chances are this won't be an issue for your content, but just so you know about it). There has been a lot of discussion about this limitation, but it's a complicated topic. Read http://dev.w3.org/html5/spec-author-view/the-time-element.html for more information and examples, or www.quirksmode.org/blog/archives/2009/04/making_time_saf.html for an extensive explanation of some of the issues.

TIP If you use `time` with `pubdate` to indicate an `article`'s publication date, it's common but not mandatory to place it in either a `header` or `footer` element of the `article` element. Regardless, be sure it's nested somewhere within the relevant `article`.

TIP If a `time` element with the `pubdate` attribute doesn't have an `article` element as an ancestor, it represents the publication date and time of the whole page.

TIP You can specify `pubdate` as either

```
<time pubdate></time>
```

or `<time pubdate="pubdate"></time>`.

However, if you include it, either `datetime` or a text content version of the time is required (A).

TIP The `datetime` attribute's machine-readable format (see the "Understanding the `datetime` Format" sidebar) allows for syncing dates and times between Web applications. As of this writing, no browser displays the `datetime` value ((B) and (D)).

TIP You may not nest a `time` element inside another one.

Marking Important and Emphasized Text

The **strong** element denotes important text, while **em** conveys emphasis. You can use them individually or together as your content requires (A and B).

To mark important text:

1. Type ****.
2. Type the text that you want to mark as important.
3. Type ****.

To emphasize text:

1. Type ****.
2. Type the text that you want to emphasize.
3. Type ****.

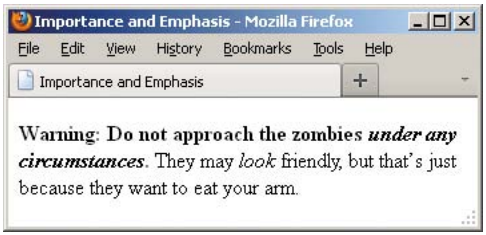
TIP **Do not use the b and i elements as replacements for strong and em, respectively. Although they may look similar in a browser, their meanings are very different (see the sidebar "The b and i Elements: Redefined in HTML5").**

TIP **You may nest strong text inside a phrase that is also marked with strong. If you do, the importance of strong text increases each time it's a child of another strong. The same is true of the level of emphasis for em text nested in another em. For example, "due by November 17th" is marked as more important semantically than the other strong text in this sentence: Remember that entries are due by November 17th. .**

TIP **You can style any text as bold or italic with CSS, as well as override the browser's default styling of elements like strong and em (B). For details, consult "Creating Italics" and "Applying Bold Formatting" in Chapter 10.**

A The first sentence has both **strong** and **em**, while the second has **em** only. If **under any circumstances** were marked up instead as **under any circumstances**, it would have greater importance than the text contained in the surrounding **strong**.

```
...
<body>

<p><strong>Warning: Do not approach the
→ zombies <em>under any circumstances</em>.
→ </strong> They may <em>look</em>
→ friendly, but that's just because they want
→ to eat your arm.</p>

</body>
</html>
```



B Browsers typically display **strong** text in boldface and **em** text in italics. If **em** is a child of a **strong** element (see the first sentence in A), its text will be both italicized and bold.

The b and i Elements: Redefined in HTML5

HTML5 focuses on semantics, not on an element's presentation. The **b** and **i** elements are holdovers from the earliest days of HTML, when they were used to make text bold or italic (CSS didn't exist yet). They rightly fell out of favor in HTML 4 and XHTML 1 because of their presentational nature. Coders were encouraged to use **strong** instead of **b**, and **em** instead of **i**. It turns out, though, that **em** and **strong** are not always semantically appropriate. HTML5 addresses this by redefining **b** and **i**.

Some typographic conventions in traditional publishing fall through the cracks of available HTML semantics. Among them are italicizing certain scientific names (for example, "The *Ulmus americana* is the Massachusetts state tree."), named vehicles (for example, the "We rode the *Orient Express*."), and foreign (to English) language phrases (for example, "The couple exhibited a *joie de vivre* that was infectious."). These terms aren't italicized for emphasis, just stylized per convention.

Rather than create several new semantic elements (and further muddy the waters) to address cases like these, HTML5 takes a practical stance by trying to make do with what is available: **em** for all levels of emphasis, **strong** for importance, and **b** and **i** for the through-the-cracks cases.

The notion is that although **b** and **i** don't carry explicit semantic meaning, the reader will recognize that a difference is implied because they differ from the surrounding text. And you're still free to change their appearance from bold and italics with CSS. HTML5 emphasizes that you use **b** and **i** only as a last resort when another element (such as **strong**, **em**, **cite**, and others) won't do.

The b Element in Brief

HTML5 redefines the **b** element this way:

> The **b** element represents a span of text to which attention is being drawn for utilitarian purposes without conveying any extra importance and with no implication of an alternate voice or mood, such as key words in a document abstract, product names in a review, actionable words in interactive text-driven software, or an article lede.

For example:

```
<p>The <b>XR-5</b>, also dubbed the <b>Extreme Robot 5</b>, is the best robot we've ever
→ tested.</p>
```

The **b** element renders as bold by default.

The i Element in Brief

HTML5 redefines the **i** element this way:

> The **i** element represents a span of text in an alternate voice or mood, or otherwise offset from the normal prose in a manner indicating a different quality of text, such as a taxonomic designation, a technical term, an idiomatic phrase from another language, a thought, or a ship name in Western texts.

Here are some examples:

```
<p>The <i lang="la">Ulmus americana</i> is the Massachusetts state tree.</p>
<p>The <i>Orient Express</i> began service in 1883.<p>
<p>The couple exhibited a <i lang="fr">joie de vivre</i> that was infectious.<p>
```

The **i** element displays in italics by default.

Indicating a Citation or Reference

Use the **cite** element for a citation or reference to a source. Examples include the title of a play, script, or book; the name of a song, movie, photo, or sculpture; a concert or musical tour; a specification; a newspaper or legal paper; and more (A and B).

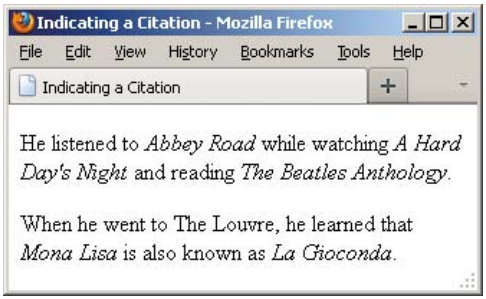
To cite a reference:

1. Type **<cite>**.
2. Type the reference's name.
3. Type **</cite>**.

TIP **For instances in which you are quoting from the cited source, use the blockquote or q elements, as appropriate, to mark up the quoted text (see "Quoting Text"). To be clear, cite is only for the source, not what you are quoting from it.**

A The **cite** element is appropriate for marking up the titles of works of art, music, movies, and books.

```
...

<p>He listened to <cite>Abbey Road</cite>
→ while watching <cite>A Hard Day's Night
→ </cite> and reading <cite>The Beatles
→ Anthology</cite>.

<p>When he went to The Louvre, he learned
→ that <cite>Mona Lisa</cite> is also known
→ as <cite lang="it">La Gioconda</cite>.</p>

...
```



B The **cite** element renders in italics by default.

HTML5 and Using the cite Element for Names

Amid a good amount of disagreement from the development community, HTML5 explicitly declares that using **cite** for a reference to a person's name is invalid, even though previous versions of HTML allowed it and many developers and designers used it that way.

The HTML 4 spec provides the following example (I've changed the element names from uppercase to lowercase):

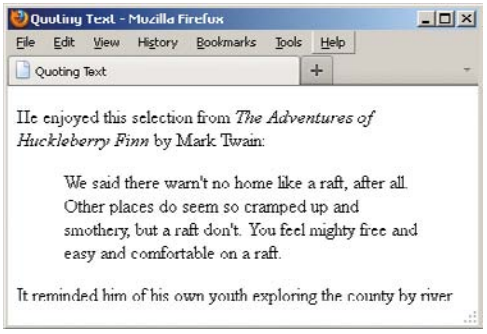
```
As <cite>Harry S. Truman</cite> said,
<q lang="en-us">The buck stops here.</q>
```

In addition to instances like that, sites have often used **cite** for the name of visitors who leave comments in blog postings and articles (the default WordPress theme does too).

Many developers have made it clear that they intend to continue to use **cite** on names associated with quotes in their HTML5 pages because HTML5 doesn't provide an alternative they deem acceptable (namely, the **span** and **b** elements). Jeremy Keith made the case vociferously in http://24ways.org/2009/incite-a-riot/.

A A **blockquote** can be as short or as long as you need. Optionally, include the **cite** attribute—not to be confused with the **cite** element shown in the first paragraph—to provide the location of the quoted text. However, browsers don't display the **cite** attribute's information B. (See the second tip for a related recommendation.)

```
...
<body>

<p>He enjoyed this selection from <cite>The
→ Adventures of Huckleberry Finn</cite> by
→ Mark Twain:</p>

<blockquote cite="http://www.marktwain
→ books.edu/the-adventures-of-huckleberry
→ -finn/">
    <p>We said there warn't no home like a
    → raft, after all. Other places do seem
    → so cramped up and smothery, but a
    → raft don't. You feel mighty free and
    → easy and comfortable on a raft.</p>
</blockquote>

<p>It reminded him of his own youth exploring
→ the county by river.</p>

</body>
</html>
```



B Browsers typically indent **blockquote** text by default. Historically, browsers haven't displayed the **cite** attribute's value (see the second tip for a related recommendation). The **cite** element, on the other hand, is supported by all browsers and typically renders in italics, as shown. All of these defaults can be overridden with CSS.

Quoting Text

There are two special elements for marking text quoted from a source. The **blockquote** element represents a quote (generally a longer one, but not necessarily) that stands alone A and renders on its own line by default B. Meanwhile, the **q** element is for short quotes, like those within a sentence C (on the next page).

Browsers are supposed to enclose **q** element text in language-specific quotation marks automatically, but Internet Explorer didn't support this until IE8. Some browsers have issues with nested quotes, too. Be sure to read the tips to learn about alternatives to using the **q** element.

To quote a block of text:

1. Type **<blockquote** to begin a block quote.
2. If desired, type **cite="*url*"**, where ***url*** is the address of the source of the quote.
3. Type **>** to complete the start tag.
4. Type the text you wish to quote, surrounding it with paragraphs and other elements as appropriate.
5. Type **</blockquote>**.

To quote a short phrase:

1. Type **<q** to begin quoting a word or phrase.
2. If desired, type **cite="*url*"**, where ***url*** is the address of the source of the quote.
3. If the quote's language is different than the page's default language (as specified by the **lang** attribute on the **html** element), type **lang="*xx*"**, where ***xx*** is the two-letter code for the language the quote will be in. This code is *supposed* to determine the type of quote marks that will be used (“” for English, «» for many European languages, and so on), though browser support for this rendering can vary.
4. Type **>** to complete the start tag.
5. Type the text that should be quoted.
6. Type **</q>**.

TIP **Although it's allowed, avoid placing text directly between the start and end `blockquote` tags. Instead, enclose it in `p` or other semantically appropriate elements within the `blockquote`.**

TIP **You can use the optional `cite` attribute on `blockquote` and `q` to provide a URL to the source you are quoting. Unfortunately, browsers traditionally haven't presented the `cite` URL to users** B**, so it's not the most useful of attributes on its own. Consequently, if you do include `cite`, I recommend you repeat the URL in a link (the `a` element) in your content, allowing visitors to access it. Less effectively, you could expose `cite`'s value via JavaScript.**

C Add the **lang** attribute to the **q** element if the quoted text is in a different language than the page's default (as specified by the **lang** attribute on the **html** element).

```
…
<body>

<p>And then she said, <q>Have you read
→ Barbara Kingsolver's <cite>High Tide in
→ Tucson</cite>? It's inspiring.</q></p>

<p>She tried again, this time in French:
→ <q lang="fr">Avez-vous lu le livre
→ <cite>High Tide in Tucson</cite> de
→ Kingsolver? C'est inspirational.</q></p>

</body>
</html>
```

D Browsers are supposed to add curly double quotes around `q` elements (and curly single quotes around nested `q` elements) automatically. As shown here, Firefox does, but not all browsers do (for example, older versions of Internet Explorer).

TIP **The `blockquote` element is known as a sectioning root in HTML5, which means it can have `h1`–`h6` headings (and thus, its own outline), but they don't contribute to the document outline. This is very different than sectioning *content*. Please see "Understanding HTML5's Document Outline" in Chapter 3.**

TIP **The `q` element is invalid for a quote that extends beyond one paragraph.**

TIP **Be sure you don't use `q` simply because you want quotation marks around a word or phrase. For instance, `<p>Every time I hear the word <q>soy</q>, I jump for joy.</p>` is improper because "soy" isn't a quote from a source.**

TIP **You can nest `blockquote` and `q` elements. For example, `<p>The short story began, <q>When she was a child, she would say, <q>Howdy, stranger!</q> to everyone she passed.</q></p>`. Nested `q` elements should automatically have the appropriate quotation marks—in English the outer quotes should be double and the inner ones should be single—but browser support varies. Since outer and inner quotations are treated differently in languages, add the `lang` attribute to `q` as needed (C and D).**

TIP **Because of cross-browser issues with `q` D, many (probably the majority of) coders choose to simply type the proper quotation marks or use character entities instead of the `q` element. In his in-depth article "Quoting and citing with `<blockquote>`, `<q>`, `<cite>`, and the `cite` attribute" at HTML5 Doctor, Oli Studholme discusses this and more, such as a series of options for styling quotation marks with the `q` element and related browser support information (http://html5doctor.com/blockquote-q-cite/).**

Highlighting Text

We've all used a highlighter pen at some point or another. Maybe it was when studying for an exam or going through a contract. Whatever the case, you used the highlighter to mark key words or phrases that were relevant to a task.

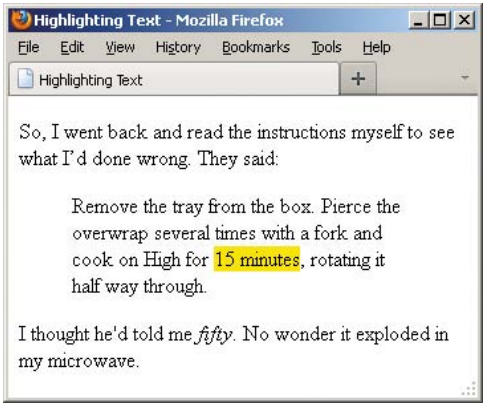
HTML5 replicates this with the new **mark** element. Think of **mark** like a semantic version of a highlighter pen. In other words, what's important is that you're noting certain words; how they appear is irrelevant. Style its text with CSS as you please (or not at all), but use **mark** only when it's pertinent to do so.

No matter when you use **mark**, it's to draw the reader's attention to a particular text segment. Here are some use cases for it:

- To highlight a search term when it appears in a search results page or an article. When people talk about **mark**, this is the most common context. Suppose you used a site's search feature to look for "solar panels." The search results or each resulting article could use **<mark>solar panels</mark>** to highlight the term throughout the text.
- To call attention to part of a quote that wasn't highlighted by the author in its original form (A and B).
- To draw attention to a code fragment (C and D).

A Although **mark** may see its most widespread use in search results, here's another valid use of it. The phrase "15 minutes" was not highlighted in the instructions on the packaging. Instead, the author of this HTML used **mark** to call out the phrase as part of the story. Default browser rendering of **mark** text varies B.

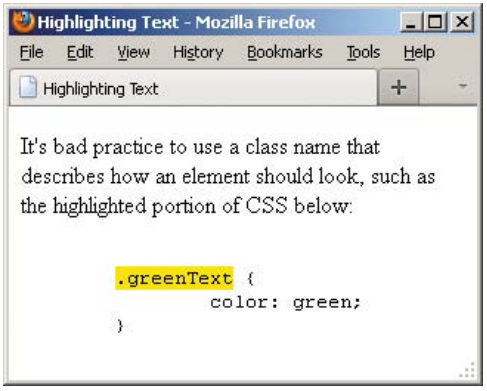
```
...
<body>

<p>So, I went back and read the instructions
→ myself to see what I'd done wrong. They
→ said:</p>

<blockquote>
    <p>Remove the tray from the box. Pierce
    → the overwrap several times with a
    → fork and cook on High for <mark>15
    → minutes</mark>, rotating it half way
    → through.</p>
</blockquote>

<p>I thought he'd told me <em>fifty</em>. No
→ wonder it exploded in my microwave.</p>

</body>
</html>
```



B Browsers with native support of the **mark** element display a yellow background behind the text by default. Older browsers don't, but you can tell them to do so with a simple rule in your style sheet (see the tips).

C This example uses **mark** to draw attention to a segment of code.

```
...
<body>

<p>It's bad practice to use a class name that
→ describes how an element should look, such
→ as the highlighted portion of CSS below:
<pre>
    <code>
    <mark>.greenText</mark> {
       color: green;
    }
    </code>
</pre>

</body>
</html>
```



D This code noted with **mark** is called out.

To highlight text:

1. Type **<mark>**.
2. Type the word or words to which you want to call attention.
3. Type **</mark>**.

TIP The mark element is not the same as either em (which represents emphasis) or strong (which represents importance). Both are covered in this chapter as well.

TIP Since mark is new in HTML5, older browsers don't render a background color by default B and D. You can instruct them to do so by adding `mark { background-color: yellow; }` to your style sheet.

TIP Be sure not to use mark simply to give text a background color or other visual treatment. If all you're looking for is a means to style text and there's no proper semantic HTML element with which to wrap it, use the span element (covered in this chapter) and style it with CSS.

Explaining Abbreviations

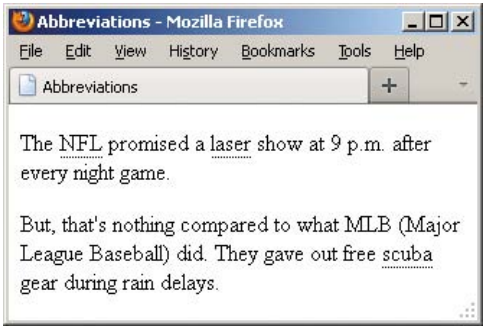
Abbreviations abound, whether as Jr., M.D., or even good ol' HTML. You can use the **abbr** element to mark up abbreviations and explain their meaning (A through C). You don't have to wrap every abbreviation in **abbr**, only when you think it would be helpful for visitors to be given the expanded meaning.

To explain abbreviations:

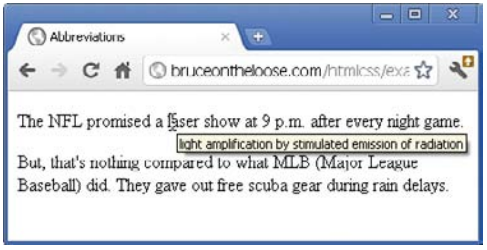
1. Type **<abbr**.
2. Optionally, next type **title="*expansion*"**, where ***expansion*** is the words that represent the abbreviation.
3. Type **>**.
4. Then type the abbreviation itself.
5. Finally, finish up with **</abbr>**.
6. Optionally, type a space and **(*expansion*)**, where ***expansion*** is the words that represent the abbreviation.

A Use the optional **title** attribute to provide the expanded version of an abbreviation. Alternatively, and arguably preferably, you could place the expansion in parentheses after the abbreviation. Or mix and match. Most people will be familiar with words like *laser* and *scuba*, so marking them up with **abbr** and providing titles isn't really necessary, but I've done it here for demonstration purposes.

```
...
<body>

<p>The <abbr title="National Football
→ League">NFL</abbr> promised a <abbr
→ title="light amplification by
→ stimulated emission of radiation">
→ laser</abbr> show at 9 p.m. after every
→ night game.</p>

<p>But, that's nothing compared to what
→ <abbr>MLB</abbr> (Major League
→ Baseball) did. They gave out free
→ <abbr title="self-contained underwater
→ breathing apparatus">scuba</abbr> gear
→ during rain delays.</p>

</body>
</html>
```



B When abbreviations have a `title` attribute, Firefox and Opera draw attention to them with a dotted underline. (You can instruct other browsers to do the same with CSS; see the tips.) In all browsers except IE6, when your visitors hover on an `abbr`, the contents of the element's `title` are shown in a tool tip.



C Chrome and some other browsers display the title of abbreviations as a tool tip, but they don't display the abbreviation itself any differently unless you apply some CSS yourself.

TIP **It's common practice to include an abbreviation's expansion (by way of a `title` or a parenthetical) only the first time it appears on a page A.**

TIP **A parenthetical abbreviation expansion is the most explicit way to describe an abbreviation, making it available to the widest set of visitors A. For instance, users on touch screen devices like smartphones and tablets may not be able to hover on an `abbr` element to see a `title` tool tip. So if you provide an expansion, consider putting it in parentheses whenever possible.**

TIP **If you use an abbreviation in its plural form, make the expansion plural as well.**

TIP **As a visual cue to sighted users, browsers like Firefox and Opera display `abbr` with a dotted bottom border if it has a `title` B. If you'd like to replicate that effect in all browsers (except IE6), add the following to your style sheet: `abbr[title] { border-bottom: 1px dotted #000; }`. Browsers provide the `title` attribute's contents as a tool tip C regardless of whether the `abbr` is styled with a border.**

TIP **If you don't see the dotted bottom border on your `abbr` in Internet Explorer 7, try adjusting the parent element's CSS `line-height` property (see Chapter 10).**

TIP **IE6 doesn't support `abbr`, so you won't see a border or a tool tip, just the text. If you really want to style `abbr` in IE6, you could put `document.createElement('abbr');` in a JavaScript file targeted for IE6 before your CSS. I say skip that and let IE6 be an outlier in this case. (See Chapter 11 to learn more about `document.createElement` as it pertains to styling elements that are new in HTML5 in IE8 and lower.)**

TIP **HTML had an `acronym` element before HTML5, but developers and designers often were confused by the difference between an abbreviation and an acronym, so HTML5 eliminated the `acronym` element in favor of `abbr` for all instances.**

Defining a Term

The **dfn** element marks the one place in your document that you define a term. Subsequent uses of the term are not marked. You wrap **dfn** only around the term you're defining, not around the definition A.

It's important where you place the **dfn** in relation to its definition. HTML5 states, "The paragraph, description list group, or section that is the nearest ancestor of the **dfn** element must also contain the definition(s) for the term given by the **dfn** element." This means that the **dfn** and its definition should be in proximity to each other. This is the case in both A and the example given in the third tip; the **dfn** and its definition are in the same paragraph.

To mark the defining instance of a term:

1. Type **<dfn>**.
2. Type the term you wish to define.
3. Type **</dfn>**.

TIP **You can also use dfn in a definition list (the dl element). See Chapter 15.**

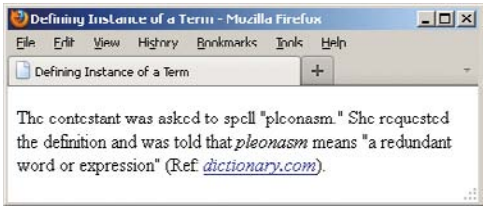
TIP **If you want to direct users to the defining instance of a term, you can add an id to the dfn and link to it from other points in the site.**

TIP **dfn may also enclose another phrasing element like abbr, when appropriate. For example, A <dfn><abbr title="Junior">Jr.</abbr></dfn> is a son with the same full name as his father. .**

TIP **HTML5 says that if you use the optional title attribute on a dfn, it should have the same value as the dfn term. As in the previous tip, if you nest a single abbr in dfn and the dfn has no text node of its own, the optional title should be on the abbr only.**

A Note that although pleonasm appears twice in the example, **dfn** marks the second one only, because that's when I defined the term (that is, it's the defining instance). Similarly, if I were to use pleonasm subsequently in the document, I wouldn't wrap it in **dfn** because I've already defined it. By default, browsers style **dfn** text differently than normal text B. Also, you don't have to use the **cite** element each time you use **dfn**, just when you reference a source.

```
...
<body>

<p>The contestant was asked to spell
→ "pleonasm." She requested the definition
→ and was told that <dfn>pleonasm</dfn>
→ means "a redundant word or expression"
→ (Ref: <cite><a href=" http://dictionary.
→ reference.com/browse/pleonasm" rel=
→ "external">dictionary.com</a></cite>).</p>

</body>
</html>
```



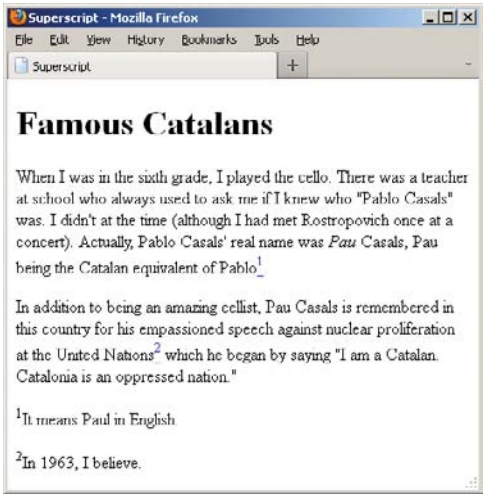
B The **dfn** element renders in italics by default in some browsers (Firefox, in this case), just like **cite**, but not in Webkit-based browsers such as Safari and Chrome. You can make them consistent by adding **dfn { font-style: italic; }** to your style sheet (see Chapters 8 and 10).

A One use of the **sup** element is to indicate footnotes. I placed the footnotes in a **footer** within the **article** rather than on the page at large because they are associated. I also linked each footnote number within the text to its footnote in the footer so visitors can access them more easily. Note, too, that the **title** attribute on the links provides another cue.

```
...
<body>

<article>
    <h1>Famous Catalans</h1>
    <p>When I was in the sixth grade, I
    → played the cello. There was a
    → teacher at school who always used
    → to ask me if I knew who "Pablo
    → Casals" was. I didn't at the time
    → (although I had met Rostropovich once
    → at a concert). Actually, Pablo Casals'
    → real name was <i>Pau</i> Casals, Pau
    → being the Catalan equivalent of Pablo
    → <a href="#footnote-1" title="Read
    → footnote"><sup>1</sup></a>.</p>

    <p>In addition to being an amazing
    → cellist, Pau Casals is remembered in
    → this country for his empassioned
    → speech against nuclear proliferation
    → at the United Nations <a href=
    → "#footnote-2" title="Read
    → footnote"><sup>2</sup></a> which
    → he began by saying "I am a Catalan.
    → Catalonia is an oppressed nation."</p>

    <footer>
        <p><sup>1</sup>It means Paul in
        → English.</p>
        <p><sup>2</sup>In 1963, I believe.</p>
    </footer>
</article>

</body>
</html>
```

Creating Superscripts and Subscripts

Letters or numbers that are raised or lowered slightly relative to the main body text are called superscripts and subscripts, respectively A. HTML includes elements for defining both kinds of text. Common uses for superscripts include marking trademark symbols, exponents, and footnotes B. Subscripts are common in chemical notation.



B Unfortunately, the **sub** and **sup** elements spoil the line spacing. Notice that there is more space between lines 4 and 5 of the first paragraph and lines 2 and 3 of the second than between the other lines. A little CSS comes to the rescue, though; see the sidebar "Fixing the Spacing between Lines when Using **sub** or **sup**" to learn how to fix this. You could also change the treatment of linked superscripts so that an underline doesn't appear so far from the superscripted text.

To create superscripts or subscripts:

1. Type **<sub>** to create a subscript or **<sup>** to create a superscript.
2. Type the characters or symbols that represent the subscript or superscript.
3. Type **</sub>** or **</sup>**, depending on what you used in step 1, to complete the element.

TIP **Most browsers automatically reduce the font size of sub- or superscripted text by a few points.**

TIP **Superscripts are the ideal way to mark up certain foreign language abbreviations like M^{lle} for *Mademoiselle* in French or 3^a for tercera in Spanish, or numerics like 2nd and 5th.**

TIP **One proper use of subscripts is for writing out chemical molecules like H_20. For example, `<p>I'm parched. Could I please have a glass of H<sub>2</sub>O?</p>`.**

TIP **Super- and subscripted characters gently spoil the even spacing between lines B. See the sidebar for a solution.**

Fixing the Spacing between Lines when Using `sub` or `sup`

The **`sub`** and **`sup`** elements tend to throw off the line height between lines of text B. Fortunately, you can set it straight with a bit of CSS.

The following code comes from Nicolas Gallagher and Jonathan Neal's excellent **`normalize.css`** (http://necolas.github.com/normalize.css/). They didn't invent the method that follows; they borrowed it from https://gist.github.com/413930 and removed the code comments. The second GitHub link includes a full explanation of what this CSS does, so I encourage you to give it a look. I also recommend checking out **`normalize.css`**, which you can use on your own projects. It helps you achieve a consistent baseline for rendering across browsers and is documented thoroughly (see "Resetting or Normalizing Default Styles" in Chapter 11).

```
/*
* Prevents sub and sup affecting line-height in all browsers
* gist.github.com/413930
*/
sub,
sup {
    font-size: 75%;
    line-height: 0;
    position: relative;
    vertical-align: baseline;
}

sup {
    top: -0.5em;
}

sub {
    bottom: -0.25em;
}
```

You may need to adjust this CSS a bit to level out the line heights, depending on your content's font size, but this should give you a very good start at the least. You'll learn about creating style sheets and how to add this CSS to your site in Chapter 8.

Noting Edits and Inaccurate Text

Sometimes you may want to indicate content edits that have occurred since the previous version of your page, or mark up text that is no longer accurate or relevant. There are two elements for noting edits: the **ins** element represents content that has been added, and the **del** element marks content that has been removed (A through D). You may use them together or individually.

Meanwhile, the **s** element notes content that is no longer accurate or relevant (it's not for edits) (E and F).

To mark an edit involving newly inserted text:

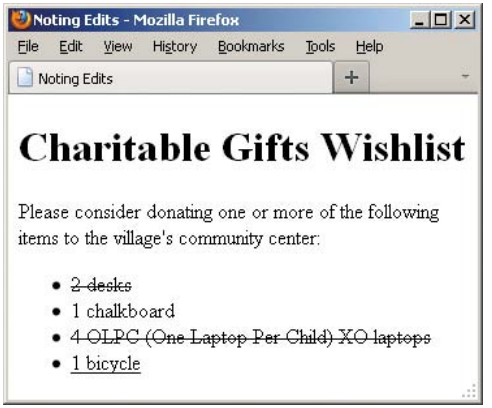
1. Type **<ins>**.
2. Type the new content.
3. Type **</ins>**.

To mark an edit involving deleted text:

1. Place the cursor before the text or element you wish to mark as deleted.
2. Type **<del>**.
3. Place the cursor after the text or element you wish to mark as deleted.
4. Type **</del>**.

A One item (the bicycle) has been added to this gift list since it was previously published, and purchased items have been removed, as noted by the **del** elements. You are not required to use **del** each time you use **ins**, or vice versa. Browsers differentiate the contents of each element visually by default B.

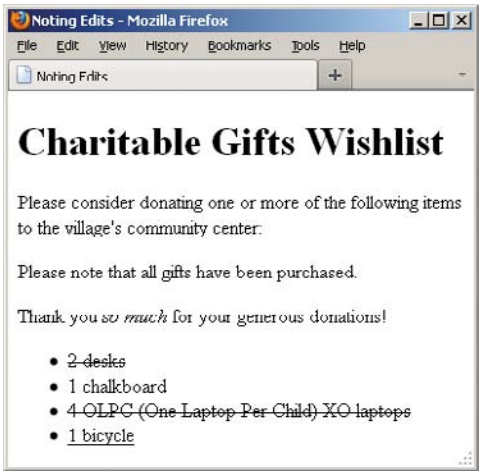
```
...
<body>

<h1>Charitable Gifts Wishlist</h1>

<p>Please consider donating one or more
→ of the following items to the village's
→ community center:</p>

<ul>
    <li><del>2 desks</del></li>
    <li>1 chalkboard</li>
    <li><del>4 <abbr>OLPC</abbr> (One
    → Laptop Per Child) XO laptops
    → </del></li>
    <li><ins>1 bicycle</ins></li>
</ul>

</body>
</html>
```



B Browsers typically display a line through deleted text and underline inserted text. You can change these treatments with CSS.

C Both **del** and **ins** are rare in that they can surround both phrasing ("inline" in pre-HTML5 parlance) content and blocks of content, as shown here. However, default browser rendering varies D.

```
...
<body>

<h1>Charitable Gifts Wishlist</h1>

<del>
    <p>Please consider donating one or more of the following items to the village's community
    → center:</p>
</del>

<ins>
    <p>Please note that all gifts have been purchased.</p>
    <p>Thank you <em>so much</em> for your generous donations!</p>
</ins>

<del>
    <ul>
        <li><del>2 desks</del></li>
        <li>1 chalkboard</li>
        <li><del>4 <abbr>OLPC</abbr> (One Laptop Per Child) XO laptops</del></li>
        <li><ins>1 bicycle</ins></li>
    </ul>
</del>

</body>
</html>
```



D Most browsers, like Chrome (left), display **del** and **ins** wrapped around blocks of content by default as expected. That is, they reflect that entire pieces of content have been deleted or inserted. As of this writing, Firefox does not; it only renders the lines for **del** and **ins** text phrases within other elements. See the "Getting **del** and **ins** to Display Consistently" sidebar to learn how to rectify this.

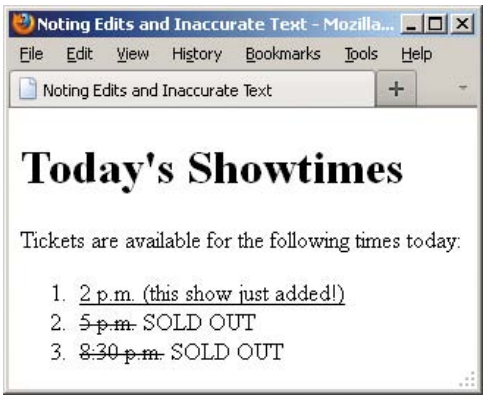
To mark text that is no longer accurate or relevant:

1. Place the cursor before the text you wish to mark as no longer accurate or relevant.
2. Type **<s>**.
3. Place the cursor after the text you wish to mark.
4. Type **</s>**.

TIP **Both `del` and `ins` support two attributes: `cite` and `datetime`. The `cite` attribute (not the same as the `cite` element) is for providing a URL to a source that explains why an edit was made. For example, `<ins cite="http://www.movienews.com/ticket-demand-high.html">2 p.m. (this show just added!)</ins>`. Use the `datetime` attribute to indicate the time of the edit. (See "Specifying Time" to learn about `datetime`'s acceptable format.) Browsers don't display the values you assign to either of these attributes, so their use isn't widespread, but feel free to include them to add context to your content. The values could be extracted with JavaScript or a program that parses through your page.**

E This example shows an ordered list (the **`ol`** element) of show times. The time slots for which ticket availability is no longer relevant have been marked with the **`s`** element. You can use **`s`** around any phrases, not just text within list items (**`li`** elements), but not around a whole paragraph or other "block-level" element like you can with **`del`** and **`ins`**.

```
...
<body>

<h1>Today's Showtimes</h1>

<p>Tickets are available for the following
→ times today:</p>

<ol>
    <li><ins>2 p.m. (this show just added!)
    → </ins></li>
    <li><s>5 p.m.</s> SOLD OUT</li>
    <li><s>8:30 p.m.</s> SOLD OUT</li>
</ol>

</body>
</html>
```



F The **`s`** element renders as a strikethrough by default in browsers.

Getting `del` and `ins` to Display Consistently

Browsers render content in a block-level **`del`** and **`ins`** inconsistently. Most display a strikethrough for **`del`** and an underline for **`ins`** on all nested content as expected, but at the least, Firefox does not D.

You can rectify this with the following explicit CSS rules (the * means that every element inside **`del`** and **`ins`** gets the treatment):

```
del * {
    text-decoration:
    →line-through;
}

ins * {
    text-decoration: underline;
}
```

Please consult Chapter 8 if you aren't sure how to add this CSS to a style sheet.

TIP **Use `del` and `ins` anytime you want to inform your visitors of your content's evolution. For instance, you'll often see them used in a Web development or design tutorial to indicate information learned since it was initially posted, while maintaining the copy as it originally stood for completeness. The same is true of blogs, news sites, and so on.**

TIP **Text marked with the `ins` element is generally underlined B. Since links are often underlined as well (if not in your site, then in many others), this may be confusing to visitors. You may want to use styles to change how inserted passages (or links) are displayed (see Chapter 10).**

TIP **Text marked with the `del` element is generally struck out B. Why not just erase it and be done with it? It depends on the context of your content. Striking it out makes it easy for sighted users to know what has changed. (Also, screen readers could announce the content as having been removed, but their support for doing so has been lacking historically.)**

TIP **Only use `del`, `ins`, and `s` for their semantic value. If you wish to underline or strike out text purely for cosmetic reasons, you can do so with CSS (see "Decorating Text" in Chapter 10).**

TIP **HTML5 notes that "The `s` element is not appropriate when indicating document edits; to mark a span of text as having been removed from a document, use the `del` element." You may find the distinction a little subtle at times. It's up to you to decide which is the appropriate semantic choice for your content.**

Marking Up Code

If your content contains code samples, file names, or program names, the **code** element is for you (A and B). To show a standalone block of code (outside of a sentence), wrap the **code** element with a **pre** element to maintain its formatting (see "Using Preformatted Text" for an example).

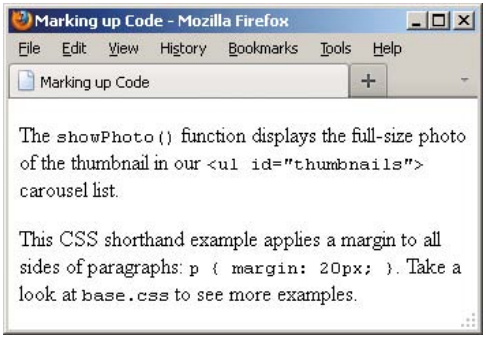
To mark up code or a file name:

1. Type **<code>**.
2. Type the code or file name.
3. Type **</code>**.

TIP **You can change the default monospaced font applied to code B with CSS (see Chapter 10).**

A The **code** element indicates that the text is code or a file name. It also renders as a monospaced font by default B. If your code needs to display **<** or **>** signs, use the **<** and **>** character entities instead, respectively. Here, the second instance of **code** demonstrates this. If you did use **<** and **>**, the browser would treat your code as an HTML element, not text.

```
...
<body>

<p>The <code>showPhoto()</code> function
→ displays the full-size photo of the
→ thumbnail in our <code>&lt;ul id=
→ "thumbnail"&gt;</code> carousel list.</p>

<p>This CSS shorthand example applies a
→ margin to all sides of paragraphs:
→ <code>p { margin: 20px; }</code>. Take
→ a look at <code>base.css</code> to see
→ more examples.</p>

</body>
</html>
```



B The **code** element's text even looks like code because of the monospaced default font.

Other Computer and Related Elements: kbd, samp, and var

The **kbd**, **samp**, and **var** elements see infrequent use, but you may have occasion to take advantage of them in your content. I'll explain each briefly.

The kbd Element

Use **kbd** to mark up user input instructions.

```
<p>To log into the demo:</p>
<ol>
    <li>Type <kbd>tryDemo</kbd> in the User Name field</li>
    <li><kbd>TAB</kbd> to the Password field and type <kbd>demoPass</kbd></li>
    <li>Hit <kbd>RETURN</kbd> or <kbd>ENTER</kbd></li>
</ol>
```

Like **code**, **kbd** renders as a monospaced font by default.

The samp Element

The **samp** element indicates sample output from a program or system.

```
<p>Once the payment went through, the site returned a message reading,
→ <samp>Thanks for your order!</samp></p>
```

samp also renders as a monospaced font by default.

The var Element

The **var** element represents a variable or placeholder value.

```
<p>Einstein is best known for <var>E</var>=<var>m</var><var>c</var><sup>2</sup>.
→ </p>
```

```
var can also be a placeholder value in content, like a Mad Libs sheet in which you'd put
→ <var>adjective</var>, <var>verb</var>, and so on.
```

var renders in italics by default.

Note that you can use the **math** and other **MathML** elements in your HTML5 pages for advanced math-related markup. See http://dev.w3.org/html5/spec-author-view/mathml.html for more information.

Using Preformatted Text

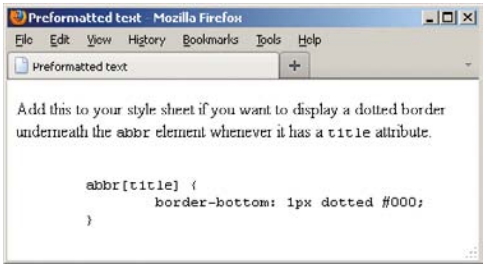
Usually, browsers collapse all extra returns and spaces and automatically break lines according to the size of the window. Preformatted text lets you maintain and display the original line breaks and spacing that you've inserted in the text. It is ideal for computer code examples A, though you can also use it for text (hello, ASCII art!).

To use preformatted text:

1. Type **<pre>**.
2. Type or copy the text that you wish to display as is, with all the necessary spaces, returns, and line breaks. Unless it is code, do not mark up the text with any HTML, such as **p** elements.
3. Type **</pre>**.

A The **pre** element is ideal for text that contains important spaces and line breaks, like the bit of CSS code shown here. Note, too, the use of the **code** element to mark up pieces of code or code-related text outside of **pre** (see "Marking Up Code" for more details).

```
…
<body>

<p>Add this to your style sheet if you want
→ to display a dotted border underneath the
→ <code>abbr</code> element whenever it has
→ a <code>title</code> attribute.</p>

<pre>
    <code>
    abbr[title] {
       border-bottom: 1px dotted #000;
    }
    </code>
</pre>

</body>
</html>
```



B Notice that the indentation and line breaks are maintained in the **pre** content.

Presentation Considerations with `pre`

Be aware that browsers typically disable automatic word wrapping of content inside a **`pre`**, so if it's too wide, it might affect your layout or force a horizontal scrollbar. The following CSS rule enables wrapping within **`pre`** in many browsers, but not in Internet Explorer 7 and below.

```
pre {
    white-space: pre-wrap;
}
```

On a related note, in most cases I don't recommend you use the **`white-space: pre;`** CSS declaration on an element such as **`div`** as a substitute for **`pre`**, because the whitespace can be crucial to the semantics of the enclosed content, especially code, and only **`pre`** always preserves it. (Also, if the user has disabled CSS in his or her browser, the formatting will be lost.)

Please see CSS coverage beginning in Chapter 7. Text formatting, in particular, is discussed in Chapter 10.

TIP **Preformatted text is typically displayed with a monospaced font like Courier or Courier New B. You can use CSS to change the font, if you like (see Chapter 10).**

TIP **If what you want to display—such as a code sample in a tutorial—contains HTML elements, you'll have to substitute each < and > around the element name with their appropriate character entities, `<` and `>` respectively (see "Marking Up Code" for an example). Otherwise the browser may try to display those elements. Be sure to validate your pages to see if you've nested HTML elements in `pre` when you shouldn't have (see "Validating Your Code" in Chapter 20).**

TIP **The `pre` element isn't a shortcut for avoiding marking up your content with proper semantics and styling its presentation with CSS. For instance, if you want to post a news article you wrote in a word processor, don't simply copy and paste it into a `pre` because you like the spacing the way it is. Instead, wrap your content in `p` (and other relevant text elements) and write CSS to control the layout as desired.**

TIP **`pre`, like a paragraph, always displays on a new line by default B.**

Specifying Fine Print

According to HTML5, **small** represents side comments such as fine print, which "typically features disclaimers, caveats, legal restrictions, or copyrights. Small print is also sometimes used for attribution or for satisfying licensing requirements." **small** is intended for brief portions of inline text, not for text spanning multiple paragraphs or other elements (A and B).

To specify fine print:

1. Type **<small>**.
2. Type the text that represents a legal disclaimer, note, attribution, and so on.
3. Type **</small>**.

TIP Be sure to use small only because it fits your content, not because you want to reduce the text size, as happens in some browsers B. You can always adjust the size with CSS (even making it larger if you'd like). See "Setting the Font Size" in Chapter 10 for more information.

TIP small is a common choice for marking up your page's copyright notice (A and B). It's meant for short phrases like that, so don't wrap it around long legal notices, such as your Terms of Use or Privacy Policy pages. Those should be marked up with paragraphs and other semantics, as necessary.

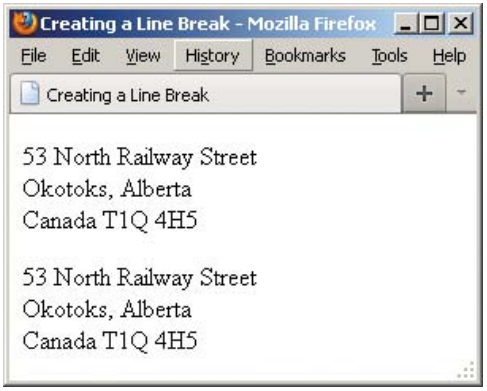
A The **small** element denotes brief legal notices in both instances shown. The second one is a copyright notice contained in a page-level **footer**, a common convention.

```
…
<body>

<p>Order now to receive free shipping.
→ <small>(Some restrictions may apply.)
→ </small></p>

…

<footer>
    <p><small>&copy; 2011 The Super
    → Store. All Rights Reserved.
    → </small></p>
</footer>

</body>
</html>
```

B The **small** element may render smaller than normal text in some browsers, but the visual size is immaterial to whether you should mark up your content with it.

A The same address appears twice, but I coded them a little differently for demonstration purposes. Remember that the returns in your code are always ignored, so both paragraphs display the same way B. Also, you can code **br** as either **
** or **
** in HTML5.

```
...
<body>

<p>53 North Railway Street<br />
Okotoks, Alberta<br />
Canada T1Q 4H5</p>

<p>53 North Railway Street <br />Okotoks,
→ Alberta <br />Canada T1Q 4H5</p>

</body>
</html>
```



B Each **br** element forces the subsequent content to a new line.

Creating a Line Break

Browsers automatically wrap text according to the width of the block or window that contains content. It's best to let content flow like this in most cases, but sometimes you'll want to force a line break manually. You achieve this with the **br** element.

To be sure, using **br** is a last resort tactic because it mixes presentation with your HTML instead of leaving all display control to your CSS. For instance, never use **br** to simulate spacing between paragraphs. Instead, mark up the two paragraphs with **p** elements and define the spacing between the two with the CSS **margin** property.

So, when might **br** be OK? Well, the **br** element is suitable for creating line breaks in poems, in a street address (A and B), and occasionally in other short lines of text that should appear one after another.

To insert a line break:

Type **
** (or **
) where the line break should occur. There is no separate end **br tag because it's what's known as an *empty* (or *void*) *element*; it lacks content.

**TIP Typing br as either
 or
 is perfectly valid in HTML5.**

TIP Styles can help you control the space between lines in a paragraph (see "Setting the Line Height" in Chapter 10) and between the paragraphs themselves (see "Setting the Margins around an Element" in Chapter 11).

TIP The hCard microformat (http://microformats.org/wiki/hcard) is "for representing people, companies, organizations, and places" in a semantic manner that's human- and machine-readable. You could use it to represent a street address instead of the provided example A.

Creating Spans

The **span** element, like **div**, has absolutely no semantic meaning. The difference is that **span** is appropriate around a word or phrase only, whereas **div** is for blocks of content (see "Creating Generic Containers" in Chapter 3).

span is useful when you want to apply any of the following to a snippet of content for which HTML doesn't provide an appropriate semantic element:

- Attributes, like **class**, **dir**, **id**, **lang**, **title**, and more (A and B)
- Styling with CSS
- Behavior with JavaScript

Because **span** has no semantic meaning, use it as a last resort when no other element will do.

A In this case, I want to specify the language of a portion of text, but there isn't an HTML element whose semantics are a fit for "La Casa Milà" in the context of a sentence. The **h1** that contains "La Casa Milà" before the paragraph is appropriate semantically because the text is the heading for the content that follows. So for the heading, I simply added the **lang** attribute to the **h1** rather than wrap a **span** around the heading text unnecessarily for that purpose.

```
...
<body>

<h1 lang="es">La Casa Milà</h1>

<p>Gaudí's work was essentially useful.
→ <span lang="es">La Casa Milà</span> is
→ an apartment building and <em>real people
→ </em> live there.</p>

</body>
</html>
```

B The **span** element has no default styling.

To add spans:

1. Type **<span**.
2. If desired, type **id="*name*"**, where ***name*** uniquely identifies the spanned content.
3. If desired, type **class="*name*"**, where ***name*** is the name of the class that the spanned content belongs to.
4. If desired, type other attributes (such as **dir**, **lang**, or **title**) and their values.
5. Type **>** to complete the start **span** tag.
6. Create the content you wish to contain in the **span**.
7. Type ****.

TIP A span doesn't have default formatting B, but just as with other HTML elements, you can apply your own with CSS (see Chapters 10 and 11).

TIP You may apply both a `class` and `id` attribute to the same `span` element, although it's more common to apply one or the other, if at all. The principal difference is that `class` is for a group of elements, whereas `id` is for identifying individual, unique elements on a page.

TIP Microformats often use `span` to attach semantic class names to content as a way of filling the gaps where HTML doesn't provide a suitable semantic element. You can learn more about them at http://microformats.org.

Other Elements

This section covers other elements that you can include within your text, but which typically have fewer occasions to be used or have limited browser support (or both).

The u element

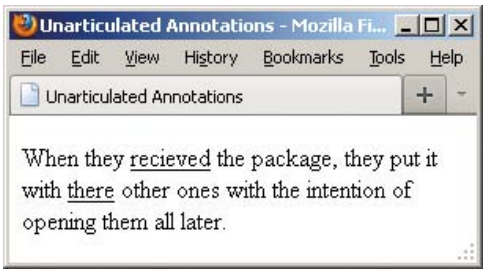
Like **b**, **i**, **s**, and **small**, the **u** element has been redefined in HTML5 to disassociate it from its past as a non-semantic, presentational element. In those days, the **u** element was for underlining text. Now, it's for unarticulated annotations. HTML5 defines it thus:

> The **u** element represents a span of text with an unarticulated, though explicitly rendered, non-textual annotation, such as labeling the text as being a proper name in Chinese text (a Chinese proper name mark), or labeling the text as being misspelt.

Here is an example of how you could use **u** to note misspelled words:

```
<p>When they <u class="spelling">
→recieved</u> the package, they put
→it with <u class="spelling">there
→</u> other ones with the intention
→of opening them all later.</p>
```

The **class** is entirely optional, and its value (which can be whatever you'd like) doesn't render with the content to explicitly indicate a spelling error. But, you could use it to style misspelled words differently (though **u** still renders as underlined text by default). Or, you could add a **title** attribute with a note such as "[sic]," a convention in some languages to indicate a misspelling.



A Like links, **u** elements are underlined by default, which can cause confusion unless you change one or both with CSS.

Use **u** only when an element like **cite**, **em**, or **mark** doesn't fit your desired semantics. Also, it's best to change its styling if **u** text will be confused with linked text, which is also underlined by default A.

The wbr element

HTML5 introduces a cousin of **br** called **wbr**. It represents "a line break opportunity." Use it in between words or letters in a long, unbroken phrase (or, say, a URL) to indicate where it could wrap if necessary to fit the text in the available space in a readable fashion. So, unlike **br**, **wbr** doesn't force a wrap, it just lets the browser know where it *can* force a line break if needed.

Here are a couple of examples:

```
<p>They liked to say,
"FriendlyFleasandFireFlies<wbr />
→FriendlyFleasandFireFlies<wbr />
→FriendlyFleasandFireFlies<wbr />
→" as fast as they could over and
→over.</p>

<p>His favorite site is this<wbr
/>is<wbr />a<wbr />really<wbr
/>really<wbr />longurl.com.</p>
```

You can type **wbr** as either **<wbr />** or **<wbr>**. As you might have guessed, you won't find many occasions to use **wbr**. Additionally, browser support is inconsistent as of this writing. Although **wbr** works in current versions of Chrome and Firefox, Internet Explorer and Opera simply ignore it.

The `ruby`, `rp`, and `rt` elements

A *ruby annotation* is a convention in East Asian languages, such as Chinese and Japanese, typically used to show the pronunciation of lesser-known characters. These small annotative characters appear either above or to the right of the characters they annotate. They are often called simply *ruby* or *rubi*, and the Japanese ruby characters are known as *furigana*.

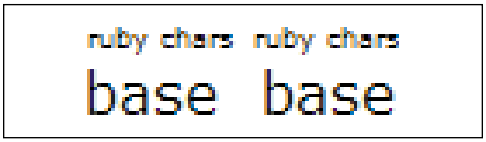
The **`ruby`** element, as well as its **`rt`** and **`rp`** child elements, is HTML5's mechanism for adding them to your content. **`rt`** specifies the ruby characters that annotate the base characters. The optional **`rp`** element allows you to display parentheses around the ruby text in browsers that don't support **`ruby`**.

The following example demonstrates this structure with English placeholder copy to help you understand the arrangement of information both in the code and in a supporting browser B. The area for ruby text is highlighted:

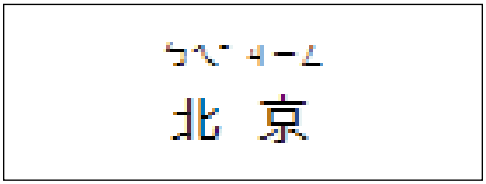
```
<ruby>
    base <rp>(</rp><rt>ruby chars
    → </rt><rp>)</rp>
    base <rp>(</rp><rt>ruby chars
    → </rt><rp>)</rp>
</ruby>
```

Now, a real-world example with the two Chinese base characters for "Beijing," and their accompanying ruby characters C:

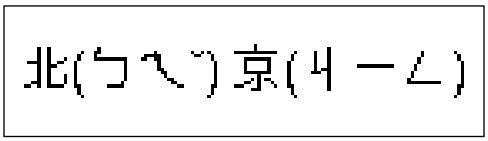
```
<ruby>
    北 <rp>(</rp><rt>ㄅㄟˇ</rt><rp>)
    → </rp>
    京 <rp>(</rp><rt>ㄐㄧㄥ</rt><rp>)
    → </rp>
</ruby>
```



B A supporting browser will display the ruby text above the base (or possibly on the side), without parentheses.



C Now, the ruby markup for "Beijing" as seen in a supporting browser.



D A browser that supports ruby ignores the `rp` parentheses and just presents the `rt` content B and C. However, a browser that doesn't support ruby displays the `rt` content in parentheses, as seen here.

You can see how important the parentheses are for browsers that don't support `ruby` D. Without them, the base and ruby text would run together, clouding the message.

TIP At the time of this writing, only Safari 5+, Chrome 11+, and all versions of Internet Explorer have basic `ruby` support (all the more reason to use `rp` in your markup). The HTML Ruby Firefox add-on (https://addons.mozilla.org/en-US/firefox/addon/6812) provides support for Firefox in the meantime.

TIP You can learn more about ruby characters at http://en.wikipedia.org/wiki/Ruby_character.

The `bdi` and `bdo` elements

If your HTML pages ever mix left-to-right characters (like Latin characters in most languages) and right-to-left characters (like characters in Arabic or Hebrew), the **`bdi`** and **`bdo`** elements may be of interest.

But, first, a little backstory. The base directionality of your content defaults to left-to-right unless you set the **`dir`** attribute on the **`html`** element to **`rtl`**. For instance, **`<html dir="rtl" lang="he">`** specifies the base directionality of your content is right-to-left and the base language is Hebrew.

Just as I've done with **`lang`** in several examples throughout the book, you may also set **`dir`** on elements within the page when the content deviates from the page's base setting. So, if the base were set to English (**`<html lang="en">`**) and you wanted to include a paragraph in Hebrew, you'd mark it up as **`<p dir="rtl" lang="he">...</p>`**.

With those settings in place, the content will display in the desired directionality most of the time; Unicode's bidirectional ("bidi") algorithm takes care of figuring it out.

The **bdo** ("bidirectional override") element is for those occasions when the algorithm *doesn't* display the content as intended and you need to override it. Typically, that's the case when the content in the HTML source is in visual order instead of logical order.

Visual order is just what it sounds like—the HTML source code content is in the same order in which you want it displayed. *Logical order* is the opposite for a right-to-left language like Hebrew; the first character going right to left is typed first, then the second character (in other words, the one to the left of it), and so on.

In line with best practices, Unicode expects bidirectional text in logical order. So, if it's visual instead, the algorithm will still reverse the characters, displaying them opposite of what is intended. If you aren't able to change the text in the HTML source to logical order (for instance, maybe it's coming from a database or a feed), your only recourse is to wrap it in a **bdo**.

To use **bdo**, you must include the **dir** attribute and set it to either **ltr** (left-to-right) or **rtl** (right-to-left) to specify the direction you want. Continuing our earlier example of a Hebrew paragraph within an otherwise English page, you would type, `<p lang="he"><bdo dir="rtl">...</bdo></p>`. **bdo** is appropriate for phrases or sentences within a paragraph. You wouldn't wrap it around several paragraphs.

The **bdi** element, new in HTML5, is for cases when the content's directionality is unknown. You don't have to include the **dir** attribute because it's set to auto by default. HTML5 provides the following example, which I've modified slightly:

> This element is especially useful when embedding user-generated content with an unknown directionality.
>
> In this example, usernames are shown along with the number of posts that the user has submitted. If the **bdi** element were not used, the username of the Arabic user would end up confusing the text (the bidirectional algorithm would put the colon and the number "3" next to the word "User" rather than next to the word "posts").

```
<ul>
    <li>User <bdi>jcranmer</bdi>:
    → 12 posts.</li>
    <li>User <bdi>hober</bdi>:
    → 5 posts.</li>
    <li>User <bdi>ابنايا</bdi>:
    → 3 posts.</li>
</ul>
```

TIP If you want to learn more on the subject of incorporating right-to-left languages, I recommend reading the W3C's article "Creating HTML Pages in Arabic, Hebrew, and Other Right-to-Left Scripts" (www.w3.org/International/tutorials/bidi-xhtml/).

The `meter` element

The **`meter`** element is another that is new thanks to HTML5. You can use it to indicate a fractional value or a measurement within a known range. Or, in plain language, it's the type of gauge you use for the likes of voting results (for example, "30% Smith, 37% Garcia, 33% Clark"), the number of tickets sold (for example, "811 out of 850"), numerical test grades, and disk usage.

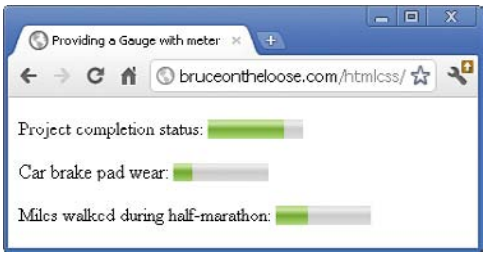
HTML5 suggests browsers could render a **`meter`** not unlike a thermometer on its side—a horizontal bar with the measured value colored differently than the maximum value (unless they're the same, of course). Chrome, one of the few browsers that supports **`meter`** so far, does just that E. For non-supporting browsers, you can style **`meter`** to some extent with CSS or enhance it further with JavaScript.

Although it's not required, it's best to include text inside **`meter`** that reflects the current measurement for non-supporting browsers to display F.

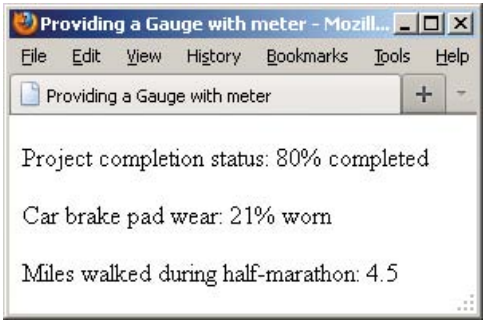
Here are some **`meter`** examples (as seen in E and F):

```
<p>Project completion status: <meter
→ value="0.80">80% completed</meter>
→ </p>

<p>Car brake pad wear: <meter low=
→ "0.25" high="0.75" optimum="0"
→ value="0.21">21% worn</meter></p>

<p>Miles walked during half-marathon:
→ <meter min="0" max="13.1" value="4.5"
→ title="Miles">4.5</meter></p>
```

`meter` doesn't have defined units of measure, but you can use the **`title`** attribute to specify text of your choosing, as in the last example. Chrome displays it as a tooltip E.



E A browser like Chrome that supports **`meter`** displays the gauge automatically, coloring it in based on the attribute values. It doesn't display the text in between **`<meter>`** and **`</meter>`**.



F Most browsers, like Firefox, don't support **`meter`**, so instead of a colored bar, they display the text content inside the **`meter`** element. You can change the look with CSS.

TIP `meter` supports several attributes. The `value` attribute is the only one that's required. `min` and `max` default to 0 and 1.0, respectively, if omitted. The `low`, `high`, and `optimum` attributes work together to split the range into low, medium, and high segments. `optimum` indicates the optimum position within the range, such as "0 brake pad wear" in one of the examples. Set `optimum` in between if neither a low nor a high value is optimal.

TIP At the time of this writing, `meter` is supported only by Chrome 11+ and Opera 11+. This partially explains why you don't yet see it in the wild too much. Feel free to use it, but just understand that most browsers will render the `meter` text rather than the visual gauge by default F.

TIP The style of the gauge that each supporting browser displays may vary.

TIP Some people have experimented with styling `meter` CSS for both supporting and non-supporting browsers. Search online for "style HTML5 meter with CSS" to see some of the results (note that some use JavaScript).

TIP `meter` is not for marking up general measurements, such as height, weight, distance, or circumference, that have no known range. For example, you cannot do this: `<p>I walked <meter value="4.5">4.5</meter> miles yesterday.</p>`.

TIP Be sure not to mix up your uses of the `meter` and `progress` elements.

The `progress` element

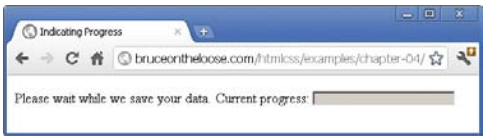
The **`progress`** element is yet another of the new elements in HTML5. You use it for a progress bar, like the kind you might see in a Web application to indicate progress while it is saving or loading a large amount of data.

As with **`meter`**, supporting browsers automatically display a progress bar based on the values of the attributes G. And again like **`meter`**, it's usually best to include text (for example, "0% saved," as shown in the example) inside **`progress`** to reflect the current progress for older browsers to display H, even though it's not required.

Here's an example:

```
<p>Please wait while we save your
→ data. Current progress: <progress
→ max="100" value="0">0% saved
→ </progress></p>
```

A full discussion of **`progress`** is beyond the scope of this book since typically you would dynamically update both the **`value`** attribute and the inner text with JavaScript as the task progresses (for example, to indicate that it's 37% completed). The visual results are the same whether you do that with JavaScript or hard-code it in the HTML, that is, **`<progress max="100" value="37">37% saved</progress>`** I. Of course, non-supporting browsers would display it similarly to H.



G A browser like Chrome that supports **`progress`** displays the progress bar automatically, coloring it in based on the value. It doesn't display the text in between **`<progress>`** and **`</progress>`**. The **`value`** attribute is set to **`0`** in this example, so the whole bar is the same color.

H Firefox doesn't support **`progress`**, so instead of a colored bar, it displays the text content inside the element. You can change the look with CSS.

I The **`progress`** bar in Chrome when the **`value`** attribute is set to **`37`** programmatically with JavaScript (or directly in the HTML), assuming **`max="100"`**.

TIP The `progress` element supports three attributes, all of which are optional: `max`, `value`, and `form`. The `max` attribute specifies the total amount of work for the task and must be greater than 0. `value` is the amount completed relative to the task. Assign the `form` attribute to the `id` of a `form` element on the page if you want to associate the `progress` element with a `form` it isn't nested within.

TIP Here's a small taste of how to modify a `progress` element with JavaScript. Let's assume the bar had been coded with an `id` of your choosing, like this:

```
<progress max="100" value="0"
→ id="progressBar">0% saved</progress>
```

JavaScript such as the following would give you access to the element:

```
var bar = document.getElementById
→ ('progressBar');
```

Then you could get or set the value via `bar.value` as needed. For example, `bar.value = 37;` would set it.

TIP The `progress` element has pretty solid support among modern browsers as of this writing: Chrome 11+, Firefox 6+, Internet Explorer 10 (available only as a Platform Preview at the time of this writing), and Opera 11+. Safari doesn't support it.

TIP The style of the `progress` bar that each supporting browser displays may vary, though you can style it yourself to some extent with CSS.

This page intentionally left blank

Index

/*, ***/**, using for CSS comments, 182
: (colon) versus = (equals) sign, 205
; (semicolon), using with CSS properties, 205
<!--, **-->**, using for HTML comments, 97
3D, positioning elements in, 318–319
320 and Up, 351

A

a element. *See* anchors
AAC audio file format, 468
abbr element, 118–119
abbreviations, explaining, 118–119
absolute versus relative URLs, 21–23
accessibility. *See also* ARIA (Accessible Rich Internet Applications), and screen readers
  advocates, 91
  explained, 11
  HTML5 media, 467
active links, 230
:active pseudo-class, 231
address element
  defining contact information with, 102–103
  using with **article** element, 70
Adobe Fireworks, 153, 155
Adobe Photoshop, 153–155
  finding image sizes, 159
  mockups, 359
  scaling images, 161
::after pseudo-element, 229
aligning
  elements vertically, 322
  text, 268–269
alt attribute, 157
alternate text, 157
anchors. *See also* links
  creating, 172–173, 175–177
  linking to, 174
animated images, saving, 151
Apple's Link Maker, 177
ARIA (Accessible Rich Internet Applications), 88–91. *See also* accessibility
  form role, 91
  landmark roles, 88–89
  role="banner" definition, 89
  role="complementary" definition, 90
  role="contentinfo" definition, 90
  role="main" definition, 89
  role="navigation" definition, 89
  screen reader test results, 90
  spec, 91
ARIA landmark roles. *See* landmark roles
  versus **id**s, 284–285
  overlap with HTML5 elements, 88
  recommendation, 90
  styling elements with, 284–285
article element, 68–71
  address element, 70
  children of, 15

article element *(continued)*
content in, 9
examples, 70–71
footer element, 70
nesting, 69
nesting content in, 9
providing contact information, 102–103
versus **section** element, 69, 73, 283
ASCII characters, 16
aside element, 75–79
examples, 78–79
versus **figure** element, 77
restrictions, 77
as sidebar, 76–77
assistive technologies, 54. *See also* screen readers
Ateş, Faruk, 377
attribute selectors, 232–235. *See also* selectors
attribute values, enclosing in quotes, 510
attributes
contents of, 14
numeric values, 15
values, 14–15
audio
Flash and hyperlink fallback, 478–479
Flash fallbacks, 476–477
hyperlink fallbacks, 475
audio attributes
autoplay, 471
controls, 471
loop, 471
muted, 471
preload, 471, 473
src, 471
audio controls, 470
audio element, 471
audio file formats
AAC, 468
MP3, 468
MP4, 468
Ogg Vorbis, 468
type attribute, 474
WAV, 468
audio files
adding to Web pages, 469–470
preload attribute, 471, 473
audio in loop
autoplay attribute, 472
controls attribute, 472
audio sources, multiple, 474–475
author contact information, adding, 102–103
autoplay
audio attribute, 471–472
video attribute, 454–457

B

b element, 111
redefinition of, 111
versus **strong** element, 110
background color
choosing, 297
fallback, 391
background images
controlling attachment of, 295
multiple, 388
repeating, 294
specifying position of, 295
using, 294
background properties, 262–263, 296–297
background-attachment property, 295
background-color property, 296, 388–389
background-image property, 294, 388–389
background-position property, 295, 297, 388–389
background-repeat property, 294, 388–389
backgrounds. *See also* gradient backgrounds
applying, 388–389
changing, 294–297
changing for text, 260–263
creating, 297
multiple, 388
BART's site, 329
BBEdit text editor, downloading, 29
bdi element, 139–141
logical order, 140
visual order, 140
bdo element, 139–141
logical order, 140
visual order, 140
Beaird, Jason, 27
::before pseudo-element, 229
Bester, Michael, 196
block-level elements, 7

block-level links, 168–170
blockquote element, 113–115
  nesting with **q** elements, 115
  as sectioning root, 115
body element, 44
bold formatting
  applying, 248–249
  removing, 248
BOM, Web resource for, 32
border properties, setting, 312
border style, defining, 311
border-image property, 313
border-radius property, 376, 378–381
borders
  adding to images, 156
  colors, 311
  deleting from images, 156
  setting, 311–313
  shortcuts, 312–313
  widths, 311
Boston Globe, 332
box model, 292–293
box-shadow property, 384–386
br element
  coding, 133
  using with **white-space** property, 267
browser developer tools
  Chrome Developer Tools, 507
  Firebug for Firefox, 507
  Internet Explorer, 507
  Opera Dragonfly, 507
  Safari Web Inspector, 507
browsers
  capabilities, 518
  compatibility, 375
  considering in layouts, 276–277
  default style sheets, 7
  finding image sizes, 158
  Graded Browser Support, 518
  styling HTML5 elements in, 286–289
  support for, 448
  testing, 518
  viewing Web pages in, 37–38

C

calendar date, specifying, 107
Camen, Kroc, 466
canvas element, using with video, 485
capitalize value, using with **text-transform**, 270
captions, creating for figures, 104–105
Cascading Style Sheets (CSS). *See* CSS (Cascading Style Sheets)
character encoding, specifying, 16
characters, reading right-to-left, 139–141
@charset declaration, using with style sheets, 199
checkboxes, creating, 440
Chrome
  Developer Tools, 507
  verifying sites in, 518
circles, creating with **border-radius**, 380
citations, indicating, 112
cite element, 112
Clark, Keith, 351
Clarke, Andy, 351
class attribute. *See also* pseudo-classes
  assigning to elements, 92–94
  versus **id** attributes, 94
  versus **id** selectors, 220
  naming, 94
  selecting elements by, 218–220
  using with microformats, 92–94
clear property, using with **float**s, 308–310
client side vs. server side, 421
Coda text editor, 29
code
  indenting, 5
  marking up, 128
  validating, 508, 514–515
code element, 128
codecs, 452
colon (:) versus equals (=) sign, 205
color, setting, 258–259. *See also* CSS colors
ColorZilla's gradient generator, 393
comments, 282. *See also* HTML comments
  adding to style rules, 182–183
  conditional, 351
compressing files, 177
conditional comments, 351
contact information, adding, 102–103
containers
  creating, 84–87
  wrapping around content, 84–85

content. *See also* text content
separating from presentation, 276
syndicated, 57
controls
audio attribute, 471–472
video attribute, 454–456
corners
elliptical, 380
rounding, 378–381
Coyier, Chris, 322
Creative Commons licenses, 152
CSS (Cascading Style Sheets)
adjacent sibling combinator, 226
colliding rules, 184–187
comments for style rules, 182–183
style rules, 181
CSS code, viewing, 212
CSS colors, 190. *See also* color
HSL, 193–196
HSLA, 193–196
RGB, 191
RGBA, 192–196
CSS errors, checking, 515
CSS properties
bare numbers, 189
hexadecimal colors, 191
inherit value, 188
lengths, 188–189
percentages, 188–189
predefined values, 188
URLs, 190
using ; (semicolon) with, 205
CSS reset, beginning main style sheet with, 290
CSS Tricks, 395
CSS troubleshooting
browser support, 513
curly braces, 512
declarations, 512
developer tools, 513
linking HTML documents, 513
property/value pairs, 512
punctuation, 513
separating properties from values, 512
spaces, 512–513
</style> end tag, 513
values, 512
CSS1, introduction of, 8
CSS3
backgrounds, 388–389
browser compatibility, 375
drop shadows, 382–387
general sibling combinator, 226
gradient backgrounds, 390–393
opacity elements, 394–395
polyfills for progressive enhancement, 376–377
rounding corners of elements, 378–381
vendor prefixes, 373–374
CSS3 Generator, 374
CSS3 selectors, resource for, 239
cursor property
auto value, 323
crosshair value, 323
default value, 323
move value, 323
pointer value, 323
progress value, 323
text value, 323
wait value, 323
***x*-resize** value, 323
cursors, changing, 323
custom markers. *See also* markers
displaying, 405
URLs (Uniform Resource Locators), 405
using, 404–405
CyberDuck FTP client, 528

D

datetime attribute, 106–108
debugging techniques
checking HTML, 510–511
syntax highlighting, 506–507
default page, specifying, 33–34. *See also* HTML pages; Web pages
default styles. *See also* styles
normalizing, 290–291
resetting, 290–291
default.htm page, 33
defining terms, 120
del element, 124–127
deleting, borders from images, 156
description list (**dl**). *See also* lists
creating, 414–415

dt and **dd** elements, 413
name-value groups, 412
nesting, 415
Devlin, Ian, 467, 487
dfn (definition) element, 120
Digital Rights Management (DRM), 481
disabled attribute, 447
display property, 324
block value, 325
inline value, 325
inline-block value, 325
none value, 325
div element
applying styles to, 85
best practices, 86
examples, 87
as generic container, 84–87
versus **span** element, 85
structuring pages with, 279
surrounding content, 84
using with JavaScript, 85
dl (description list). *See also* lists
creating, 414–415
dt and **dd** elements, 413
name-value groups, 412
nesting, 415
DOCTYPE
case insensitivity, 45
rendering in browsers, 45
for XHTML Strict document, 45
<!DOCTYPE html> declaration, 4, 24
document outline, 50–55
algorithm, 57
assistive technologies, 54
explicit semantics, 53
h1-**h6** hierarchy, 51
screen readers, 54
sectioning elements, 51–52, 55
document structure, inspecting, 511
document.createElement(), 287
documents. *See also* HTML documents; Web pages
ending, 5
saving, 35
structuring, 278
domain name
connecting to Web host, 524
getting, 522
Dribbble site, 376
DRM (Digital Rights Management), 481
drop shadows
adding to elements, 384–387
adding to text, 382–383
drop-down navigation, using nested lists for, 411
Dunham, Ethan, 360–361

E

editing Web pages, 35
edits, noting, 124–127
Electric Mobile Simulator for Windows, 347
element box
controlling appearance of, 293
positioning, 293
elements, 13–14. *See also* pseudo-elements
adding padding around, 304–305
aligning vertically, 322
ancestors, 221
applying styles to groups of, 236–237
auto value for **width**, 300
contents of, 13
descendants, 221
displaying, 324–326
empty, 13
end and start tags, 511
end tag, 13
floating, 306–310
formatting, 93
hiding, 324–326
naming with classes or IDs, 92–94
nesting, 15
offsetting in natural flow, 314–315
overlap with landmark roles, 88
positioning absolutely, 316–317
positioning in 3D, 318–319
rounding corners of, 378–381
selecting based on adjacent sibling, 226
selecting based on ancestors, 222–223
selecting based on attributes, 232–235
selecting based on child, 224–226
selecting based on parents, 223–224
selecting based on type, 217
selecting by **class** or **id**, 218–220
selecting by context, 221–226
selecting by name, 216–217

elements *(continued)*
selecting first letters of, 228
selecting first lines of, 227
selecting parts of, 227–229
setting height or width, 298–301
space and forward slash, 13
specifying groups of, 236–237
start and end tags, 511
start tag, 13
styling in browsers, 286–289
typing names of, 14
void, 13, 511
elliptical corners, creating, 380
em element, 110
calculating values for indents, 265
versus **i**, 110–111
versus **mark**, 117
using with paragraphs, 10
email boxes, creating, 432–433
embedded scripts, adding, 502
embedded style sheets, creating, 202–203
emulators, using with mobile devices, 347
encoding, choosing for files, 32
end tags, including in elements, 13
.eot (Embedded OpenType) Web fonts, 354
equals (=) sign versus colon (:), 205
event handlers. *See* JavaScript events
external scripts, loading, 499
external style sheets. *See also* style sheets
benefits of, 200
changing, 200
creating, 198–199
@import rules in, 207
importing, 199
linking to, 199–201
rules, 201
URLs in, 201

F

family trees, creating for Web pages, 15
Faulkner, Steve, 91, 168
favicons
adding to Web sites, 162–163
saving, 163
Featherstone, Derek, 168
Ferg, Stephen, 489
Fetch FTP client, 528
fieldset element, 426–427
figcaption element, 104–105
figure element
versus **aside** element, 77
using, 104–105
figures, creating, 104–105
file extensions, using with text editors, 32
file names
dashes between words, 19, 24
extensions, 19
filename, marking up, 128
.htm and **.html** extensions, 19, 30–31
lowercase, 19, 24
file scheme, 20
file uploads, allowing, 442
files
choosing encoding for, 32
compressing, 177
hiding, 34
naming, 36
organizing, 36
transferring to servers, 525–528
viewing, 35
FileZilla, transferring files to server with, 527
fine print, specifying, 132
Firebug for Firefox, 212, 507
Firefox, verifying sites in, 518
::first-letter syntax, 229
::first-line syntax, 229
Firtman, Maximiliano, 351
Flash animation, embedding, 482–483
Flash fallback player, 463
Flash fallbacks
audio, 476–477
and hyperlink fallback for audio, 478–479
and hyperlink fallback for video, 465–466
video, 464–465
float property
clearfix method, 310
left value, 306
none value, 307
overflow method, 310
right value, 306
using **clear** property with, 308–310
focus links, 230
folder names, lowercase, 24

folders
naming, 36
organizing files in, 36
sub-folders, 36
using, 370
font declarations, combining, 257
font families
naming, 364
setting, 243
font files, organizing in folders, 370
font names
generic, 245
with non-ASCII characters, 243
using quotes with, 243
font size
% unit, 250, 251–252, 254
child of element, 254
em unit, 251–252, 254
ex unit, 254
parent element, 254
pt unit type, 253
px unit, 250–254
rem unit, 254
setting, 250–254
Font Squirrel, 355–356, 358, 366
font values, setting all, 256–257
Fontdeck service, 356
@font-face feature, 355, 360–364
rule, 360
syntax, 360
using, 370
font-family property, 243–244, 247
fonts. *See also* Web fonts
defaults, 244
specifying, 244–245
Fonts.com service, 356
FontShop, 356
Fontspring Web site, 361
font-style: italic, 247
font-variant property, 271
font-weight: bold, 248
font-weight: normal, 248
Food Sense home page, 330–331
footer element, 80–83
contents of, 81
examples, 82–83
placement of, 81
restrictions, 81
using with **article** element, 70
using with **role="contentinfo"**, 81
footer links, placing **nav** elements in, 65
footers, creating, 80–83
form data, sending via email, 424–425
form elements
for attribute, 435
disabling, 447
id attribute, 435
organizing, 426–427
output, 448
title attribute, 435
form input attributes, 428
form parts, labeling, 434–435
form element, 419
forms
creating, 419–420
disabled attribute, 447
fieldset element, 426–427
get method, 442
hidden fields in, 443
legend element, 426–427
processing, 421–423
resource for, 448
security, 423
submitting with images, 446
fractional values, indicating, 142–143
FTP clients
CyberDuck, 528
Fetch, 528
FileZilla, 525–528
resizing windows of, 528
Transmit, 528
FTP sites, defining properties for, 526

G

Gallagher, Nicolas, 123, 290
Gasston, Peter, 334
generic containers
creating, 84–87
div element, 84–87
GIF format
color, 149
images, 148
lossless, 151
transparency, 151

Google Apps, 518
Google Closure Compiler, 501
Google WebFonts, 356
Graded Browser Support, 518
gradient backgrounds, 390–393. *See also* backgrounds
gradient generator, 393
grouping
  headings, 58–59
  selectors, 237
groups of elements, specifying, 236–237

H

H.264 video file formats, 452
h1 heading, using, 9
h1-**h6** elements, 48
  sizes of, 49
  using consistently, 49, 55
HandBrake video converter, 452
hanging indent, creating, 265
hasLayout, 395
head element
  explained, 44
  indenting code nested in, 45
header element, 61–63, 279
headers
  creating, 61–63
  versus headings, 63
  nav element, 63
  restrictions, 63
  using, 63
headings, 282
  adding **id**s to, 49
  creating, 48–49
  grouping, 58–59
  versus headers, 63
  lang attribute in, 49
  in search engines, 49
  using, 9, 48
height: property
  versus **min-height**, 300
  setting, 298–299
height video attribute, 454
hexadecimal colors, 191
hgroup element, 58–59
hh:mm:ss format, 108
hidden fields
  creating, 443
  readonly attribute, 443
hiding files, 34
highlighting text, 116–117
homepage, specifying, 33–34
host, finding for sites, 523–524
hover links, 230
:hover pseudo-class, 231
href attribute
  beginning with **#**, 172
  values in, 15
HSL and HSLA color, 193–196
HSL Color Picker, 194–195
.htm and **.html** extensions, 19, 30–31
HTML
  checking, 514
  markup, 6
  semantic, 6, 24
  start and end tags, 511
  validating, 515
HTML code, viewing, 39
HTML comments. *See also* comments
  adding, 96–97
  restrictions, 97
  syntax, 97
HTML documents, beginning, 24. *See also* documents
HTML elements
  block-level, 7
  displaying, 6–8
  inline, 7
HTML forms. *See* forms
HTML Lint, 515
HTML markup, components, 24. *See also* markup
HTML pages. *See also* default page; Web pages
  above **<body>** start tag, 4
  basic page, 3
  carriage returns, 3
  DOCTYPE, 4, 24
  ending documents, 5
  foundation, 43
  h1 heading, 9
  headings, 9
  images, 9
  links, 10

organizing in folders, 370
paragraphs, 10
semantics, 9–10
testing, 516–517
text content, 3, 5
title element, 4
HTML troubleshooting
attribute values, 510
character formatting, 510
element nesting, 510
typos, 510
</html> end tag, 5
html element, 45
HTML5
DOCTYPE, 45
formatting code, 515
new forms features, 448
Outliner, 52
phrasing content, 7
semantics, 8–9
"HTML5 Canvas: The Basics," 487
HTML5 elements, 13–14. *See also* pseudo-elements
adding padding around, 304–305
aligning vertically, 322
ancestors, 221
applying styles to groups of, 236–237
auto value for **width**, 300
contents of, 13
descendants, 221
displaying, 324–326
empty, 13
end and start tags, 511
end tag, 13
floating, 306–310
formatting, 93
hiding, 324–326
naming with classes or IDs, 92–94
nesting, 15
offsetting in natural flow, 314–315
overlap with landmark roles, 88
positioning absolutely, 316–317
positioning in 3D, 318–319
rounding corners of, 378–381
selecting based on adjacent sibling, 226
selecting based on ancestors, 222–223
selecting based on attributes, 232–235
selecting based on child, 224–226
selecting based on parents, 223–224
selecting based on type, 217
selecting by **class** or **id**, 218–220
selecting by context, 221–226
selecting by name, 216–217
selecting first letters of, 228
selecting first lines of, 227
selecting parts of, 227–229
setting height or width, 298–301
space and forward slash, 13
specifying groups of, 236–237
start and end tags, 511
start tag, 13
styling in browsers, 286–289
typing names of, 14
void, 13, 511
HTML5 pages. *See also* Web pages
body section, 44
head section, 44
starting, 43–44
HTML5 shiv, 287
HTML5 Video, 487
html5.js, downloading, 289. *See also* HTML5 shiv
http scheme, 20
https://, using, 431–432
Hudson, Roger, 489
hyperlink fallbacks
audio, 475
video, 461–462

I

i element, 111
redefinition of, 111
versus **em** element, 110
icons, adding for Web sites, 162–163
id attribute
versus **class** attribute, 94
naming elements with, 92–94
selecting elements by, 218–220
id vs. **class** selectors, 220
ids versus ARIA landmark roles, 284–285
image editors
Adobe Fireworks, 153
Adobe Photoshop, 153
choosing, 153
scaling images with, 161

image size, specifying, 158–159
images, 17. *See also* poster images
  adding borders to, 156
  adding to pages, 9
  alternate text, 157
  animation, 151
  color, 149
  Creative Commons licenses, 152
  deleting borders from, 156
  format, 148
  getting, 152
  GIF format, 148, 151
  including in HTML pages, 9
  inserting on pages, 156
  JPEG format, 148, 150
  missing, 519
  pixels, 149
  PNG format, 148, 151
  saving, 154–155, 519
  scaling with browsers, 160
  scaling with image editor, 161
  size and resolution, 149–150
  speed, 150–151
  transparency, 151
  troubleshooting, 519
  using to submit forms, 446
`img` element, using, 9
`@import` rules
  in external style sheets, 207
  in **`style`** element, 206
`!important`, marking styles with, 207
indenting code, 5, 45
indents
  adding, 265
  removing, 265
`index.html` default page, 33–34
inline elements, 7
inline scripts, 500
inline styles, applying, 204–205
`ins` element, 124–127
inset shadow, creating, 385
Internet Explorer
  Developer Tools, 507
  Gradient filter, 195–196
  recognizing CSS, 287
  verifying sites in, 518
iOS Simulator, 347
iPad
  rendering, 345
  testing pages for, 347
iPhones
  support for media queries, 341
  testing pages for, 347
Irish, Paul, 377
ISP, using as Web host, 523
italics, creating, 246–247

J

JavaScript
  adding to Web pages, 499–500
  `document.createElement()`, 287
  inline scripts, 500
  libraries, 498
  Modernizr library, 287
  polyfills for progressive enhancement, 376–377
JavaScript events
  `onblur`, 503
  `onchange`, 503
  `onclick`, 503
  `ondblclick`, 503
  `onfocus`, 503
  `onkeydown`, 503
  `onkeypress`, 504
  `onkeyup`, 504
  `onload`, 504
  `onmousedown`, 504
  `onmousemove`, 504
  `onmouseout`, 504
  `onmouseover`, 504
  `onmouseup`, 504
  `onreset`, 504
  `onselect`, 504
  `onsubmit`, 504
  touch-based handlers, 504
  `touchend`, 504
  `touchmove`, 504
  `touchstart`, 504
JAWS screen reader, 91
Jehl, Scott, 348
Johansson, Roger
  “Bring On the Tables,” 489
  `@font-face` code, 370

Johnston, Jason, 377
JPEG format, 148, 150
jQuery JavaScript library, 498
JW Player, 463

K

kbd element, 129
Keith, Jeremy, 332
kerning, specifying, 264
-khtml- prefix, 373
Kiss, Jason, 91
Kissane, Erin, 27

L

label element
  example of, 14
  using with forms, 434
landmark roles
  versus **id**s, 284–285
  overlap with HTML5 elements, 88
  recommendation, 90
  styling elements with, 284–285
lang attribute, 43
  in headings, 49
  using with **q** element, 114
layout with styles. *See also* styles
  approaches, 277–278
  background color, 296
  background images, 294–295
  background properties, 296–297
  box model, 292–293
  browsers, 276–277
  content and presentation, 276
layouts
  elastic, 278
  fixed, 277
  fluid, 277–278
The League of Moveable Type, 355–356
“Learning SVG,” 487
legend element, 426–427
letter spacing, setting, 264
li (list item) elements, 398–400
line break, creating, 133, 137
line height, setting, 255
line spacing, fixing, 123
linking thumbnail images, 177
links, 17. *See also* anchors
  active, 230
  block-level, 168–170
  changing appearance of, 230
  creating, 167–170
  defining, 10
  defining rules for, 231
  designating for navigation, 65
  destination, 166
  focus, 230
  hover, 230
  labels, 166, 170
  LVFHA mnemonic, 231
  marking up groups of links with, 399
  nesting in **nav** element, 64
  opening, 171
  selecting based on states, 230–231
  structuring in **ul** and **ol** elements, 65
  target attribute, 171
  visited, 230
  wrapping in **nav** element, 66–67
list content, placement of, 400
list item (**li**) elements, 398–400
list numbering, starting, 403
list type, choosing, 399
lists. *See also* **dl** (description lists); nested lists
  choosing markers, 401–402
  creating, 398–400
  custom markers, 404–405
  displaying without markers, 402
  hanging markers, 406
  indenting, 400
  nesting, 400
  ordered (**ol**), 398–400
  right-to-left content direction, 400
  start value, 403
  unordered (**ul**), 398–400
  value attribute, 403
list-style properties, setting, 407
list-style-type property, 401
loop
  audio attribute, 471
  video attribute, 454
lowercase value, using with **text-transform**, 270

M

mailto scheme, 20
Marcotte, Ethan, 331
margins
  auto value, 302–303
  setting around elements, 302–303
  setting values for, 301
mark element, 116–117
markers. *See also* custom markers
  choosing for lists, 401
  controlling hanging, 406
  inside value, 406
  outside value, 406
markup, defined, 1, 6. *See also* HTML markup
math element, 129
mathML element, 129
max-width property, setting, 299
McLellan, Drew, 483
@media at-rule, using in style sheets, 208–209
media queries
  building pages adapting with, 349–350
  chaining features and values, 336
  content and HTML, 340–341
  declarations in rules, 338
  defining, 336–337
  design implementation, 341–342
  evolving layout, 343–346
  examples, 334–336, 344–345
  feature: value pair, 335
  features of, 333–334
  iPhone 4, 351
  logic portion, 335
  min-width and **max-width**, 348
  Opera Mobile 11 browser, 351
  rendering styles in Internet Explorer, 348
  syntax, 334–336
  type portion, 335
  width feature, 338
media sources, **source** element, 460
media-specific style sheets, 208–209
meta element, 339
meter element
  versus **progress** element, 143
  using, 142–143
Meyer, Eric, 290
min-height versus **height**, 300
Miro Video Converter, 452
Mobile Boilerplate, 347, 350
mobile coding tools, 346
mobile devices, HTML5 and CSS3 support for, 351
"mobile first" design, 332
mobile phones. *See also* responsive Web design
  base styling, 340
  building baseline for, 341–342
  building for desktop, 342
  building sites for, 328–332
  testing pages on, 347
Mobile Safari's viewport, 335
Modernizr JavaScript library, 287, 348, 377
monospaced font, rendering, 129
MooTools JavaScript library, 498
-moz- prefix, 373, 378–379
MP3 audio file format, 468
MP4
  audio file format, 468
  video file formats, 452
-ms- prefix, 373
multimedia files, getting, 480
multimedia resources, 487
muted
  audio attribute, 471
  video attribute, 454
MyFonts, 356

N

nav element, 64–67
  in headers, 63
  nesting links in, 64
  placing footer links in, 65
  restrictions, 65
  role attribute, 64
  using with screen readers, 65
  wrapping links in, 66–67
navigation
  with keyboard, 170
  marking, 64–67
Neal, Jonathan, 123, 290, 466
nested lists. *See also* lists
  drop-down navigation, 411
  :hover pseudo-class, 411
  selectors, 409
  styling, 408–411

none value, using with **text-transform**, 270
normalize.css, 123, 290–291
Notepad text editor, using, 28–30
NVDA screen reader, 91

O

Ogg Theora video file formats, 452
Ogg Vorbis audio file format, 468
ol (ordered list)
  Arabic numerals, 409
  creating, 398–400
  marker types, 402
  using with links, 65
onblur JavaScript event, 503
onchange JavaScript event, 503
onclick JavaScript event, 503
ondblclick JavaScript event, 503
"One Web" presentation, 332
onfocus JavaScript event, 503
onkeydown JavaScript event, 503
onkeypress JavaScript event, 504
onkeyup JavaScript event, 504
online resources
  320 and Up, 351
  Apple's Link Maker, 177
  ARIA spec, 91
  BOM, 32
  browser compatibility, 375
  browser developer tools, 507
  Coda text editor, 29
  collapse value for **visibility**, 326
  ColorZilla's gradient generator, 393
  conditional comments, 351
  Creative Commons licenses, 152
  CSS error checking, 515
  CSS Tricks, 395
  CSS3 Generator, 374
  CSS3 selectors, 239
  developer tools, 507
  Electric Mobile Simulator for Windows, 347
  event handlers, 504
  Firebug for Firefox, 212
  Font Squirrel, 355–356, 358, 366
  Fontdeck service, 356
  Fonts.com service, 356
  FontShop, 356
  Fontspring, 361
online resources *(continued)*
  forms, 428
  Google Apps, 518
  Google Closure Compiler, 501
  Google WebFonts, 356
  Graded Browser Support, 518
  gradient backgrounds, 392
  gradient generator, 393
  HandBrake, 452
  hasLayout, 395
  HTML forms, 428
  HTML Lint, 515
  "HTML5 Canvas: The Basics," 487
  HTML5 Video, 487
  HTML5's new features, 448
  iOS Simulator, 347
  JavaScript events, 504
  JavaScript libraries, 498
  jQuery JavaScript library, 498
  JW Player, 463
  The League of Moveable Type, 355–356
  "Learning SVG," 487
  Meyer reset, 290
  Miro Video Converter, 452
  Mobile Boilerplate, 347, 350
  mobile devices, 351
  "mobile first" design, 332
  Modernizr, 287, 348, 377
  MooTools JavaScript library, 498
  multimedia, 487
  MyFonts, 356
  normalize css, 123
  "One Web" presentation, 332
  PHP server-side language, 422
  polyfills, 377
  ProtoFluid, 347
  right-to-left languages, 141
  showform.php script, 420
  SitePoint, 520
  Stack Overflow, 520
  Sublime Text editor, 29
  table structures, 489
  text editors, 29
  TextMate, 29
  TextWrangler, 28
  Typekit service, 356–357, 359
  validating code, 515

online resources *(continued)*
video, 487
video converters, 452
"Video for Everybody," 466
Video for Everybody Generator, 466
"Video on the Web," 487
Web Font Specimen, 357
WebINK service, 356
WebVTT (Web Video Text Tracks), 467
"WebVTT and Video Subtitles," 487
Wufoo, 448
YouTube video, 484
YUI Compressor, 501
YUI JavaScript library, 498
onload JavaScript event, 504
onmousedown JavaScript event, 504
onmousemove JavaScript event, 504
onmouseout JavaScript event, 504
onmouseover JavaScript event, 504
onmouseup JavaScript event, 504
onreset JavaScript event, 504
onselect JavaScript event, 504
onsubmit JavaScript event, 504
opacity, setting for elements, 394–395
Opera
Dragonfly, 507
verifying sites in, 518
ordered list (**ol**)
Arabic numerals, 409
creating, 398–400
marker types, 402
using with links, 65
outline algorithm, 57
output form element, 448
overflow, treatment by browsers, 320–321
overflow property
auto value, 320
hidden value, 320
scroll value, 320
visible value, 320–321

P

p element, using, 100
padding
adding around elements, 304–305
-bottom suffix, 305
-left suffix, 305
-right suffix, 305
-top suffix, 305
page constructs
information, 60
layout, 60
semantics, 60
pages. *See* HTML pages; Web pages
paragraphs
a element, 10
em element, 10
inspecting, 511
marking up, 10
starting, 100–101
parents and children, 15
password boxes, creating, 431
passwords, protecting, 431
patterns, finding, 433
Pfeiffer, Silvia, 487
Photoshop, 153–155
finding image sizes, 159
mockups, 359
scaling images, 161
PHP server-side language, 421–423
phrases, quoting, 114
pixels, 149
placeholder value, representing, 129
placeholders versus **label**s, 434
player.swf file, 463
plugins, 18, 451
PNG format, 148
alpha transparency, 151
color, 149
lossless, 151
polyfills, using for progressive enhancement, 376–377
position: absolute, 316
position: fixed, 317
position: relative, 314–316
position: static, 317
positioning, relative, 314–315
poster images, specifying for videos, 457. *See also* images
poster video attribute, 454
Powers, Shelley, 487
pre element, 130–131
math-related markup, 129
using with **white-space** property, 267

preload
audio attribute, 471, 473
video attribute, 454, 458
print, fine, 132
progress element
versus **meter** element, 143
using, 144–145
progressive enhancement
applying, 330–331
Dribbble site, 376
using polyfills for, 376–377
pronunciation, indicating, 138
ProtoFluid, downloading, 347
pseudo-classes, 229. *See also* **class** attribute
:active, 231
:hover, 231
pseudo-elements. *See also* elements
::after, 229
::before, 229
::first-letter, 229
::first-line, 229
pubdate attribute, 106–107
specifying, 109
using with **time**, 109

Q

q element, 114–115
quotes
enclosing attribute values in, 510
using with font names, 243
quoting
phrases, 114
text, 113–115

R

radial gradients, 391
radio buttons
creating, 436–437
name attribute, 436
value attribute, 436–437
readonly attribute, using with hidden fields, 443
references, indicating, 112
regular expressions, use of, 433
relative positioning, 314–315
relative URLs
versus absolute URLs, 21–23
using, 169
Resig, John, 287
resources
320 and Up, 351
Apple's Link Maker, 177
ARIA spec, 91
BOM, 32
browser compatibility, 375
browser developer tools, 507
Coda text editor, 29
collapse value for **visibility**, 326
ColorZilla's gradient generator, 393
conditional comments, 351
Creative Commons licenses, 152
CSS error checking, 515
CSS Tricks, 395
CSS3 Generator, 374
CSS3 selectors, 239
developer tools, 507
Electric Mobile Simulator for Windows, 347
event handlers, 504
Firebug for Firefox, 212
Font Squirrel, 355–356, 358, 366
Fontdeck service, 356
Fonts.com service, 356
FontShop, 356
Fontspring, 361
forms, 428
Google Apps, 518
Google Closure Compiler, 501
Google WebFonts, 356
Graded Browser Support, 518
gradient backgrounds, 392
gradient generator, 393
HandBrake, 452
hasLayout, 395
HTML forms, 428
HTML Lint, 515
"HTML5 Canvas: The Basics," 487
HTML5 Video, 487
HTML5's new features, 448
iOS Simulator, 347
JavaScript events, 504
JavaScript libraries, 498
jQuery JavaScript library, 498
JW Player, 463
The League of Moveable Type, 355–356
"Learning SVG," 487
Meyer reset, 290

resources *(continued)*
Miro Video Converter, 452
Mobile Boilerplate, 347, 350
mobile devices, 351
"mobile first" design, 332
Modernizr, 287, 348, 377
MooTools JavaScript library, 498
multimedia, 487
MyFonts, 356
`normalize.css`, 123
"One Web" presentation, 332
PHP server-side language, 422
polyfills, 377
ProtoFluid, 347
right-to-left languages, 141
`showform.php` script, 420
SitePoint, 520
Stack Overflow, 520
Sublime Text editor, 29
table structures, 489
text editors, 29
TextMate, 29
TextWrangler, 28
Typekit service, 356–357, 359
validating code, 515
video, 487
video converters, 452
"Video for Everybody," 466
Video for Everybody Generator, 466
"Video on the Web," 487
Web Font Specimen, 357
WebINK service, 356
WebVTT (Web Video Text Tracks), 467
"WebVTT and Video Subtitles," 487
Wufoo, 448
YouTube video, 484
YUI Compressor, 501
YUI JavaScript library, 498
`respond.js` script, 348, 351
responsive Web design, 331–332, 341–342. *See also* mobile phones
defining styles for breakpoints, 343
grid-based layout, 331
images and media, 331
media queries, 331, 343
pixel widths, 343, 346
`reversed` attribute, 400
RGB color, 191
RGBA color, 193–196
`role` attribute, using with **`nav`** element, 64
`role="banner"` definition, 89
`role="complementary"` definition, 90
`role="contentinfo,"` using with **`footer`** element, 81
`role="contentinfo"` definition, 90
`role="main,"` using with **`article`** element, 69
`role="main"` definition, 89
`role="navigation"` definition, 89
`rp` element, 138–139
`rt` element, 138–139
`ruby` element, 138–139

S

`s` element, 126–127
Safari
verifying sites in, 518
Web Inspector, 507
`samp` element, 129
saving
animated images, 151
changes to documents, 35
favicons, 163
files as UTF-8, 45
images, 154–155, 519
Web pages, 30–32
Scalable Vector Graphics (SVG)
coupling with video, 486
Web fonts, 354
scaling images
with browser, 160
with image editor, 161
with Photoshop, 161
screen readers, 12, 54
availability of, 91
JAWS, 91
landmark role support, 91
`nav` element, 65
NVDA, 91
VoiceOver, 91
`script` element
best practices, 501
blocking behavior, 501
processing, 502
`</script>` end tab, 502
`src` attribute, 499

scripting best practices, 501
scripts
  adding embedded, 502
  Google Closure Compiler, 501
  loading external, 499
  YUI Compressor, 501
search engine optimization (SEO), 12
section element
  versus **article** element, 69, 73, 283
  example, 74
  terminology, 50
  using, 72–74
  using with **role="main"**, 69
sections, defining, 72–74
secure server, using, 431
Seddon, Ryan, 377
select boxes
  creating, 438–439
  grouping options, 439
  option element, 438–439
  select element, 438
  size attribute, 438–439
selectors. *See also* attribute selectors
  combining, 238–239
  constructing, 214–215
  grouping, 237
semantics
  accessibility, 11
  displaying HTML, 12
  importance of, 11–12
  screen readers, 12
semicolon (;), using with CSS properties, 205
SEO (search engine optimization), 12
server, transferring files to, 525–528
server side vs. client side, 421
Sexton, Alex, 377
shims, using for progressive enhancement, 376–377
showform.php script, downloading, 420
sidebar, **aside** as, 76–77
simulators, using with mobile devices, 347
SitePoint, 520
sites
  HTML 5 Outliner, 52
  loading, 528
  planning, 26
  sketching out, 26–27
small caps
  removing, 271
  using, 271
small element, 8, 132
Sneddon, Geoffrey, 52
Snook, Jonathan, 254
source code, saving, 40
source element
  media attribute, 460
  type attribute, 460
  using with multiple media, 460
  using with **video** element, 461–462
spacing
  controlling, 264
  fixing between lines, 123
span element
  versus **div** element, 85
  using, 134
spans, creating, 134–135
src attribute
  audio, 471
  contents, 15
  video, 454, 460
Stack Overflow, Web resources, 520
stacking order, specifying, 315, 317
start tags, including in elements, 13
strikethrough, applying, 126
strong element, 110
  versus **b** element, 110
  versus **i** element, 110
  versus **mark**, 117
  nesting, 110
style element, **@import** rules in, 206
style rules
  adding comments to, 182–183
  cascading, 185–187
  constructing, 181
  creating, 237
  declaration blocks, 181
  inheritance, 185–186
  location of, 187
  selectors, 181
  specificity, 186–187
style sheets. *See also* external style sheets
  alternate, 210–211
  @charset declaration, 199
  CSS reset, 290

style sheets *(continued)*
embedded, 202–203
external, 198–200
linking to, 201
media-specific, 208–209
naming, 199
organizing in folders, 370
persistent styles, 210
preferred styles, 210–211
rendering headings in, 7
styles. *See also* default styles; layout with styles
applying to groups of elements, 236–237
location of, 206–207
styles-480.css, styles in, 333
sub element, 121–122
Sublime Text editor, 29
submit button
button element, 445
creating, 444–445
with image, 444
labeling, 445
name/value pair, 445
subscripts, creating, 121–122
sup element, 121–122
superscripts, creating, 121–122
SVG (Scalable Vector Graphics)
coupling with video, 486
Web fonts, 354
syndicated content, management of, 57
syntax highlighting, using, 39, 506

T

Tab key, pressing, 170
table element, 489
tables
borders for data cells, 492
caption text, 490
cells, 490
colspan attribute, 494–495
figcaption, 493
headers, 490
padding, 492
rows, 490
rowspan attribute, 494–495
scope attribute, 490, 493
spanning columns and rows, 494–495
structuring, 490–493
tbody element, 491
td element, 490
tfoot element, 491
thead element, 491
tr element, 490
target attribute, 171
telephone boxes, creating, 432–433
terms, defining, 120
testing
browsers, 518
HTML pages, 516–517
local versions of sites, 517
testing techniques
enabling browser features, 509
file uploads, 508–509
saving files, 509
URL entry, 509
validating code, 508
text
adding drop shadows to, 382–383
aligning, 268–269
alternate, 157
decorating, 272–273
deleting, 124
emphasizing, 110
highlighting, 116–117
inserting, 124
marking important, 110
noting inaccuracies, 124–127
quoting, 113–115
removing decorations, 273
using preformatted, 130–131
text areas
cols, 441
creating, 441
maximum characters, 441
maxlength, 441
rows, 441
text background, changing, 260–263
text boxes
autofocus attribute, 430
creating, 428–430
maxlength attribute, 430
name attribute, 428
placeholder attribute, 429–430
required attribute, 429

text case, changing, 270
text content, 16–17. *See also* content
text editors
  choosing encoding, 32
  default extensions, 32
  using, 28–29
text-decoration property, 272–273
TextMate editor, 29–30
text-only format
  choosing, 32
  saving Web pages in, 30
text-shadow property, 382–383
text-transform property, 270
TextWrangler
  downloading, 28
  using, 28
thumbnail images, linking, 177
time, specifying, 106–109
time element, 106–107
  restrictions, 109
  using with **pubdate**, 109
title attribute
  adding to elements, 95
  versus **title** element, 95
  using with **abbr** element, 118
  using with **dfn** element, 120
title element, 4
  best practices, 47
  core message, 47
  placement of, 46
  restrictions, 47
  special characters in, 47
  versus **title** attribute, 95
titles
  creating, 46–47
  length of, 46
  using as linked text, 46
tool tip labels, adding, 95
touchend JavaScript event, 504
touchmove JavaScript event, 504
touchstart JavaScript event, 504
troubleshooting
  CSS, 512–513
  enabling browser features, 509
  file uploads, 508–509
  HTML, 510–512
  images, 519
  resources, 520
  saving files, 509
  techniques, 520
  URL entry, 509
  validating code, 508
TrueDoc Web fonts, 354
.ttf (TrueType) Web fonts, 354, 359
Typekit service, 356–357, 359
typos, correcting, 510–511

U

u element, 136
Uggedal, Eivind, 350
ul (unordered list)
  creating, 398–400
  using with links, 65
Ullman, Larry, 422
Unicode, 16
uppercase value, using with **text-transform**, 270
URL boxes, creating, 432–433
URLs (Uniform Resource Locators), 20–23
  absolute versus relative, 21–23
  creating, 175–177
  file scheme, 20
  http scheme, 20
  lowercase letters, 170
  mailto scheme, 20
  scheme, 20
  server name, 20
  trailing forward slash, 20
  using **cite** and **blockquote** with, 114
  visiting, 528
user input instructions, marking up, 129
UTF-8 encoding
  choosing, 32
  saving files in, 45

V

validating code, 514–515
var element, 129
vendor prefixes, 373–374
vertical-align property
  baseline value, 322
  bottom value, 322
  middle value, 322
  sub value, 322

vertical-align property *(continued)*
super value, 322
text-bottom value, 322
text-top value, 322
top value, 322
video
adding to Web pages, 453
adding with Flash fallbacks, 463–466
autoplay attribute, 455–456
books, 487
controls attribute, 455–456
coupling with SVG (Scalable Vector Graphics), 486
embedding YouTube, 484
hyperlink fallbacks, 461–462
looping, 457
multiple sources, 459
object element for Flash fallbacks, 463–466
online resources, 487
preload attribute, 454
preventing preloading, 458
specifying poster images, 457
using with **canvas** element, 485
video attributes
autoplay, 454–456
autoplay and **loop**, 457
controls, 454–456
height, 454
loop, 454
muted, 454
poster, 454
preload, 454
src, 454, 460
width, 454
video element, using with **source** element, 461–462
video file formats
converting between, 452
H.264, 452
MP4, 452
Ogg Theora, 452
WebM, 452–453, 455, 457
“Video for Everybody,” 466
Video for Everybody Generator, 466
“Video on the Web,” 487
View Source command, using, 39
viewports
features of, 339
meta element, 339
visibility property, 324
collapse value, 326
hidden value, 326
visited links, 230
visitors, allowing to upload files, 442
Visscher, Sjoerd, 287
VoiceOver screen reader, 91
void elements, omitting end tags from, 511

W

WAI-ARIA. *See* ARIA (Accessible Rich Internet Applications)
WAV audio file format, 468
wbr element, 137
Web design. *See* responsive Web design
Web Font Specimen, 357
Web fonts. *See also* fonts
bold formatting, 366–369
browser support, 355
demo.html file, 358–359
downloading, 358–359
.eot (Embedded OpenType), 354
features of, 354
file types, 354
finding, 356–357
@font-face feature, 355
incorporating into Web pages, 361–362
italic formatting, 366–369
legal issues, 355
managing file sizes, 365–369
quality, 357
rendering, 357
self-hosting, 356
services, 356–357, 370
styling, 365–369
subsetting, 365–366
.svg (Scalable Vector Graphics), 354
TrueDoc, 354
.ttf (TrueType), 354, 359
using, 362–364
using for headlines, 369
.woff (Web Open Font Format), 354

Web host
connecting to domain, 524
finding for sites, 523–524
Web Open Font Format (.woff), 354
Web pages. *See also* default page; documents; HTML pages
`article` versus **`section`** elements, 283
background-related capabilities, 261
blog entries, 283
comments, 282
components, 24
containers, 279
content, 24
creating, 28–29
`div`s, 279
editing, 35
family trees, 15
file references, 1
`footer` element, 279
`header` element, 279
heading elements, 282
HTML, 2
marking up, 283
markup, 1
ordering content, 282
saving, 30–32
structure, 279–283
text content, 1, 16–17
viewing in browsers, 37–38
Web resources
320 and Up, 351
Apple's Link Maker, 177
ARIA spec, 91
BOM, 32
browser compatibility, 375
browser developer tools, 507
Coda text editor, 29
`collapse` value for **`visibility`**, 326
ColorZilla's gradient generator, 393
conditional comments, 351
Creative Commons licenses, 152
CSS error checking, 515
CSS Tricks, 395
CSS3 Generator, 374
CSS3 selectors, 239
developer tools, 507
Web resources *(continued)*
Electric Mobile Simulator for Windows, 347
event handlers, 504
Firebug for Firefox, 212
Font Squirrel, 355–356, 358, 366
Fontdeck service, 356
Fonts.com service, 356
FontShop, 356
Fontspring, 361
forms, 428
Google Apps, 518
Google Closure Compiler, 501
Google WebFonts, 356
Graded Browser Support, 518
gradient backgrounds, 392
gradient generator, 393
HandBrake, 452
`hasLayout`, 395
HTML forms, 428
HTML Lint, 515
"HTML5 Canvas: The Basics," 487
HTML5 Video, 487
HTML5's new features, 448
iOS Simulator, 347
JavaScript events, 504
JavaScript libraries, 498
jQuery JavaScript library, 498
JW Player, 463
The League of Moveable Type, 355–356
"Learning SVG," 487
Meyer reset, 290
Miro Video Converter, 452
Mobile Boilerplate, 347, 350
mobile devices, 351
"mobile first" design, 332
Modernizr, 287, 348, 377
MooTools JavaScript library, 498
multimedia, 487
MyFonts, 356
`normalize.css`, 123
"One Web" presentation, 332
PHP server-side language, 422
polyfills, 377
ProtoFluid, 347
right-to-left languages, 141
`showform.php` script, 420

Web resources *(continued)*
SitePoint, 520
Stack Overflow, 520
Sublime Text editor, 29
table structures, 489
text editors, 29
TextMate, 29
TextWrangler, 28
Typekit service, 356–357, 359
validating code, 515
video, 487
video converters, 452
"Video for Everybody," 466
Video for Everybody Generator, 466
"Video on the Web," 487
Web Font Specimen, 357
WebINK service, 356
WebVTT (Web Video Text Tracks), 467
"WebVTT and Video Subtitles," 487
Wufoo, 448
YouTube video, 484
YUI Compressor, 501
YUI JavaScript library, 498
Web sites
HTML 5 Outliner, 52
loading, 528
planning, 26
sketching out, 26–27
WebINK service, 356
-webkit- prefix, 373, 378–379
WebM videos
autoplay and **loop**, 457
autoplay attribute, 456
controls attribute, 455
described, 454
without controls, 453
WebVTT (Web Video Text Tracks), 467
"WebVTT and Video Subtitles," 487
white space properties, setting, 266–267
width: property
auto property, 300
setting, 298–299
width video attribute, 454
.woff (Web Open Font Format), 354
word processors, avoiding use of, 29
word spacing, setting, 264
Wroblewski, Luke, 332, 351

X

XHTML Strict document, **DOCTYPE** for, 45

Y

YouTube video, embedding, 484
YUI Compressor, 501
YUI JavaScript library, 498
YYYY-MM-DD format, 108

Z

z-index property, 315, 317–319