Dedication
To family.
To those I know who endured difficult challenges, demonstrating courage and perseverance all the way.
Acknowledgments

One of my favorite parts of working on this book has been the people I’ve been able to work with. All are dedicated, professional, good-natured, and good-humored folks who made it a real pleasure. The book wouldn’t be the same without their contributions.

A grateful, sincere thank you goes out to:
Nancy Aldrich-Ruenzel and Nancy Davis, for their continued trust in me.
Cliff Colby, for his support, for bringing the team together, and for keeping things light.
Robyn Thomas, for making the engine go, improving copy, tracking all the details, being flexible, and providing encouragement.
Scout Festa, for her skill in simplifying language, for her watchful eye, and for helping to keep things consistent and polished.
Aubrey Taylor, for all the great suggestions and technical feedback. They were very helpful, and readers are better off for them.
David Van Ness, for leading the charge in making it all look great and for all his efforts in refining the layouts.
Valerie Haynes Perry, for compiling the all-important index, which will be the first destination for many readers in search of information.
The marketing, sales, and other folks at Peachpit, for working hard to make the book available to readers.
Natalia Ammon, for the wonderful design of the example webpage that adorns the pages of Chapters 11 and 12, and other spots. You can see more of her work at www.nataliaammon.com.
Zach Szukala, for recommending Natalia.
Scott Boms, Ian Devlin, Seth Lemoine, Erik Vorhes, and Brian Warren, for their contributions to the previous edition.
Victor Gavenda, for providing access to necessary software.
Dan Cederholm, Ethan Dunham, Paul Irish, Mark James, Randy Jensen, Julie Lamba, Fontfabric, Google, and Namecheap, for allowing me to use screen shots or design assets (as the case may be).
C.R. Freer, for working her camera magic.
My family and friends, for providing inspiration and breaks, for being patient, and for not disowning me while I was holed up writing for months.
Robert Reinhardt, as always, for getting me started in writing books and for having a swell beard.
The Boston Bruins, for providing a lot of playoffs thrills during my infrequent breaks.
The numerous folks in the web community who have shared their expertise and experiences for the betterment of others. (I’ve cited many of you throughout the book.)
To you readers, for inspiring me to recall when I began learning HTML and CSS so that I may explain them in ways I hope you find helpful. Thank you for choosing this book as part of your journey in contributing to the web. Happy reading!
And, lastly, I would like to give a special thank you to Elizabeth Castro, who created this title in the 1990s. She has taught countless readers how to build webpages over many editions and many years. Because the web has given me so much, I’m genuinely appreciative of the opportunity to teach readers via this title as well.

—Bruce
## Contents at a Glance

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction</td>
<td></td>
<td>xv</td>
</tr>
<tr>
<td>Chapter 1</td>
<td>Webpage Building Blocks</td>
<td>1</td>
</tr>
<tr>
<td>Chapter 2</td>
<td>Working with Webpage Files</td>
<td>27</td>
</tr>
<tr>
<td>Chapter 3</td>
<td>Basic HTML Structure</td>
<td>43</td>
</tr>
<tr>
<td>Chapter 4</td>
<td>Text</td>
<td>87</td>
</tr>
<tr>
<td>Chapter 5</td>
<td>Images</td>
<td>133</td>
</tr>
<tr>
<td>Chapter 6</td>
<td>Links</td>
<td>157</td>
</tr>
<tr>
<td>Chapter 7</td>
<td>CSS Building Blocks</td>
<td>169</td>
</tr>
<tr>
<td>Chapter 8</td>
<td>Working with Style Sheets</td>
<td>189</td>
</tr>
<tr>
<td>Chapter 9</td>
<td>Defining Selectors</td>
<td>203</td>
</tr>
<tr>
<td>Chapter 10</td>
<td>Formatting Text with Styles</td>
<td>229</td>
</tr>
<tr>
<td>Chapter 11</td>
<td>Layout with Styles</td>
<td>265</td>
</tr>
<tr>
<td>Chapter 12</td>
<td>Building Responsive Webpages</td>
<td>309</td>
</tr>
<tr>
<td>Chapter 13</td>
<td>Working with Web Fonts</td>
<td>335</td>
</tr>
<tr>
<td>Chapter 14</td>
<td>Enhancements and Effects with CSS</td>
<td>361</td>
</tr>
<tr>
<td>Chapter 15</td>
<td>Lists</td>
<td>389</td>
</tr>
<tr>
<td>Chapter 16</td>
<td>Forms</td>
<td>409</td>
</tr>
<tr>
<td>Chapter 17</td>
<td>Video, Audio, and Other Multimedia.</td>
<td>449</td>
</tr>
<tr>
<td>Chapter 18</td>
<td>Tables</td>
<td>477</td>
</tr>
<tr>
<td>Chapter 19</td>
<td>Adding JavaScript</td>
<td>485</td>
</tr>
<tr>
<td>Chapter 20</td>
<td>Testing &amp; Debugging Webpages</td>
<td>495</td>
</tr>
<tr>
<td>Chapter 21</td>
<td>Publishing Your Pages on the Web</td>
<td>511</td>
</tr>
<tr>
<td>Appendix</td>
<td>HTML Reference</td>
<td>519</td>
</tr>
<tr>
<td></td>
<td>Index</td>
<td>533</td>
</tr>
</tbody>
</table>
This page intentionally left blank
# Table of Contents

Introduction .......................................................... xv
HTML and CSS in Brief ................................. xvi
Web Browsers ............................................................. xvii
Web Standards and Specifications ................. xviii
Progressive Enhancement: A Best Practice ... xx
Is This Book for You? ............................................ xxi
How This Book Works ............................................ xxii
Companion Website .............................................. xxvi

## Chapter 1  Webpage Building Blocks ......................... 1
Thinking in HTML ..................................................... 3
A Basic HTML Page .................................................. 4
Markup: Elements, Attributes, Values, and More .... 8
A Webpage's Text Content ......................................... 12
Links, Images, and Other Non-Text Content .......... 13
File and Folder Names ............................................... 14
URLs .................................................................. 15
HTML: Markup with Meaning ...................................... 20
A Browser's Default Display of Webpages .............. 24
Key Takeaways .......................................................... 26

## Chapter 2  Working with Webpage Files ................. 27
Planning Your Site ..................................................... 28
Creating a New Webpage ............................................ 30
Saving Your Webpage ................................................ 32
Specifying a Default Page or Homepage ................ 35
Editing Webpages ...................................................... 36
Organizing Files ....................................................... 37
Viewing Your Page in a Browser .............................. 38
The Inspiration of Others .......................................... 40

## Chapter 3  Basic HTML Structure ......................... 43
Starting Your Webpage .............................................. 44
Creating a Title .......................................................... 48
<table>
<thead>
<tr>
<th>Chapter</th>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>Text</td>
<td>87</td>
</tr>
<tr>
<td></td>
<td>Adding a Paragraph</td>
<td>88</td>
</tr>
<tr>
<td></td>
<td>Specifying Fine Print</td>
<td>89</td>
</tr>
<tr>
<td></td>
<td>Marking Important and Emphasized Text</td>
<td>90</td>
</tr>
<tr>
<td></td>
<td>Creating a Figure</td>
<td>92</td>
</tr>
<tr>
<td></td>
<td>Indicating a Citation or Reference</td>
<td>94</td>
</tr>
<tr>
<td></td>
<td>Quoting Text</td>
<td>95</td>
</tr>
<tr>
<td></td>
<td>Specifying Time</td>
<td>98</td>
</tr>
<tr>
<td></td>
<td>Explaining Abbreviations</td>
<td>101</td>
</tr>
<tr>
<td></td>
<td>Defining a Term</td>
<td>103</td>
</tr>
<tr>
<td></td>
<td>Creating Superscripts and Subscripts</td>
<td>104</td>
</tr>
<tr>
<td></td>
<td>Adding Author Contact Information</td>
<td>106</td>
</tr>
<tr>
<td></td>
<td>Noting Edits and Inaccurate Text</td>
<td>108</td>
</tr>
<tr>
<td></td>
<td>Marking Up Code</td>
<td>112</td>
</tr>
<tr>
<td></td>
<td>Using Preformatted Text</td>
<td>114</td>
</tr>
<tr>
<td></td>
<td>Highlighting Text</td>
<td>116</td>
</tr>
<tr>
<td></td>
<td>Creating a Line Break</td>
<td>118</td>
</tr>
<tr>
<td></td>
<td>Creating Spans</td>
<td>120</td>
</tr>
<tr>
<td></td>
<td>Other Elements</td>
<td>122</td>
</tr>
<tr>
<td>5</td>
<td>Images</td>
<td>133</td>
</tr>
<tr>
<td></td>
<td>Images for the Web</td>
<td>134</td>
</tr>
<tr>
<td></td>
<td>Getting Images</td>
<td>140</td>
</tr>
<tr>
<td></td>
<td>Choosing an Image Editor</td>
<td>141</td>
</tr>
<tr>
<td></td>
<td>Saving Your Images</td>
<td>142</td>
</tr>
<tr>
<td></td>
<td>Inserting Images on a Page</td>
<td>145</td>
</tr>
<tr>
<td>Chapter 6</td>
<td>Links</td>
<td>157</td>
</tr>
<tr>
<td>-----------</td>
<td>-------</td>
<td>-----</td>
</tr>
<tr>
<td>Offering Alternative Text</td>
<td>147</td>
<td></td>
</tr>
<tr>
<td>Specifying Image Sizes</td>
<td>149</td>
<td></td>
</tr>
<tr>
<td>Scaling Images with the Browser</td>
<td>152</td>
<td></td>
</tr>
<tr>
<td>Scaling Images with an Image Editor</td>
<td>154</td>
<td></td>
</tr>
<tr>
<td>Adding Icons for Your Website</td>
<td>155</td>
<td></td>
</tr>
<tr>
<td>Creating a Link to Another Webpage</td>
<td>158</td>
<td></td>
</tr>
<tr>
<td>(and Other Link Basics)</td>
<td>158</td>
<td></td>
</tr>
<tr>
<td>Creating and Linking to Anchors</td>
<td>164</td>
<td></td>
</tr>
<tr>
<td>Creating Other Kinds of Links</td>
<td>166</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Chapter 7</th>
<th>CSS Building Blocks</th>
<th>169</th>
</tr>
</thead>
<tbody>
<tr>
<td>Constructing a Style Rule</td>
<td>171</td>
<td></td>
</tr>
<tr>
<td>Adding Comments to Style Rules</td>
<td>172</td>
<td></td>
</tr>
<tr>
<td>Understanding Inheritance</td>
<td>174</td>
<td></td>
</tr>
<tr>
<td>The Cascade: When Rules Collide</td>
<td>177</td>
<td></td>
</tr>
<tr>
<td>A Property's Value</td>
<td>180</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Chapter 8</th>
<th>Working with Style Sheets</th>
<th>189</th>
</tr>
</thead>
<tbody>
<tr>
<td>Creating an External Style Sheet</td>
<td>190</td>
<td></td>
</tr>
<tr>
<td>Linking to External Style Sheets</td>
<td>192</td>
<td></td>
</tr>
<tr>
<td>Creating an Embedded Style Sheet</td>
<td>194</td>
<td></td>
</tr>
<tr>
<td>Applying Inline Styles</td>
<td>196</td>
<td></td>
</tr>
<tr>
<td>The Cascade and the Order of Styles</td>
<td>198</td>
<td></td>
</tr>
<tr>
<td>Using Media-Specific Style Sheets</td>
<td>200</td>
<td></td>
</tr>
<tr>
<td>The Inspiration of Others: CSS</td>
<td>202</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Chapter 9</th>
<th>Defining Selectors</th>
<th>203</th>
</tr>
</thead>
<tbody>
<tr>
<td>Constructing Selectors</td>
<td>204</td>
<td></td>
</tr>
<tr>
<td>Selecting Elements by Name</td>
<td>206</td>
<td></td>
</tr>
<tr>
<td>Selecting Elements by Class or ID</td>
<td>208</td>
<td></td>
</tr>
<tr>
<td>Selecting Elements by Context</td>
<td>212</td>
<td></td>
</tr>
<tr>
<td>Selecting an Element That Is the First or Last Child</td>
<td>216</td>
<td></td>
</tr>
<tr>
<td>Selecting the First Letter or First Line of an Element</td>
<td>218</td>
<td></td>
</tr>
<tr>
<td>Selecting Links Based on Their State</td>
<td>220</td>
<td></td>
</tr>
<tr>
<td>Selecting Elements Based on Attributes</td>
<td>222</td>
<td></td>
</tr>
<tr>
<td>Specifying Groups of Elements</td>
<td>226</td>
<td></td>
</tr>
<tr>
<td>Combining Selectors</td>
<td>227</td>
<td></td>
</tr>
</tbody>
</table>
Chapter 10  Formatting Text with Styles  229
Before and After  230
Choosing a Font Family  232
Specifying Alternate Fonts  233
Creating Italic s  236
Applying Bold Formatting  238
Setting the Font Size  240
Setting the Line Height  245
Setting All Font Values at Once  246
Setting the Color  248
Setting the Background  250
Controlling Spacing  257
Adding Indents  258
Aligning Text  259
Changing the Text Case  260
Using Small Caps  261
Decorating Text  262
Setting Whitespace Properties  264

Chapter 11  Layout with Styles  265
Considerations When Beginning a Layout  266
Structuring Your Pages  268
Styling HTML5 Elements in Older Browsers  272
Resetting or Normalizing Default Styles  274
The Box Model  276
Controlling the Display Type and Visibility of Elements  278
Setting the Height or Width for an Element  282
Adding Padding Around an Element  286
Setting the Border  288
Setting the Margins Around an Element  292
Making Elements Float  295
Controlling Where Elements Float  297
Positioning Elements Relatively  301
Positioning Elements Absolutely  302
Positioning Elements in a Stack  304
Determining How to Treat Overflow  305
Aligning Elements Vertically  306
Changing the Cursor  308
<table>
<thead>
<tr>
<th>Chapter 12</th>
<th>Building Responsive Webpages</th>
<th>309</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Responsive Web Design: An Overview</td>
<td>310</td>
</tr>
<tr>
<td></td>
<td>Making Images Flexible</td>
<td>312</td>
</tr>
<tr>
<td></td>
<td>Creating a Flexible Layout Grid</td>
<td>315</td>
</tr>
<tr>
<td></td>
<td>Understanding and Implementing Media Queries</td>
<td>319</td>
</tr>
<tr>
<td></td>
<td>Putting It All Together</td>
<td>326</td>
</tr>
<tr>
<td></td>
<td>Accommodating Older Versions of Internet Explorer</td>
<td>333</td>
</tr>
<tr>
<td>Chapter 13</td>
<td>Working with Web Fonts</td>
<td>335</td>
</tr>
<tr>
<td></td>
<td>What Is a Web Font?</td>
<td>336</td>
</tr>
<tr>
<td></td>
<td>Where to Find Web Fonts</td>
<td>338</td>
</tr>
<tr>
<td></td>
<td>Downloading Your First Web Font</td>
<td>342</td>
</tr>
<tr>
<td></td>
<td>Understanding the @font-face Rule</td>
<td>345</td>
</tr>
<tr>
<td></td>
<td>Styling Text with a Web Font</td>
<td>346</td>
</tr>
<tr>
<td></td>
<td>Applying Italics and Bold with a Web Font</td>
<td>349</td>
</tr>
<tr>
<td></td>
<td>Using Web Fonts from Google Fonts</td>
<td>357</td>
</tr>
<tr>
<td>Chapter 14</td>
<td>Enhancements and Effects with CSS</td>
<td>361</td>
</tr>
<tr>
<td></td>
<td>Browser Compatibility, Progressive Enhancement, and Polyfills</td>
<td>362</td>
</tr>
<tr>
<td></td>
<td>Understanding Vendor Prefixes</td>
<td>364</td>
</tr>
<tr>
<td></td>
<td>Rounding the Corners of Elements</td>
<td>365</td>
</tr>
<tr>
<td></td>
<td>Adding Drop Shadows to Text</td>
<td>368</td>
</tr>
<tr>
<td></td>
<td>Adding Drop Shadows to Elements</td>
<td>370</td>
</tr>
<tr>
<td></td>
<td>Applying Multiple Backgrounds</td>
<td>373</td>
</tr>
<tr>
<td></td>
<td>Using Gradient Backgrounds</td>
<td>376</td>
</tr>
<tr>
<td></td>
<td>Setting the Opacity of Elements</td>
<td>382</td>
</tr>
<tr>
<td></td>
<td>Effects with Generated Content</td>
<td>384</td>
</tr>
<tr>
<td></td>
<td>Combining Images with Sprites</td>
<td>387</td>
</tr>
<tr>
<td>Chapter 15</td>
<td>Lists</td>
<td>389</td>
</tr>
<tr>
<td></td>
<td>Creating Ordered and Unordered Lists</td>
<td>390</td>
</tr>
<tr>
<td></td>
<td>Choosing Your Markers</td>
<td>393</td>
</tr>
<tr>
<td></td>
<td>Using Custom Markers</td>
<td>394</td>
</tr>
<tr>
<td></td>
<td>Choosing Where to Start List Numbering</td>
<td>397</td>
</tr>
<tr>
<td></td>
<td>Controlling Where Markers Hang</td>
<td>398</td>
</tr>
<tr>
<td></td>
<td>Setting All List-Style Properties at Once</td>
<td>399</td>
</tr>
<tr>
<td></td>
<td>Styling Nested Lists</td>
<td>400</td>
</tr>
<tr>
<td></td>
<td>Creating Description Lists</td>
<td>404</td>
</tr>
</tbody>
</table>
**Chapter 16**  
Forms .......................................................... 409  
Improvements to Forms in HTML5 ......................... 410  
Creating Forms .............................................. 413  
Processing Forms ........................................... 416  
Organizing the Form Elements ............................. 418  
Creating Text Boxes ........................................ 422  
Labeling Form Parts ........................................ 425  
Creating Password Boxes ................................ 427  
Creating Email, Search, Telephone, and URL Boxes ... 428  
Creating Radio Buttons .................................... 432  
Creating Checkboxes ....................................... 434  
Creating Text Areas ........................................ 436  
Creating Select Boxes ..................................... 437  
Allowing Visitors to Upload Files ......................... 439  
Creating Hidden Fields .................................... 440  
Creating a Submit Button .................................. 441  
Disabling Form Elements ................................... 444  
Styling Forms Based on Their State ....................... 446

**Chapter 17**  
Video, Audio, and Other Multimedia ....................... 449  
Third-Party Plugins and Going Native .................... 450  
Video File Formats ......................................... 451  
Adding a Video to Your Webpage ......................... 452  
Adding Controls and Autoplay to Your Video ............ 454  
Looping a Video and Specifying a Poster Image ........ 456  
Preventing a Video from Preloading ...................... 457  
Using Video with Multiple Sources and a Text Fallback 459  
Providing Accessibility .................................... 462  
Audio File Formats ......................................... 463  
Adding an Audio File with Controls to Your Webpage ... 464  
Autoplaying, Looping, and Preloading Audio .......... 466  
Providing Multiple Audio Sources with a Fallback .... 468  
Adding Video and Audio with a Flash Fallback ......... 470  
Advanced Multimedia ....................................... 475  
Further Resources .......................................... 476
| Chapter 18 | Tables   | 477 |
|           | Structuring Tables | 478 |
|           | Spanning Columns and Rows | 482 |
| Chapter 19 | Adding JavaScript | 485 |
|           | Loading an External Script | 487 |
|           | Adding an Embedded Script | 492 |
|           | JavaScript Events | 493 |
| Chapter 20 | Testing & Debugging Webpages | 495 |
|           | Validating Your Code | 496 |
|           | Testing Your Pages | 498 |
|           | Trying Some Debugging Techniques | 502 |
|           | Checking the Easy Stuff: General | 504 |
|           | Checking the Easy Stuff: HTML | 506 |
|           | Checking the Easy Stuff: CSS | 508 |
|           | When Images Don't Display | 510 |
| Chapter 21 | Publishing Your Pages on the Web | 511 |
|           | Getting Your Own Domain Name | 512 |
|           | Finding a Host for Your Site | 513 |
|           | Transferring Files to the Server | 515 |
| Appendix | HTML Reference | 519 |
|           | Index | 533 |
Whether you are just beginning your venture into building websites or have built some before but want to ensure that your knowledge is current, you’ve come along at a very exciting time.

How we code and style webpages, the browsers in which we view the pages, and the devices on which we visit the web 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.

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 from any metropolis, rural community, or anywhere in between and on any web-enabled device. In short, the web’s promise lies in its universality. And its reach continues to expand as technology finds its way to communities that were once shut out.

Better still, the web belongs to everyone, and 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 webpages. You’ll find clear, easy-to-follow instructions that take you through the process of creating pages step by step. And 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 webpage requires at least some HTML; it wouldn’t be a webpage without it.

As you will learn in greater detail as you read this book, HTML is used to define your content, and CSS is used to control how your content and webpage 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” later in this introduction for an overview of the chapters and a summary of the primary topics covered.

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. The same approach applies to usage of the terms CSS (general) and CSS3 (specific to CSS3).

HTML and HTML5

It helps to know some basics about the origins of HTML to understand HTML5.

HTML began in the early 1990s as a short document that detailed a handful of elements used to build webpages. Many of those elements were for content such as headings, paragraphs, lists, and links to other pages. 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 websites. 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, main, figure, and many more) that are used to describe content. Others are complex and aid in creating powerful web applications. You’ll need a firm grasp of creating webpages before you can graduate to the more complicated features that HTML5 provides, which is why this book focuses on the former. HTML5 also introduces native audio and video playback to your webpages, which the book also covers.

CSS and 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, rounded corners, gradients, and much more. (See “What this book will teach you” for details of what’s covered.)
We all use a web browser to visit websites, whether on a computer, a phone, or another device. However, the browser you use might be different than the one someone else uses.

Windows comes preinstalled with Internet Explorer, Microsoft’s browser. OS X comes preinstalled with Safari, Apple’s browser. There are other browsers you may download for free and use instead, such as Chrome (by Google), Firefox (by Mozilla), and Opera (by Opera Software)—and that’s just for the desktop.

On mobile devices, you’ll find the mobile version of Safari (for iPhone, iPad, and iPod touch); various default Android browsers; Chrome for Android; Firefox for Android; Opera Mini; and more.

I’ll refer to various browsers throughout the book. For the most part, the latest version of each one has similar support for the HTML and CSS features you’ll learn about. But sometimes a feature doesn’t work on one or more browsers (or works differently). I’ll note those cases and typically offer a way to handle them. This mostly pertains to Internet Explorer 8, the oldest browser that is still relevant enough to be of concern. (Its usage is dropping, so that could change in 2014 or so.)

“Testing Your Pages” in Chapter 20 provides information about how to acquire various browsers, which ones are the most important for testing your webpages, and how to test your pages.
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.

The W3C releases specifications (or specs, for short) that document these web standards. They define the parameters of languages like HTML and CSS. In other words, specs standardize the rules. Follow the W3C’s activity at www.w3.org.

The W3C site is the industry’s primary source of web standards specifications.

The W3C and WHATWG

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

If you want to dig into various specs (recommended!), here are the latest versions:

- HTML5 (W3C):
  http://www.w3.org/TR/html5/
- HTML5.1 (W3C):
  http://www.w3.org/TR/html51/
- HTML Living Standard (WHATWG):
  http://www.whatwg.org/specs/web-apps/current-work/multipage/

The HTML Living Standard includes newer features under development (and very much in flux) and informs the W3C’s HTML5.1 spec.

There are too many CSS specs to list, but you can see them at http://www.w3.org/standards/techs/css#w3c_all.
With standards in place, we can build our pages from the agreed-upon set of rules, and browsers can be built to display our pages with those rules in mind. (On the whole, browsers implement the standards well. Older versions of Internet Explorer, especially Internet Explorer 8, 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.

Differences Between HTML4 and HTML5

If you have prior experience with HTML4 and are wondering what is different in HTML5, the W3C has created just the document for you: http://www.w3.org/TR/html5-diff/.

I call out many of the differences at various points in the book. They aren’t particularly important to know for those of you who are new to HTML, because HTML5 is what virtually everyone uses now. But you might find the W3C’s document interesting to peruse regardless.
Progressive Enhancement: A Best Practice

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

The idea is simple but powerful: Start your site with HTML content and basic behavior that is accessible to all visitors A. To the same page, add your design with CSS B and additional behavior with JavaScript (a programming language). These components are kept separate but work together.
The result is that browsers capable of accessing basic pages will get the simplified, default experience A. Even browsers from the inception of the web more than 20 years ago can display this page; so too can the oldest or simplest of mobile phones with web browsers. And screen readers, software that reads webpages aloud to visually impaired visitors, will be able to navigate it easily.

Meanwhile, modern browsers capable of viewing more-robust sites will see the enhanced version B. The capabilities of yet other (somewhat older) browsers might fall somewhere in between; so, too, could the way they display the page. 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.

More Examples
Take an early peek at Chapter 12 if you’re interested in seeing how the principle of progressive enhancement helps you build a website that adapts its layout based on a device’s screen size and browser capabilities. It can look great on mobile, desktop, and beyond.

Or see Chapter 14 for how older browsers can display simplified designs while modern browsers display ones enhanced with CSS3 effects.

Elsewhere in the book, you’ll learn other techniques that allow you to build progressively enhanced webpages.
Is This Book for You?

This book assumes no prior knowledge of building websites. 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 many 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 the new elements in HTML5, several CSS3 effects, responsive web design, and various best practices.

What this book will teach you

The chapters are organized like so:

- Chapters 1 through 6 and 15 through 18 cover the principles of creating HTML pages and most of the HTML elements at your disposal, with clear examples demonstrating how and when 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 pre-written 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.

Covered topics include the following:

- 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 HTML content, CSS presentation, and JavaScript behavior—a key aspect of progressive enhancement.
- Structuring your content in a meaningful way by using HTML elements that have been around for years as well as ones that are new in HTML5.
- Linking from one webpage to another, or from one part of a page to another part.
- Adding images to your pages and optimizing them for the web. This includes creating images targeted for Apple’s Retina display and other high-pixel-density screens.
- Improving your site’s accessibility with ARIA (Accessible Rich Internet Applications) landmark roles and other good coding practices.
- Styling text (size, color, bold, italics, and more) and adding background colors and images.
- Implementing a multi-column webpage layout.
- Building a responsive webpage. That is, a page that shrinks or expands to fit your visitor’s screen and with a layout that adapts in other ways as you wish. The result is a page that’s appropriate for mobile phones, tablets, laptops, desktop computers, and other web-enabled devices.
Some of the topics not covered include the following:

- **The HTML5** `details`, `summary`, `menu`, `command`, `output`, and `keygen` elements. The W3C has included some of these on their list of features that might not make the cut when HTML5 is finalized in 2014. The others are used infrequently at best.
- **The HTML5 canvas** element, which allows you to draw graphics, create games, and more. Also, Scalable Vector Graphics (SVG). Both are mentioned briefly in Chapter 17, with links to more information.
- **The HTML5 APIs** and other advanced features that require JavaScript knowledge or are otherwise not directly related to the new HTML5 elements.
- **CSS3 transforms**, **animations**, and **transitions**. See www.htmlcssvqs.com/resources/ for links to learn more.
- **CSS3’s new layout methods**, such as FlexBox, Grid, and more. They are poised to change the way we lay out pages once the specs shake out and browser support is stronger. See Zoe Mickley Gillenwater’s presentation at www.slideshare.net/zomigi/css3-layout, or see Peter Gasston’s article at www.netmagazine.com/features/pros-guide-css-layouts.

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

### What this book won’t teach you

Alas, with so many developments in the world of HTML and CSS in recent years, we had to leave out some topics. With a couple of exceptions, we stuck to omitting items that you would likely have fewer occasions to use, are still subject to change, lack widespread browser support, require JavaScript knowledge, or are advanced subjects.
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 webpage 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 similar in any of the latest versions of Chrome, Internet Explorer, Opera, or Safari.

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

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 provided as well.

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.

```html
...  
<body>  
<header class="masthead" role="banner">  
...  
<nav role="navigation">  
<ul class="nav-main">  
<li><a href="/" class="current-page">Home</a></li>  
<li><a href="/about/">About</a></li>  
<li><a href="/contact/">Contact</a></li>  
</ul>  
</nav>  
...  
</header>  
...</body>  
</html>
```
Conventions used in this book

The book uses the following conventions:

- Text that is a placeholder for a value you would create yourself is italicized. Most placeholders appear in the step-by-step instructions. For example, “Type padding: \(x\);, where \(x\) is the amount of desired space to be added.

- Code that you should actually type or that represents HTML or CSS code appears in this font.

- An arrow (\(\rightarrow\)) in a code figure indicates a continuation of the previous line—the line has been wrapped to fit in the book’s column. 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.

- Modern browsers collectively refers to the versions of browsers with solid support for the latest HTML5 and CSS3 features. Generally, this includes recent versions of the browsers discussed in the “Web Browsers” section of this introduction, but not IE8.

- Whenever a plus sign (+) follows a browser version number, it means “the version listed plus subsequent versions.” For instance, IE8+ refers to Internet Explorer 8 and all versions after it.
Companion Website

The book’s companion website 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), a list of errata, and more.

The URLs for some of the key pages on the site follow:
- Home page:
  www.htmlcssvqs.com
- Code examples:
  www.htmlcssvqs.com/8ed/examples/

You can view the code examples directly from the site 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 website.

Please feel free to use the code as you please, modifying it as needed for your own projects.

I hope you find the site helpful!
Unless a site is heavy on videos or photo galleries, most content on webpages 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 a bonus. If not, just override the default formatting with your own CSS.
Adding a Paragraph

HTML does not recognize the returns or other extra whitespace that you enter in your text editor. To start a new paragraph in your webpage, you use the \texttt{p} element (\texttt{A} and \texttt{B}).

To create a new paragraph:

1. Type \texttt{<p>}. 
2. Type the contents of the new paragraph. 
3. Type \texttt{</p>} to end the paragraph.

\textbf{TIP} You can use styles to format paragraphs (and other page text) with a particular font, size, or color (and more). For details, consult Chapter 10.

\textbf{TIP} To control the amount of space between lines within a paragraph, consult “Setting the Line Height” in Chapter 10. To control the amount of space before or after a paragraph, consult “Setting the Margins Around an Element” or “Adding Padding Around an Element,” both of which are in Chapter 11.

\textbf{TIP} You can justify paragraph text or align it to the left, right, or center with CSS (see “Aligning Text” in Chapter 10).

Unsurprisingly, \texttt{p} is one of the most frequently used HTML elements. (Note: In practice, I would wrap an \texttt{article} around this particular content. I omitted it to make the example generic and to avoid giving the impression that \texttt{p} elements must always be nested in an \texttt{article}.)

... 
\begin{verbatim}
<body>

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

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

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

<h2 lang="es">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>

</body>
</html>
\end{verbatim}

Here you see the typical default rendering of paragraphs. By default, browsers provide vertical space between headings and paragraphs, and between paragraphs themselves. As with all content elements, you have full control over the formatting with CSS.
Specifying Fine Print

The **small** element represents side comments such as fine print, which, according to the HTML5 spec, “typically features disclaimers, caveats, legal restrictions, or copyrights. Small print is also sometimes used for attribution or for satisfying licensing requirements.”

The **small** element 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’s appropriate for 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** The **small** element 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.
Marking Important and Emphasized Text

The `strong` element denotes important text, whereas `em` represents stress emphasis. You can use them individually or together, as your content requires (A and B).

**To mark important text:**
1. Type `<strong>`.
2. Type the text that you want to mark as important.
3. Type `</strong>`.

**To emphasize text:**
1. Type `<em>`.
2. Type the text that you want to emphasize.
3. Type `</em>`.

**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** Just as when you emphasize words in speech, where you place `em` in a sentence affects its meaning. For example, `<p><em>Run</em> over here.</p>` and `<p>Run over `<em>`here`</em>.</p>` convey different messages.

**Tip** 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 in another `em`. For example, “due by April 12th” is marked as more important semantically than the other `strong` text in this sentence: `<p><strong>Remember that entries are</strong> due by March 12th</p>`.

**A** The first sentence has both `strong` and `em`, whereas the second has `em` only.

```
... 
<body>
<p><strong>Warning: Do not approach the zombies under any circumstances</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.

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

**Tip** You can style any text as bold or italic with CSS, as well as negate the browser’s default styling of elements like `strong` and `em`. For details, consult “Creating Italics” and “Applying Bold Formatting” in Chapter 10.

**Tip** If you had experience with HTML before HTML5, you may know that at that time `strong` represented text with stronger emphasis than `em` text. In HTML5, however, `em` is the only element that indicates emphasis, and `strong` has shifted to importance.
The b and i Elements: Redefined in HTML5

HTML5 focuses on semantics, not on an element’s presentation. The b and i elements are hold-overs from the earliest days of HTML, when they were used to make text bold or italic (CSS didn’t exist yet). They 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, “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 to address cases like these (and further muddy the waters), HTML5 takes a practical stance by trying to make do with what is available: em for all levels of stress emphasis, strong for importance, and b and i for the through-the-cracks cases. 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 or short span of transliterated prose 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>We rode the <i>Orient Express</i>.</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.
Creating a Figure

No doubt you’ve seen figures in printed newspapers, magazines, reports, and more. Typically, figures are referenced from the main text on a page (like a news story). This very book has them on most pages.

Prior to HTML5, there wasn’t an element designed for this use, so developers cobbled together solutions on their own. This often involved the less-than-ideal, non-semantic `div` element. HTML5 has changed that with `figure` and `figcaption` (A and B). A `figure` element may contain a chart, a photo, a graph, an illustration, a code segment, or similar self-contained content.

You may refer to a `figure` from other content on your page (as shown in A and B), but it isn’t required. 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.

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). (Note: `figures` aren’t required to be in an `article`, but it’s probably suitable in most cases.)
Typically, figure is part of the content that refers to it, but it could also live elsewhere on the page or on another page, such as in an appendix.

The figure element may include multiple pieces of content. For instance, as could include two charts: one for revenue and another for profits. You can even nest one figure inside another one. Keep in mind, though, that regardless of how much content a figure has, only one figcaption is allowed per figure.

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

See “Quoting Text” to learn how to use figure with a blockquote element.

You can’t use the figcaption element unless it’s in a figure with other content.

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.

Modern browsers apply left and right margins of 40px to a figure by default. You can change that with the margin-left and margin-right CSS properties. For example, margin-left: 0; would make the figure flush left. Also, you can make the text containing a figure wrap around it with figure { float: left; } (so the text will wrap around the right side) or figure { float: right; } (so the text will wrap around the left side). You may need to set a width to the figure as well so it doesn’t occupy too much horizontal real estate. CSS coverage begins in Chapter 7, and the float and width properties are demonstrated in Chapter 11.
**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 for what you are quoting from it.

**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):

```
<q lang="en-us">The buck stops here.</q>
```

In addition to instances like that, sites have often used `cite` for the name of people 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 they find the alternatives that HTML5 provides unacceptable (namely, the `span` and `b` elements). Jeremy Keith made the case vociferously in http://24ways.org/2009/incite-a-riot/.
There are two special elements for marking text quoted from a source. The blockquote element represents a standalone quote (generally a longer one, but not necessarily) (A and B) and displays on its own line by default C. Meanwhile, the q element is for short quotes, like those within a sentence D.

C Browsers typically indent blockquote text by default, and don’t display the cite attribute 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.

D Here we see two q examples. 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).

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.

B If you’d like to provide attribution, it must be outside the blockquote. You could place the attribution in a p, but the most explicit way to associate a quote with its source is with a figure andfigcaption, as shown (see “Creating a Figure”).

Quoting Text

... <body>

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

<blockquote cite=http://www. → marktwainbooks.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 in the summertime.</p>

</body>
</html>

... 

<figure>

<blockquote>
I want all my senses engaged. Let → me absorb the world’s variety and → uniqueness.
</blockquote>

<figcaption>— Maya Angelou</figcaption>

</figure>

...
Browsers are supposed to enclose q element text in language-specific quotation marks automatically, but the results are mixed. 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 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>`.
If your `blockquote` contains only a single paragraph or phrase, you don't have to enclose it in a `p` within the `blockquote`.

You can use the optional `cite` attribute on `blockquote` and `q` to provide a URL to the source you are quoting. Although historically browsers haven't displayed the `cite` attribute's URL, in theory it can be handy for search engines or other automated tools that gather quotes and their references. If you would like visitors to have access to it, you could repeat the URL in a link (via the `a` element) in your content. Less effectively, you could expose `cite`'s value via JavaScript (search online for sample code).

The `q` element is invalid for a quote that extends beyond one paragraph. Instead, use `blockquote`.

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. In that case, simply type quotation marks around the word.

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!` to everyone she passed.`</q></p>`. Nested `q` elements should display the appropriate quotation marks automatically—for example, in English the outer quotes should be double and the inner ones should be single. Since outer and inner quotations are treated differently in languages, add the `lang` attribute to `q` as needed. Unfortunately, browsers are inconsistent with nested `q` elements much like they are for non-nested ones.

Because of cross-browser issues with `q`, many (most likely the majority of) coders choose to simply type the desired quotation marks or use character entities instead of the `q` element.
Specifying Time

You can mark up a time, date, or duration with the `<time>` element, which is new in HTML5. It allows you to represent this information in a variety of ways (A and B).

The text content inside `<time>` (that is, `<time>text</time>`) appears on the screen for us humans (B and D), whereas the value of the optional `datetime` attribute is intended for the machines among us. It requires a specific format; the sidebar “Understanding the Valid Time Format” covers the basics, and the first tip explains another case when the format is required.

To specify a time, date, or duration:

1. Type `<time>` to begin a `<time>` element.
2. If desired, type `datetime="time"` where `time` is in the approved machine-readable format (see the sidebar) that represents the text you’ll enter in step 4.
3. Type `>` to complete the start tag.
4. Type the text that reflects the time, the date, or the duration that you want to display in the browser. (See the first tip if you did not include `datetime` in step 2.)
5. Type `</time>`.

---

As shown in the first example, the simplest form of the `<time>` element lacks a `datetime` attribute. But it does provide the times and date in the valid machine-readable format as required when `datetime` is omitted. The remaining examples show that the text between the `<time>` tags doesn’t need to match the valid format when `datetime` is present (the last example shows one case of each approach).

...  
<body>  
<p>The train arrives at `<time>08:45</time>` and `<time>16:20</time>` on `<time>2017-03-19</time>`.</p>  
<p>They made their dinner reservation for tonight at `<time datetime="2013-11-20T18:30:00">6:30</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>The film festival is `<time datetime="2014-07-13">July 13</time>-<time datetime="2014-07-16">16</time>`.</p>  
<!-- Example with no year -->  
<p>Her birthday is `<time datetime="03-29">March 29th</time>`.</p>  
<!-- Example of durations -->  
<p>The meeting lasted `<time>2h 41m 3s</time>` instead of the scheduled `<time datetime="2h 30m">two hours and thirty minutes</time>`.</p>  
</body>  
</html>
If you omit the `datetime` attribute, the text content inside `time` must follow the machine-friendly format rather than being “free-form.” In other words, the first example in A 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>April 20th, 2015</time>.</p>` because the `time` text doesn’t follow the format in any of the three instances. However, when you do include `datetime`, you’re free to represent the date, time, or duration in the text content as you wish, as seen in the other examples in A.

The `datetime` attribute doesn’t do anything on its own but could be used for syncing dates and times between web applications and the like (for example, think of a calendar application). That’s why it requires a standard, machine-readable format; it allows these programs to share information by speaking the same “language.”

You may not nest a `time` element inside another one or place any other elements (just text) in a `time` element that lacks a `datetime` attribute.

The `time` element allowed an optional attribute named `pubdate` in an earlier iteration of HTML5 (remember that the language is still evolving). However, `pubdate` is no longer part of HTML5. I mention this in case you come across it in an older tutorial or book (such as the seventh edition of this book!) and wonder if you should use it (you shouldn’t).

Only the `time` text displays in browsers, not the `datetime` value.

This shows how you might include a date for a blog post or news article. As is required for all cases of `datetime`, its value represents the text content in a machine-readable format.

```html
...<body>
<article>
  <h1>Cheetah and Gazelle Make Fast Friends</h1>
  <p><time datetime="2014-10-15">October 15, 2014</time></p>
  ... [article content] ...
</article>
</body>
</html>
```

As expected, the date is below the heading.
Understanding the Valid Time Format

The `datetime` attribute—or a `time` element without `datetime`—must provide the desired date and/or time in a specific machine-readable format. I’ve simplified it below:

`YYYY-MM-DDThh:mm:ss`

For example (local time):

`1985-11-03T17:19:10`

This means “November 3, 1985, at 10 seconds after 5:19 p.m. local time.” The hours portion uses a 24-hour clock, hence `17` instead of `05` for 5 p.m. 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.)

The format is a little different when representing a duration. There are a couple of syntax options, but this is the simplest to follow:

`nh nm ns`

(Where `n` is the number of hours, minutes, and seconds, respectively.)

The last example in A shows it in action.

Global Dates and Times and Time Zone Offsets

If you’d like, you can represent your dates and times in a global context instead of a local one. (Or simply the time by omitting the date.) Add a `Z` at the end to mark the time zone as UTC (Coordinated Universal Time), the primary global time standard. (See https://en.wikipedia.org/wiki/Coordinated_Universal_Time.)

For example (global date and time in UTC):

`1985-11-03T17:19:10Z`

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

For example (global date and time with offset from UTC):

`1985-11-03T17:19:10-03:30`

This means “November 3, 1985, at 10 seconds after 5:19 p.m. Newfoundland Standard Time (NST),” because NST is 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.

Just as a reminder, if you do include `datetime`, it doesn’t require the full complement of information I just described, as the examples in A show.
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 represented by 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 represented by the abbreviation.

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 parenthetical abbreviation expansion is the most explicit way to describe an abbreviation, making it available to the widest set of visitors. For instance, users on touchscreen 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.

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

continues on next page
TIP As a visual cue to sighted users, Firefox and Opera display abbr with a dotted bottom border if it has a title. If you’d like to replicate that effect in other browsers, 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 regardless of whether the abbr is styled with a border.

TIP If you don’t see the dotted bottom border under your abbr, try adjusting the parent element’s CSS line-height property (see Chapter 10).

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

C Browsers display the title of abbreviations as a tool tip when you hover the pointer over text marked up with abbr. (This figure also demonstrates an example of a browser—Chrome in this case—that doesn’t style abbreviations with a title any differently than regular text by default.)
Defining a Term

In the print world, it's customary to differentiate a term visually when you define it. Typically, this is done with italics; subsequent uses of the term are not italicized.

In HTML, when you define a term, you differentiate it semantically with the `dfn` element. You wrap its tags only around the term you're defining, not around the definition. And just as in print convention, subsequent uses of the term are not marked with `dfn`, because you aren't defining them again. (HTML refers to the point where you define a term as the “defining instance of a term.”)

To mark the defining instance of a term:

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

You can also use `dfn` in a description list (the `dl` element). See “Creating Description Lists” in Chapter 15.

Use `dfn` only when defining a term, not simply because you want to italicize text. CSS allows you to style any text in italics (see “Creating Italics” in Chapter 10).

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

If you use the optional `title` attribute on a `dfn`, it should have the same value as the `dfn` term. If you nest a single `abbr` in `dfn` and the `dfn` has no text of its own, the optional `title` should be on the `abbr` only, as in the previous tip.

Proximity of a Term and Its Definition

The location of a term marked with `dfn` relative to the location of its definition is important. 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.” Simplified, this means that the `dfn` and its definition should be near each other, which makes sense. This is the case in both A and the example given in the fourth tip; the `dfn` and its definition are in the same paragraph.
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. HTML includes elements for defining both kinds of text. Common uses for superscripts include marking trademark symbols, exponents, and footnotes. Subscripts are common in chemical notation.

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.

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

Superscripts are the ideal way to mark up certain foreign-language abbreviations—such as Mlle for Mademoiselle in French or 3a for tercera in Spanish—or to mark up numerics like 2nd and 5th.

One proper use of subscripts is for writing out chemical molecules, such as H₂O. For example, `<p>I'm parched. Could I please have a glass of H₂O?</p>`.

Super- and subscripted characters gently spoil the even spacing between lines. In the same line, 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. CSS comes to the rescue, though; see the sidebar to learn how to fix this.

One use of the `<sup>` element is to indicate footnotes. I placed the footnotes in a footer within the `<article>` rather than in 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>... Actually, Pablo Casals' real name was Pau Casals, Pau being the Catalan equivalent of Pablo. 
    &lt;a href="#footnote-1" title="Read footnote 1"&gt;&lt;sup&gt;1&lt;/sup&gt;&lt;/a&gt;.&lt;/p&gt;

  &lt;p&gt;... Pau Casals is remembered in this country for his impassioned speech against nuclear proliferation at the United Nations. 
    &lt;a href="#footnote-2" title="Read footnote 2"&gt;&lt;sup&gt;2&lt;/sup&gt;&lt;/a&gt; ...
  &lt;/p&gt;

  &lt;footer&gt;
    &lt;p id="footnote-1"&gt;&lt;sup&gt;1&lt;/sup&gt;It means Paul in English.&lt;/p&gt;
    &lt;p id="footnote-2"&gt;&lt;sup&gt;2&lt;/sup&gt;In 1963, I believe.&lt;/p&gt;
  &lt;/footer&gt;
&lt;/article&gt;
</body>
```
Fixing the Spacing Between Lines When Using sub or sup

With a little bit of CSS, you can fix the line height discrepancies caused by the sub and sup elements. The code below 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, which 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 display of elements 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.
Adding Author Contact Information

You might think the **address** element is for marking up a postal or street address, but it isn’t (except for one circumstance; see the first tip). 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 responsible for either a part of a webpage (such as a news article, product review, or report) or a whole page (A and B). Which of those is true depends on where **address** appears. The first step describes each scenario.

### 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** (see the first instance in A). 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 (see the second instance in A).

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 only contact information. Although the background information about Tracey Wong is also in the **article**’s **footer**, it’s outside the **address** element.

```html
... <body> <main role="main"> <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> </main> <!-- the page's footer with address information for the whole page --> <footer role="contentinfo"> <p><small>© 2014 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>
```
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. The example in “Creating a Line Break” shows one way to mark up a postal or street address.

The address element pertains to the nearest article it is contained in, 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, like the second instance of address in (A).

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.

The address element may contain only author contact information, 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.

See Chapter 3 to learn more about the article and footer elements.
Noting Edits and Inaccurate Text

Sometimes you may want to indicate content edits that have occurred since the previous version of your page. 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 newly inserted text:
1. Type `<ins>`.
2. Type the new content.
3. Type `</ins>`.

To mark 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.

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

---

...<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 solar-powered tablets</del></li>
<li><ins>1 bicycle</ins></li>
</ul>
</body>
Both `<del>` and `<ins>` are rare in that they can surround both phrasing content (“inline” content, in pre-HTML5 parlance) and blocks of content like entire paragraphs or lists, as shown here.

```html
...  
<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 solar-powered tablets</del></li>
    <li><ins>1 bicycle</ins></li>
  </ul>
</del>

</body>
</html>
```

Just as before, browsers indicate which content has been deleted or inserted.
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 with `del` and `ins`, 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.

**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 that was 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.

---

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 around text within list items (`li` elements), but you cannot use it around a whole paragraph or other “block-level” element like you can with `del` and `ins`.

```html
<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>
```

The `s` element renders as a strikethrough by default in browsers.
Text marked with the ins element is generally underlined by default. 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 CSS to change how inserted passages (or links) are displayed (see Chapter 10).

Text marked with the del element is generally struck out. Why not just erase it and be done with it? It depends on whether you think it’s important to indicate what’s been removed. Striking out content 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 historically been lacking.)

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).

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 or file names, the `code` element is for you (A and B).

The examples show `code` used in a sentence. 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).

**Tip** See “A Webpage’s Text Content” in Chapter 1 regarding character entities A.

---

A The `code` element indicates that the text is code or a file name. If your code needs to display `<` or `>` signs, use the `&lt;` and `&gt;` character entities, respectively (see the last tip). Here, the second instance of `code` demonstrates this. If you were to use `<` and `>`, the browser would treat your code as an HTML element, not as text to display.

```html
... 
<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: 1.25em; }<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.

The kbd Element
Use kbd to mark up user input instructions.

```html
<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.

```html
<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.

```html
<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 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 of text according to the width of the browser 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 paste 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).

```html
...<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 the text is 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. (In the vast majority of cases, those versions are too old to worry about.)

```css
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. Whitespace can be crucial to the semantics of 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. 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: &lt; and &gt; respectively (see “Marking Up Code” for an example). Otherwise the browser may try to display those elements.

**Tip** 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 then styling the way it looks 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.
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.

HTML5 replicates this with the new `<mark>` element. Think of `<mark>` as a semantic version of a highlighter pen. In other words, what’s important is that you’re noting certain words; how they appear isn’t important. 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). This is akin to the real-world task of highlighting a textbook or contract.

- 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.

```html
...<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 halfway through.</p>
</blockquote>
<p>I thought he’d told me fifty. 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).
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 stress emphasis) or `strong` (which represents importance). Both are covered earlier in this chapter.

**Tip** Since `mark` is new in HTML5, older browsers don’t render a background color by default. 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 to contain it, use the `span` element (covered later in this chapter), perhaps with a class assigned to it, and style it with CSS.
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.

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 (see the second tip).

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.

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 shown display the same way (B).

```
...<br />
</body>
</html>
```

B Each `<br>` element forces the subsequent content to a new line. Without them, the entire address would display on one line, unless the browser were narrow enough to force wrapping.
To insert a line break:

Type `<br />` (or `<br>`) 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 />` or `<br>` is perfectly valid in HTML5.

**Tip** CSS allows you to 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 using 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).

The span element 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. (The lang attribute allows you to declare the language of the element’s text.)

```html
... 
<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>
```
To add a span:

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 `</span>`.

**Tip** A span doesn’t have default formatting, but just as with other HTML elements, you can apply your own with CSS.

**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.

The `span` element has no default styling.

La Casa Milà

Gaudi’s work was essentially useful. La Casa Milà is an apartment building and real people live there.
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 (sounds a little befuddling, I know). 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:

```html
<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.
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` named `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:

```html
<p>They liked to say, "FriendlyFleas andFireFlies FriendlyFleas andFireFlies FriendlyFleasand FireFlies" as fast as they could over and over.</p>

<p>His favorite site is this is a really 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.

---

A Like links, `u` elements are underlined by default, which can cause confusion unless you change one or both with CSS.
The ruby, rp, and rt elements

A ruby annotation is a convention in East Asian languages, such as Chinese and Japanese, and is 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 supporting and non-supporting browsers. The area for ruby text is highlighted:

```xml
<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:

```xml
<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 because it ignores the rp elements.

C A non-supporting browser displays the rt content in parentheses in the normal flow of content.

D Now, the ruby markup for “Beijing” as seen in a supporting browser.
You can see how important the parentheses are for browsers that don’t support Ruby. Without them, the base and ruby text would run together, clouding the message.

> At the time of this writing, Firefox and Opera lack basic Ruby support (all the more reason to use `rp` in your markup). The Firefox add-on HTML Ruby (https://addons.mozilla.org/en-US/firefox/addon/html-ruby/) provides support for Firefox in the meantime.

> 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 that the base directionality of your content is right-to-left and that 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>`. The **bdo** element 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”).

```html
<ul>
  <li>User <bdi>jcranmer</bdi>: ➝ 12 posts.</li>
  <li>User <bdi>hober</bdi>: ➝ 5 posts.</li>
  <li>User <bdi>نايإ</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. At first glance, it seems very similar to the *progress* element, covered next, which is for indicating “the completion progress of a task” (to quote the spec).

In contrast, you can use *meter* to indicate a fractional value or a measurement within a known range. In plain language, it’s the type of gauge you use for the likes of voting results (for example, “30% Smith, 37% Garcia, 33% Hawkins”), the number of tickets sold (for example, “811 out of 850”), a numerical test grade (for example, “91 out of 100”), and disk usage (for example, “74 GB out of 256 GB”).

HTML5 suggests (but doesn’t require) that 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). Firefox, one of the browsers that supports *meter* so far, does just that. 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.

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

```html
<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">21% worn</meter></p>
<p>Miles walked during half-marathon: <meter min="0" max="13.1" value="5.5" title="Miles">4.5</meter></p>
```

A browser, like Firefox, that supports *meter* displays the gauge automatically, coloring it based on the attribute values. It doesn’t display the text in between `<meter>` and `</meter>`. As seen in the last example, if you include *title* text, it displays when you hover over the meter.

IE9 doesn’t support *meter*, so instead of a colored bar, it displays the text content inside the *meter* element. You can change the look with CSS.
The `meter` element 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. As is usual with `title` text, browsers display it as a tooltip.

**Tip** `meter` supports several attributes. The `value` attribute is the only one that’s required. The `min` and `max` attributes 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. The number assigned to `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, browser support of `meter` is still evolving: It’s not supported by Internet Explorer, mobile Safari (iOS devices), or Android’s browser. This partially explains why you don’t yet see it much in the wild. Feel free to use it, but just understand that these browsers will render the `meter` text rather than the visual gauge by default. See http://caniuse.com/#feat=progressmeter for the latest browser support.

**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** The `meter` element is not for marking up general measurements—such as height, weight, distance, or circumference—that have no known range. For example, you cannot use it for the following because the number of miles walked isn’t gauged against a range:

```html
<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. As stated earlier, it indicates the completion progress of a task. Think of 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. 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, even though it’s not required.

Here’s an example:

```html
<p>Please wait while we save your data.</p>
<p>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 code it that way in the HTML initially; for example, `<progress max="100" value="37">37% saved</progress>`. Of course, non-supporting browsers would display it similarly to the Firefox experience.

A browser, like Firefox, that supports `progress` displays the progress bar automatically, coloring it 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 bar indicates no progress.

IE9 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.

The `progress` bar in Firefox when the `value` attribute is set to 37 programmatically with JavaScript (or directly in the HTML), assuming `max="100"`. The blue area reflects the amount of progress.
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. The value attribute specifies 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.

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

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

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

```javascript
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 to 37, and the appearance of the progress element would change accordingly.

The progress element is supported by the most current version of all desktop browsers as of this writing. IE9 and prior, mobile Safari, and Android browsers don't support it. See http://caniuse.com/#feat=progressmeter for the latest support information.

The style of the progress bar that each supporting browser displays may vary, though you can style it yourself to some extent with CSS.
Symbols
/* */, using for CSS comments, 172
&amp;gt; character entity, using, 112
&amp;lt; character entity, using, 112
* (asterisk) wildcard, using with selectors, 207
{} (curly braces)
  using with CSS, 171
[] (square brackets)
  using with attribute selectors, 205, 222
  using with selectors, 205
&lt;-- and --&gt;, using for HTML comments, 86
&lt; and &gt; signs, displaying in code, 112

A
a element, 158
  using, 7, 21–22, 158
abbr element, 101
absolute positioning of elements, 302–303
absolute URLs, 16
  vs. relative URLs, 19
accept attribute, 412
accept-charset attribute, 523
accessibility, 23
  :focus pseudo-class, 385
  color contrast, 256
  form labels, 425
  importance of headings (h1–h6), 50–51
  improving with ARIA, 78–81
  keyboard navigation, 158
  progressive enhancement, xx
  providing with video, 462
  screen readers. See screen readers
accesskey attribute, 520
acronym element, elimination of, 102
action attribute, 414
address element, 106–107
Adobe Photoshop. See Photoshop
:after pseudo-element, 384–386
aligning
  elements vertically, 306–307
  text, 259
alpha transparency
  defining with RGBA and HSLA, 184–188
  explained, 138–139
  saving images with, 144
  showing in IE versions, 188
alt attribute, 145, 442
  using, 21
  using effectively, 148
  using for missing images, 147–148
alternative text, offering, 147–148
analytics, 513
ancestor element, 212–213
anchors, 164–165
animated images, saving, 139
Apache development server, 18
area element, 521
ARIA (Accessible Rich Internet Applications), 79–81
aria attribute, 422
Arial, showing on Windows, 235
article element, 60
  with address element, 61
  examples, 62
  with footer element, 61
  multiple, 61
  nesting, 61
  nesting section element in, 64
  vs. section, 271
  with section elements, 61
  using, 20–21, 60–61
ASCII characters, 12
aside element, 65–67
  examples, 68–69
  vs. figure, 93
asterisk (*) wildcard, using with selectors, 207
async attribute, 528
attribute selector examples, 224–225
attributes
  accepting values, 9–10
  accesskey, 520
  alt, 21
  aria, 520
class, 520
draggable, 520
attributes (continued)

dropzone, 520
of elements, 9
hidden, 520
href, 10
id, 520
lang, 520
numbers for values, 10
role, 520
selecting elements based on, 222–225
selector options, 223
spellcheck, 520
square brackets ([[]), 222
src, 10
tabindex, 520
title, 520
audio attributes, 465–469
audio element, 464
audio file formats, 463–465
audio files with controls, adding to pages, 464–465
audio sources, providing with fallback, 468–469
author contact info, adding, 106–107
autocomplete attribute, 424
autofocus attribute, 423
autoplay attribute, 454, 466

B

b element, 91
redefined in HTML5, 91
vs. strong element, 90
background color, changing, 250–251. See also colors
background images, 252–255. See also images
background properties, 255–256
background-clip property, 256
background-origin property, 256
backgrounds, 250–256
applying multiple, 373–375
fallback, 378
gradient, 376–381
background-size property, 256
banner landmark role, 79–80
Barcelona's Architect example
adding indents, 258
aligning text, 259
alternate fonts, 233–235
bold formatting, 238–239
changing text case, 260
controlling spacing, 257
decorating text, 262–263
default page rendering, 230
font family, 232
font size, 240–244
HTML classes, 231
italics, 236–237
line height, 245
setting background, 250–256
setting color, 248–249
setting font values, 246–247
setting whitespace properties, 264
using small caps, 261
viewing, 230
base element, 521
BBEdit text editor, 31
bdi element, 125–127
bdo element, 125–127
:before pseudo-element, 384–386
block-level elements, explained, 24–25
block-level links, 161–162
blockquote element, 95–97
nesting, 97
body element, 44
adding to webpages, 45–46
basic usage, 4, 6–7
components, 20
bold formatting
applying, 238–239
applying with web fonts, 352–353
faux bold, 239
removing, 238
bold italic
applying with web fonts, 354-355
faux bold italic, 355
BOM resource, 34
Boolean attribute
explained, 10
using with videos, 455
border color, setting, 288
border properties, setting multiple, 289
border styles, 288, 291
border width, setting, 288
border-image property, 290
border-radius property, 364–367
borders, setting, 290–291
box model
border, 276
components, 276
explained, 276
height, 277
margin, 276
padding, 276–277, 286–287
width, 277
box-sizing property
applying to all elements, 283
effect on box model, width, and height, 277
usefulness with responsive webpages, 318, 326
box-shadow property, 370–372
br element, 118–119
vs. wbr, 123
browser developer tools
  Chrome, 503
  Firefox, 503
  Internet Explorer, 503
  Opera, 503
  Safari, 503
browser support resources, 499
browsers. See also polyfills
default display of webpages, 24–25
explained and version numbers, xvii
inline, 24–25
modern browsers, xxv
  -moz- prefix, 364
  -ms- prefix, 364
  -o- prefix, 364
  obtaining for testing, 500
  prefixes, 364
  support for gradients, 381
  testing sites in, 500
  viewing pages in, 38–39
  VMs (virtual machines), 500
  -webkit- prefix, 364
bulleted lists, creating, 391, 393
button element, 442
  using with forms, 442–443
C
canvas element, 475
  using with video, 475
capitalize value, limitations of, 260
caption element, 478
captions, creating for figures, 92–93
challenge attribute, 526
character encoding, specifying, 12
  accented characters, 12
  dir attribute, 126
  left-to-right, 125
  right-to-left, 125
charset attribute, 45
checkboxes, creating for forms, 434–435
checked attribute, 433
checking for errors. See also debugging techniques
  CSS (Cascading Style Sheets), 508–509
  general, 504–505
  HTML (Hypertext Markup Language), 506–507
child element
  explained, 11, 212
  first and last, 216–217
Chrome
developer tool, 503
  refreshing pages in, 39
  undocking Developer Tools, 39
  testing sites in, 500
Chrome’s cache, disabling, 39
circles, creating using border-radius, 367
citations, indicating, 94
cite attribute, 94, 95, 110
  using with blockquote, 97
cite element, 87, 94–95, 522
  using for names, 94
class attributes, 82
  applying, 82–83
  implementing microformats, 83
  naming, 83
class names, assigning to elements, 82
class selectors. See also pseudo-classes
  vs. ID selectors, 211
  multiple classes on one element, 177, 210
  using with inline styles, 197
clearfix method, using with float property, 299
clearing floats, 297–300
“click here” labels, avoiding, 162
Cloud.typography web font service, 339
Coda text editor, 31
code
  displaying < and > signs, 112
  marking up, 112
  validating, 496–497
code editor, funny characters in, 47
code element, explained, 87, 112
codec, explained, 451
col element, 522
colgroup element, 522
colors. See also background color
  per image formats, 136
  CSS color options, 182–188
  declared with hexadecimal, 183
  declared with keywords, 182
  declared with HSL and HSLA, 186–188
  declared with RGB and RGBA, 183–185
  setting for text, 248–249
  specifying for borders, 288
colspan attribute, 482
“commenting out” declarations, 173
comments
  */ and */ for CSS, 172
  <!-- and --> for HTML, 86
  adding to HTML, 85–86
  adding to CSS, 172–173
complementary landmark role, 79–80
conditional comments, using with responsive pages, 333
consistency, checking HTML for, 496
contact info, adding, 106–107
containers, creating, 73–75
content, adding to webpages, 6–7
content attribute, 527
ccontentedible attribute, 520
contentinfo landmark role, 79–80
controls attribute, 454, 464
coords attribute, 521
corners of elements
elliptical, 367
rounding, 365–367
Creative Commons licenses, 140
CSS (Cascading Style Sheets). See also CSS3
features; inheritance; selectors; style sheets
avoiding @import, 199
backgrounds, See backgrounds
benefits, 170
browser compatibility, See browser support
resources
default browser styles, 24, 274–275
debugging techniques, 508–509
generated content, 384–386
history and relationship to CSS3, xvi
images and sprites, 387–388
!important, 199
inheritance, 174–176
link element, 199
media attribute, 200
media queries, 219
order of styles, 198–199
polyfills for non-supporting browsers, 363
progressive enhancement, xx–xxi, 362–363
specificity, law of, 178–179
the cascade, 177–179, 198
treatment of class names, 177
validating to locate errors, 496
vendor prefixes, 364
viewing others’ code, 202
CSS Arrow Please! 386
CSS border shorthand property, 146
CSS color keywords, 182–183
.css extension, using with external style sheets, 190
CSS gradients, syntax for, 376
CSS properties
bare numbers, 181
for formatting, 169
hexadecimal, 183
inherit value, 180
for layout, 169
lengths, 180–181
percentages, 180–181
predefined values, 180
for printing, 169
RGB, 183
URLs, 182
CSS validator, using, 497
CSS2, explained, 170
CSS3 colors
color keywords, 182
HSL and HSLA, 186–188
RGBA, 184–185
CSS3 features
animations, 363
background size, 256
classic techniques, 502–503.
See also checking for errors
default attribute, 530
default homepage, specifying, 35
default styles. See also styles
normalizing, 274–275
resetting, 274–275
define attribute, 528
defining terms, 103
del element, 108–109
cite attribute, 110
datetime attribute, 110
vs. s element, 111
text marked with, 111
using with content changes, 110
deleted text, marking, 108
descendant element, defined, 212–213
details element, 523
development server, Apache, 18
dfn element, 103
dir attribute, 125
using with bdi, 125
using with bdo, 126
dirname attribute, 525
disabled attribute, 444
display type, controlling, 278–281
Index 537

div element, 73, 125
adding around whole pages, 73
containing pages in, 77
vs. section element, 63
surrounding content, 74
using in HTML5, 76
dl (description list) element, 404
creating, 404–407
explained, 389, 523
nesting, 407
DOCTYPE declaration
including in webpages, 4–5, 44
document flow, explained, 278
document head, explained, 5, 46
document headings structure, defining, 50
domain, connecting with web host, 514
domain name, getting, 512
double vs. single quotes, 348
download attribute, 520
draggable attribute, 520
DRM (digital rights management), 450
drop shadows, adding to text, 368–369
dropzone attribute, 520
dt element, 389, 404
duration, specifying, 98
E
Edge Web Fonts service, 339
editing webpages, 36
edits, noting, 108–111
elements. See also alphabetical list of HTML elements in Appendix; pseudo-elements;
selecting elements
aligning vertically, 306–307
ancestors, 212
assigning classes to, 82
attributes of, 9
block-level, 24
components, 8
contents, 8
descendants, 212
display type, 278–281
document flow, 278
empty, 8
floating, 295–300
inline, 24
naming with unique IDs, 82
nesting, 11
number available, 22
offsetting in natural flow, 301
parents, 11
phrasing content, 21
positioning absolutely, 302–303
positioning in stacks, 304
relative positioning, 301
rounding corners of, 365–367
selecting based on attributes, 222–225
selecting by class, 208–210
selecting by context, 212–215
selecting by ID, 208–210
selecting by name, 206–207
selecting when first child, 216–217
selecting when last child, 216–217
specifying groups of, 226
visibility, 278–281
void, 8
wrapping text around, 295–296
em (stress emphasis) element, 90
explained, 87, 523
vs. i element, 90
and percentage font sizes, 241–243
using, 21–22, 90
em values for padding and margin, 294
email boxes, creating for forms, 428–431
em element, 523
embedded style sheets, 194–195. See external style sheets; style sheets
benefits, 192
creating, 190–191
importing, 191
linking to, 191–193
loading multiple, 193
media attribute, 200–201, 319–322
naming, 191
saving with .css extension, 190
URLs in, 193
UTF-8 encoding, 190
F
fallback backgrounds, creating, 378
favicons, 155–156
fieldset element, 418
using with forms, 418–421
figcaption element, 92–93
figure element, 92–93
file extensions, using consistently, 32–34
file names
extensions, 14
lowercase, 14, 26
separating words with dashes, 14


file uploads, handling for forms, 439
files
  naming conventions, 37
  organizing, 37, 193, 255, 356
  transferring to server, 515–518
FileZilla FTP client
  downloading, 515
  using, 516–517
Firefox
  developer tool, 503
  testing sites in, 500
Fireworks alternatives, 141
fixed-width page
  explained, 266
  pixels for, 283
Flash fallback, 470–474
Flash playback, troubleshooting, 474
Flash plugin, use of, 13
Flash security settings, changing, 474
Flickr website, 140
float property, 295–300
floats, clearing, 297–300
folder names
  extensions, 14
  lowercase, 14, 26
folders
  dividing into sub-folders, 37
  naming conventions, 37
font alternates, specifying, 233–235
font family, setting, 232
font sizes
  basing on parent element, 244
  basing on root element, 243–244
  em and percentage, 241–244
  pixels, 240
  rem (root em), 243–244
  setting, 240
Font Squirrel, 338, 342–344
font values, setting at once, 246–247
Fontdeck web font service, 339
@font-face rule, using with web fonts, 336–337, 345, 348
fonts. See also font sizes; Google Fonts web font service; styles for text formatting; text; web fonts
  Arial on Windows, 235
  Geneva stacks, 235
  Helvetica on OS X, 235
  shared defaults, 234
  specifying for alphabets, 235
Food Sense site, viewing, 311
footer element, 70–72
footers, creating, 71
for attribute, 425, 528
form attribute, 522
form element, 413
form elements
  disabling, 444–445
  example, 413
fieldset element, 418–421
headings, 421
labels, 422–424
legend element, 418–421
organizing, 418–421
placeholder attribute, 422–424, 426
screen readers, 421
separating, 422
styling, 421
form fields, placing inside labels, 426
form parts, labeling, 425–426
formaction attribute, 522
formenctype attribute, 522
formmethod attribute, 522
formnovalidate attribute, 443, 522
formtarget attribute, 525
forms
  accept attribute, 412
  attributes, 412
  autocomplete attribute, 412
  autofocus attribute, 412
  checkboxes, 434–435
  color, 411
  creating, 413–415
  creating hidden fields, 440
  creating text boxes, 422–424
  data list, 411
  date input type, 411
  disabling validation features, 415
  email boxes, 428–431
  features-enhancement for older browsers, 412
  file uploads, 439
  formnovalidate attribute, 412
  global date and time, 411
  HTML5 inputs and elements, 411
  id naming convention, 426
  improvements in HTML5, 410–412
  letting visitors upload files, 439
  list attribute, 412
  maxlength attribute, 423, 436
  method="get", 415
  method="post", 415
  month, 411
  multiple attribute, 412
  name attributes, 426
  for naming convention, 426
  novalidate attribute, 412
  number input type, 411
  output element, 411
  password boxes, 427
  pattern attribute, 412
Index

organizing webpages with, 51–52
proper use of, 52
using, 50
using all levels of, 51
height, setting for elements, 282–285
height vs. min-height, 284
Helvetica, showing on OS X, 235
hgroup element, removal of, 52
hidden attribute, 520
hidden fields, creating for forms, 440
high, low attributes, 128
highlighting text, 116–117
homepage, specifying default, 35
hr element, 524
href attribute
  contents, 10
  explained, 162
  including for links, 158–159
href values, including on webpages, 6, 158, 192
hreflang attribute, 520
HSL (hue, saturation, light) and HSLA, 186–188
HTML (Hypertext Markup Language), 26
  checking for consistency, 496
  debugging techniques, 506–507
  history and relationship to HTML5, xvi
  indenting, 7
  rendering by browsers, 12, 24–25
  semantics, 20–23
  thinking in, 3
  validating, 497
  viewing others’ code, 40–41
  writing in lowercase, 26
.html and .htm extensions, 32
html element, 44
  basic usage, 4–5
  including in webpages, 44–46
HTML elements. See elements
HTML markup. See markup
HTML pages. See also webpages
  body element, 4, 20
  common page constructs, 53
  components, 44
  DOCTYPE, 4–5, 44, 46
  example, basic, 4
  examples, larger, 60, 230–231, 269–270
  head element, 4, 45
  html element, 4, 44
  indenting code, 7, 44
  semantics, 20–23
  structure, 44–46, 50–52
HTML5
  DOCTYPE, 46–47
  differences with HTML4, xix
  document outline, 52
  empty elements, 9
Index

i element, 90–91
lcoMoon, creating icon fonts, 340
icon fonts, getting, 340. See also fonts; web fonts
icons, adding for websites, 155–156
id attribute, 82–83, 520
ID selectors vs. class selectors, 211
IE (Internet Explorer), older versions of and responsive webpages, 333–334
iframe element, 163, 524
image editors
   choosing, 141
   scaling images with, 154
image formats
   GIF, 134–135, 138
   JPEG, 134–135
   PNG-8, 134–135, 138
   PNG-24, 134–135
   PNG-32, 134–135
   WebP, 135
image maps, use of, 162
image optimization, 142–144
image path, 146
image sizes, 136–137, 139
   finding, 149–151
   height attribute, 150–153
   specifying, 149–151
   width attribute, 150–153
images. See also background images
   adding to pages, 145–146
   animation, 139
   changing alignment, 145
   changing display size of, 152
   combining with sprites, 387–388
   creating for Retina displays and other high-pixel-density displays, 152
   Creative Commons licenses, 140
debbuging problems displaying, 148, 510
dimensions, 136–137, 139
   format colors, 136
   getting, 140
   Google Usage Rights, 140
   height attribute, 150–153
   making flexible, 312–314
   offering alternative text, 147–148
   pixels, 136
   printing, 137
   resizing, 154
   retaining aspect ratio, 152
   saving, 139, 142–144
   scaling proportionally, 313
   scaling with browser, 152
   simulating slow connections, 146
   storing in folders, 146
   SVG (scalable vector graphics), 137
   testing loading time, 146
   transparency, 138
   width attribute, 150–153
img element, 145
   basic usage, 7, 21
   indenting HTML, 7, 44
   indents, adding to text with CSS, 258
index transparency, 138–139
index.html, saving as default page, 35
inheritance, 174–176. See also CSS (Cascading Style Sheets)
inline elements, explained, 24–25
inline styles, 196–198
ins element, 108–111
inserted text, marking, 108
inset shadow, creating, 371
Internet Explorer 8+, testing sites in, 500
Internet Explorer developer tool, 503
iOS Simulator website, 501
iPad, testing pages for, 501
iPhone, testing pages for, 501
ismap attribute, 524
ISP (Internet service provider), using as web host, 513
italics
   applying with web fonts, 350–351
   creating, 236–237
   real vs. faux, 237
   removing, 237
J
JavaScript
   adding embedded scripts, 492
   inline scripts, 491
   loading external scripts, 487–491
   minifying, 489
   Node.js, 486
   organizing files, 489
   overview, 485–486
   performance best practices, 490–491
   resources, 486
   sample, 488–489
   script element, 490, 492
   scripting best practices, 490–491
Index

JavaScript events
  onblur, 493
  onchange, 493
  onclick, 493
  ondblclick, 493
  onfocus, 493
  onkeydown, 493
  onkeypress, 494
  onkeyup, 494
  onload, 494
  onmousedown, 494
  onmousemove, 494
  onmouseout, 494
  onmouseover, 494
  onmouseup, 494
  onreset, 494
  onselect, 494
  onsubmit, 494

JAWS screen reader, 78
Johansson, Roger, 348
JPEG image format, 134–135
JPEGmini website, 144
JW Player Flash fallback solution, 474

K
  kbd element, 113
  kind attribute, 530
  kerning, specifying, 257
  keygen element, 526
  keytype attribute, 526

L
  label attribute, 438
  label element, 425–426
  landmark roles, 79–81
  lang attribute, 44
    using q element with, 95–96
    using with headings, 52
  language subtag lookup tool, 47
  languages, right-to-left, 127
  layout approaches
    fixed-width pages, 266
    responsive webpages, 266–267
  layout grid, making flexible, 315–318
  layout with styles. See also styles
    absolute positioning of elements, 302–303
    aligning elements vertically, 306–307
    borders, 288–291
    box model, 276–277
    browser considerations, 267
    changing cursor, 308
    display type, 278–281
    elements in stacks, 304
    height for elements, 282–285
    HTML5 in older browsers, 272–273
    making elements float, 295–300
    managing overflow, 305
    margins around elements, 292–293
    normalizing default styles, 274–275
    padding around elements, 286–287
    relative positioning of elements, 301
    resetting default styles, 274–275
    visibility of elements, 278–281
    width for elements, 282–285
  legend element, 418–421
  li element, 390
    applying classes to, 396
    placing list content in, 392
  line breaks, creating, 118–119, 123
  line height, setting, 245
  linear gradient, 376, 378
  link element, 192
  linking
    blocks of content, 161–162
    to external style sheets, 191
    thumbnail images, 168
  links
    a element, 158
    applying style properties, 162
    avoiding "click here" labels, 162
    compressing files for download, 168
    creating different kinds of, 166–168
    creating to other webpages, 158–160
    designating as navigation, 56–58
    destinations, 157
    to email addresses, 167
    href attribute, 158–159
    label lengths, 162
    labels, 157
    navigating with keyboard, 158
    navigation, 162
    to PDFs, 168
    rel attribute, 160
    selecting based on states, 220–221
    to phone numbers, 167
    tabindex attribute, 158
  list attribute, 411
  list numbering, starting, 397
  list type, choosing, 391
  lists
    Boolean reversed attribute, 392
    choosing markers, 393
    content direction, 392
    creating custom markers, 394–396
    dd (description of term), 389
    displaying without markers, 393
    dl (description list), 389, 404–407
    dt (term to describe), 389
    for marking up navigation links, 390
    hanging markers, 398
media features, 319–320
for Retina displays and other high-pixel-density displays, 332
in style sheets, 323
for style sheet for responsive webpage, 329–330
syntax, 320–322
targeting viewport widths, 330
using ems in, 322
media-specific style sheets, 200–201, 319–323
megapixels, 136
menu element, 526
meta element, 45, 324–325
meter element, 128–130
method attribute, 414
method="get" vs. method="post", 415
microformats, implementing, 83
MIME type, setting, 451
min-height vs. height, 284
Miro Video Converter, 451
missing images, fixing, 510
misspelled words, noting, 122
mobile compatibility, testing for, 501
mobile devices resources, 332
mobile first approach, following, 327
Modernizr website, 363
multimedia
native, 450
resources, 476
multiple attribute, 431, 437
muted attribute, 453, 465
name attribute, 413, 436, 437, 522
Namecheap website, 512
native multimedia
accessibility, 462
explained, 450
nav element, 56
links in, 56–58
with ul and ol, 57
navigation
including on pages, 162
marking, 56–58
navigation landmark role, 79–80
nested lists
styling, 400–403
using for drop-down navigation, 403
nesting elements, 11
nh mn ms duration format, 100
Node.js, 486
normalize.css, 105
noscript element, 527
Notepad text editor
displaying files in, 36
naming files, 33
Save as option, 33
using, 30
novalidate attribute, 415
numbered lists, starting, 397
NVDA screen reader, 78

O
object element, 527
oblique text, explained, 237
ol (ordered list) element, 390
creating, 390–392
explained, 389, 527
using with nav, 57
“One Web” presentation, 311
opacity property, setting, 382–383
open attribute, 523
Open Device Lab website, 501
Opera developer tool, 503
optgroup element, 438
optimum attribute, 128
option element, 437
output element, 411
overflow property, 300, 305

P
p (paragraph) element, 21, 88
padding
  adding around elements, 286–287
  em values for, 294
  percentage-based values for, 318
page constructs, 53. See also webpages
pages. See webpages
Paint.NET website, 141
paragraphs
  creating, 21, 88
  line spacing, 88
param element, 528
parent element
  basing selection on, 214
  explained, 11, 212
password boxes, creating for forms, 427
pattern attribute, 428
percentages, for responsive webpage, 283
photographs, saving, 139
Photoshop
  4-Up tab, 142
  finding image dimensions, 151
  RGB vs. CMYK, 144
  Save for Web command, 142–144
  scaling images with, 154
  shrinking image files, 144
Photoshop alternatives
  Acorn, 141
  Gimp, 141
Paint.NET, 141
Paint Shop Pro, 141
Pixelmator, 141
phrasing content elements, 21
picture element, proposal of, 314
Pixelmator website, 141
pixels
  measurement, 136
  transparency, 138
placeholder attribute, using with forms, 422–424, 426
placeholder value, representing, 113
PNG-* image format variations, 135
  comparing to other formats, 134
  lossless, 135
  transparency, 138
PNG-24, Transparency setting in Photoshop, 139
polyfills, using for non-supporting browsers, 363.
  See also browsers
poster attribute, 456
pre element, 114–115
preformatted text, 114–115
The Principles of Beautiful Web Design, 29
printing images, 137
preload attribute, 457, 466
progress element, 128–131
PSD file layers, 144
pseudo-classes. See also class selectors
  :active, 220–221
  :checked, 446
  :disabled, 446
  :enabled, 446
  explained, 219
  :focus, 220–221, 385, 446
  for forms, 446
  :hover, 220–221
  :invalid, 446, 448
  :link, 220–221
  :optional, 446
  :required, 446
  :valid, 446
  :visited, 220–221
pseudo-elements. See also elements
  :after, 384–386
  :before, 384–386
  explained, 219
PT Sans font, using, 343–345
publishing pages on web. See also webpages
domain and web host, 514
files to server, 515–518
finding site host, 513–514
getting domain name, 512
ISP as web host, 513
web analytics, 513
px (CSS pixels), 181
Retina displays and other high-pixel-density displays
creating images for, 153
icon fonts, 153
media queries for, 332
scaling images for, 152
sizing images for, 153
SVG (scalable vector graphics), 153
reversed attribute, 392
right-to-left languages, incorporating, 127
role attribute, 78,
root relative URLs, 18
rows, cols attributes, 436
rowspan attribute, 482
rp element, 124–125
rt element, 124–125
ruby element, explained, 124–125
rules. See style rules
S
s element, 108, 110–111
Safari
developer tool, 503
testing sites in, 500
samp element, 113
sandbox attribute, 524
saving
animated images, 139
external style sheets, 190
images, 139, 142–144
photographs, 139
source code, 41
webpages, 32–36
scope attribute, 478
screen readers, xxii, 23, 49, 50, 78
Screen Sizes website, 332
script element, 487
seamless attribute, 524
search boxes, creating for forms, 428–431
section element, 63–64
vs. article, 271
considering use of, 64
vs. div element, 63
nesting in article element, 64
section (the word) vs. section element, 46
select boxes, creating for forms, 437–438
select element, 437
selected attribute, 437
selecting
first letter of elements, 218–219
first line of elements, 218–219
links based on states, 220–221
selecting elements. See also elements based on attributes, 222–225
by class, 208–210
by context, 212–215
first child, 216–217
by ID, 208–210
last child, 216–217
by name, 206–207
Selectivizr, 448
selectors. See also CSS (Cascading Style Sheets)
ancestors, 212
class vs. ID, 211
combining, 227–228
constructing, 204
descendants, 212
examples, 204–205
keeping simple, 205
pseudo-classes for forms, 446
resource, 228
sibling elements, 215
using * (asterisk) wildcard, 207
using square brackets ([]) with, 206
semantic HTML, 20–23, 26
semantics
accessibility, 23
code maintenance, 23
CSS styling, 23
importance of, 23
SEO (search engine optimization), 23
shadows, applying to elements, 372
shape attribute, 521
sibling elements, basing selections on, 215
Silk icon set, 388
simulators, using for testing, 501
single vs. double quotes, 348
sites. See also websites
getting inspiration, 29
identifying audiences, 28
mapping folder structure, 28–29
naming conventions, 28
number of pages, 28
planning, 28
reason for creating, 28
resources, 29
sketching on paper, 28
size attribute, 423, 437
sizes attribute, 526
small caps
removing, 261
using, 261
small element, 24–25, 89
Smashing Magazine website, 29
Socialico icon font, 340
source code
saving, 41
viewing CSS, 202
viewing HTML, 40
source element, 468, 459
spacing
specifying kerning, 257
specifying tracking, 257
span attribute, 522
span element, 120–121
special characters, treatment of, 12
specificity, law of, 178–179
spellcheck attribute, 520
sprites, combining images with, 387–388
square brackets ([])
using with attributes, 222
using with selectors, 205
src attribute, 145–146, 452, 459, 464, 487
including on audio, 464
including on img, 145–146
including on video, 452
srclang attribute, 531
srcdoc attribute, 524
srcset attribute, proposal of, 314
stacks, positioning elements in, 304
start attribute, 397
step attribute, 525
strong element, 90
structure of documents, defining, 44–45, 50, 53
style element, 194, 196
style rules
adding comments to, 172–173
collision of, 177–179
constructing, 171
ignored by browsers, 448
!important, 179
law of specificity, 178–179
order of, 179
style sheets. See also CSS (Cascading Style Sheets); embedded style sheets; external style sheets
defined, 169
inline styles, 196–197
location of external, 193
media-specific, 200–201
saving external, 190
sources, 179
styles. See also default styles; layout with styles
applying to groups of elements, 226
order of, 198–199
styles for text formatting. See also fonts; text
adding indents, 258
aligning text, 259
alternate fonts, 233–235
backgrounds, 250–252
styles for text formatting (continued)
bold, 238–239
controlling spacing, 257
decorating text, 262–263
font family, 232
font sizes, 240–244
font values at once, 246–247
italics, 236–237
line height, 245
setting color, 248–249
small caps, 261
text case, 260
whitespace properties, 264
sub element, 103–104
  fixing line spacing, 105
sub-folders, creating from folders, 37
subhead element, 52
  Sublime Text editor, 31
submit button, creating for forms, 441–443
subscripts, creating, 103–104
subsetting, using with web fonts, 337
summary element, 529
sup element, 103–105
  superscripts, creating, 103–104
SVG (scalable vector graphics)
  coupling video with, 475
  explained, 137
svg element, 475

T
tabindex attribute, 158
table element, 478
tables
  adding padding, 481
  advanced examples, 477
caption element, 481
colspan attribute, 482–483
column headers, 479
row headers, 479
rowspan attribute, 482–483
scope attribute for th, 481
spanning columns, 482–483
spanning rows, 482–483
structuring, 478–481
tbody (table body) element, 481
td (table data) element, 478
tfoot (table footer) element 479–480
th (table header cell) element, 478, 480
thead (table header) element, 479
tr (table row) element, 478, 480
target attribute, 163
  accessibility concerns, 163
  best practices, 163
  opening links in iframes, 163
  usability concerns, 163
tbody element, 479
td element, 478
  telephone boxes, creating for forms, 428–431
terms, defining, 103
testing
  browsers, 500
  with emulators, 501
  for mobile compatibility, 501
  obtaining browsers for, 500
  with simulators, 501
  webpages, 498–499
text. See also fonts; styles for text formatting
  adding drop shadows to, 368–369
  aligning, 259
  alternative for missing images, 147–148
  blank alternative, 148
  character references or entities, 12
  decorating, 262–263
  emphasizing, 90
  encoding, 12
  highlighting, 116–117
  marking as important, 90
  marking deleted, 108–109
  marking inserted, 108–109
  noting inaccuracies, 108–111
  preformatted, 114–115
  quoting, 95–97
  styling with web fonts, 346–348
  wrapping around elements, 295–296
text areas, creating for forms, 436
text boxes, creating for forms, 422–424
text case
  capitalize value, 260
  changing, 260
  lowercase value, 260
  uppercase value, 260
text editors
  availability, 31
  BBEdit, 31
  Coda, 31
  creating webpages in, 31
displaying files in, 36
  Espresso, 31
  Notepad, 30–31
  for OS X, 31
  Sublime Text, 31
  TextMate, 31
  TextWrangler, 30–31
  for Windows, 31
text formatting. See styles for text formatting
textarea element, 436
  TextMate editor, 31
  text-shadow property, using, 368–371
Index

mailto scheme, 15–16
paths, 15
relative, 17–18
schemes, 15–16
separating words with dashes, 14
usemap attribute, 524
user input instructions, marking up, 113
UTF-8
  encoding pages in, 12, 34
  saving files as, 32–34, 47, 190
  without BOM, 34

V
validating code, 496–497
value attribute, 128, 130, 397, 422, 437, 443
values, including in attributes, 9–10
var element, 113
vertical alignment, applying to elements, 306–307
video element, 452
  accessibility, 462
  adding to webpages, 452–453
  adding with Flash fallback, 470–474
  autoplay, 454–455
  Boolean attributes, 455
  controls, 454–455
  coupling with SVG (Scalable Vector Graphics), 475
  with multiple sources, 459–461
  preventing from preloading, 457–458
  resources, 476
  using with canvas element, 475
video element attributes
  autoplay, 453, 456
  controls, 453
  height, 453
  loop, 453, 456
  media attribute, 461
  muted, 453
  poster, 453–454
  preload, 453
  source, 461
  src, 453
  type, 461
  width, 453
video file formats
  converting between, 451
  .m4v, 451
  .mp4, 451
  .ogv (Ogg Theora), 451
  unsupported, 454
  .webm, 451
Video.js Flash fallback solution, 474
View Source, viewing HTML code with, 40–41

TextWrangler text editor
  naming files, 33
  using, 30
tfoot element, 479
th element, 478
thead element, 479
third-party plugins, 450
time element, 98–99
time formats
  datetime attribute, 100
  nh nm ns, 100
  YYYY-MM-DDThh:mm:ss, 100
time zone offsets, 100
times, global, 100
title attribute, 84, 103
  adding to elements, 84
  using with abbreviations, 101
title element, 45–49
touch icons, creating, 155–156
tr element, 478
track element, 84, 462
tracking, specifying, 257
transferring files to server, 515–518
transparency
  alpha, 138, 144
  availability of, 138
  borders, 276
  index, 138
triangles, creating with border styles, 386
troubleshooting. See debugging techniques; testing
.txt extension, 32
type attribute, 393, 411, 422, 461, 523
type selector, explained, 206
Typekit web font service, 339, 344
typemustmatch attribute, 527

U
u element, 122–123
ul (unordered list) element, 390
  creating, 390–392
  explained, 389
  using with nav, 57
Unicode, use of, 12
URL boxes, creating for forms, 428–431
URLs (uniform resource locators)
  absolute, 16
  absolute vs. relative, 19
  components, 15
  creating links to, 166–168
  ftp scheme, 15–16
  hosts, 15
  http and https, 16
  lowercase convention, 162

mailto scheme, 15–16
paths, 15
relative, 17–18
schemes, 15–16
separating words with dashes, 14
usemap attribute, 524
user input instructions, marking up, 113
UTF-8
  encoding pages in, 12, 34
  saving files as, 32–34, 47, 190
  without BOM, 34

V
validating code, 496–497
value attribute, 128, 130, 397, 422, 437, 443
values, including in attributes, 9–10
var element, 113
vertical alignment, applying to elements, 306–307
video element, 452
  accessibility, 462
  adding to webpages, 452–453
  adding with Flash fallback, 470–474
  autoplay, 454–455
  Boolean attributes, 455
  controls, 454–455
  coupling with SVG (Scalable Vector Graphics), 475
  with multiple sources, 459–461
  preventing from preloading, 457–458
  resources, 476
  using with canvas element, 475
video element attributes
  autoplay, 453, 456
  controls, 453
  height, 453
  loop, 453, 456
  media attribute, 461
  muted, 453
  poster, 453–454
  preload, 453
  source, 461
  src, 453
  type, 461
  width, 453
video file formats
  converting between, 451
  .m4v, 451
  .mp4, 451
  .ogv (Ogg Theora), 451
  unsupported, 454
  .webm, 451
Video.js Flash fallback solution, 474
View Source, viewing HTML code with, 40–41

TextWrangler text editor
  naming files, 33
  using, 30
tfoot element, 479
th element, 478
thead element, 479
third-party plugins, 450
time element, 98–99
time formats
  datetime attribute, 100
  nh nm ns, 100
  YYYY-MM-DDThh:mm:ss, 100
time zone offsets, 100
times, global, 100
title attribute, 84, 103
  adding to elements, 84
  using with abbreviations, 101
title element, 45–49
touch icons, creating, 155–156
tr element, 478
track element, 84, 462
tracking, specifying, 257
transferring files to server, 515–518
transparency
  alpha, 138, 144
  availability of, 138
  borders, 276
  index, 138
triangles, creating with border styles, 386
troubleshooting. See debugging techniques; testing
.txt extension, 32
type attribute, 393, 411, 422, 461, 523
type selector, explained, 206
Typekit web font service, 339, 344
typemustmatch attribute, 527

U
u element, 122–123
ul (unordered list) element, 390
  creating, 390–392
  explained, 389
  using with nav, 57
Unicode, use of, 12
URL boxes, creating for forms, 428–431
URLs (uniform resource locators)
  absolute, 16
  absolute vs. relative, 19
  components, 15
  creating links to, 166–168
  ftp scheme, 15–16
  hosts, 15
  http and https, 16
  lowercase convention, 162
viewports, 324–325
meta element, 324–325
targeting with media queries, 330
visibility of elements, controlling, 278–281
visitors, allowing to upload files, 439
VMs (virtual machines), using to test
browsers, 500
VoiceOver screen reader, 78
void elements, 8

W
W3C’s validator, accessing, 497
WAI-ARIA (Web Accessibility Initiative’s Accessible
Rich Internet Applications), 78–81
wbr element, 123
web analytics, 513
web browsers. See browsers
web font services
Cloud.typography, 339
Edge Web Fonts, 339
Fontdeck, 339
Fonts.com, 339
Fontspring.com, 339
Google Fonts, 339
Typekit, 339
WebINK, 339
web fonts. See also fonts; icon fonts
applying bold italic with, 354–356
applying bold with, 352–353
applying italics with, 350–351
browser support, 336
choosing, 344
downloading, 342–344
.eot (Embedded OpenType), 336
explained, 336
file formats, 336
finding, 338–341
Font Squirrel, 338
@font-face, 336–337, 345
from Google Fonts, 357–359
The League of Moveable Type website, 338
legal issues, 336–337
managing file sizes, 337
MyFonts, 338
.otf (OpenType), 336
PT Sans, 343–345
quality and rendering, 340
self-hosting, 338
single vs. double quotes, 348
styling text with, 346–348
subsetting, 337
.svg (Scalable Vector Graphics), 336
ttf (TrueType), 336
Typekit web font service, 344
using in Photoshop, 344
using in style sheets, 346–348
using number of, 344
viewing in demo HTML, 344
.woff (Web Open Font Format), 336
web host
connecting domain with, 514
finding, 513–514
ISP as, 513
web server, creating with JavaScript, 486
WebP image format, 135
webpages. See also HTML pages; page
constructs; publishing pages on web;
responsive webpages
alt attribute, 148
ARIA spec, 81
components, 26
connection-speed simulations, 146
content, 6–7
content placement, 26
creating, 30
creating in text editors, 31
Creative Commons, 140
default missing, 35
document head, 5, 45–46
documenting, 458
editing, 36
ending, 45
file references, 1
fixed-width, 266
getting inspiration from, 40–41
html element, 5, 44
language subtag lookup tool, 47
markup, 1
meta tag, 4
refreshing in Chrome, 39
rel value resource, 162
rendering, 7
responsive, 266
saving, 32–36
sections, 46
structuring pages, 268–271
testing, 498–499
text content, 1, 12
viewing in browsers, 38–39
websites. See also sites
Acorn, 141
Adobe Edge Inspect, 501
background properties, 256
BBEdit text editor, 31
BrowserStack, 500
Can I Use, 499
Coda text editor, 31
CSS Arrow Please! 386
CSS gradient background maker, 381
CSS Sprite Generator, 388
CSS3 Generator, 364
CSS3 Patterns Gallery, 381
CSS3 PIE, 363
“A Dao of Web Design,” 311
DeviceAnywhere, 501
domain names, 512
Dribbble, 362
Electric Plum, 501
Espresso text editor, 31
favicons, 156
Flickr, 140
Flowplayer solution, 474
Font Squirrel, 342
@font-face generator, 337
Fontfabric, 340
FTP clients, 515, 518
Gimp, 141
Hover, 512
HTML5 Boilerplate, 156
HTML5 Please, 363
icon fonts, 340
ImageOptim, 144
IcoMoon, 340
iOS Simulator, 501
JAWS, 78
JPEGmini, 144
JW Player solution, 474
A List Apart, 29
MediaElement.js, 470–471
microformats, 119
Miro Video Converter, 451
Mobile HTML5, 332
Modernizr, 363
multimedia resources, 476
Namecheap, 512
normalize.css, 105
NVDA, 78
“One Web” presentation, 311
Open Device Lab, 501
Paint.NET, 141
Photoshop, 141
Pixelmator, 141
Quirksmode, 499
Resources for creating, 29
Respond.js, 334
responsive website examples, 332
Retina images, 152
Screen Sizes, 332
Selectivizr, 448
selectors in CSS3, 228
Silk icon set, 388
Smashing Magazine, 29
specificity weights, 179
Sublime Text editor, 31
table examples, 477
TextMate editor, 31
TextWrangler text editor, 30
verifying in desktop browsers, 500
video resources, 476
Video.js solution, 474
VoiceOver, 78
W3C’s validator, 497
Window-Eyes, 78
X-Icon Editor, 156
WebVTT file format, 462
white-space property, 264
width, height attributes, 150, 442, 452
width. See also max-width
calculating auto value for, 284–285
setting for elements, 282–285
Window-Eyes screen reader, 78
word processors, avoiding, 31
words, noting misspellings, 122
wrap attribute, 530

X
XHTML, use of space and forward slash, 9
X-Icon Editor, 156

Z
z-index property, using, 304