

Making Everything Easier!™

3rd Edition

HTML5 and CSS3

ALL-IN-ONE

FOR
DUMMIES®
A Wiley Brand

**8 BOOKS
IN 1**

- Creating the HTML Foundation
- Styling with CSS
- Building Layouts with CSS
- Client-Side Programming with JavaScript®
- Server-Side Programming with PHP
- Managing Data with MySQL®
- Integrating the Client and Server with AJAX
- Moving from Pages to Sites

Andy Harris



Get More and Do More at Dummies.com®



Start with **FREE** Cheat Sheets

Cheat Sheets include

- Checklists
- Charts
- Common Instructions
- And Other Good Stuff!

To access the Cheat Sheet created specifically for this book, go to
www.dummies.com/cheatsheet/html5css3aio

Get Smart at Dummies.com

Dummies.com makes your life easier with 1,000s of answers on everything from removing wallpaper to using the latest version of Windows.

Check out our

- Videos
- Illustrated Articles
- Step-by-Step Instructions

Plus, each month you can win valuable prizes by entering our Dummies.com sweepstakes. *

Want a weekly dose of Dummies? Sign up for Newsletters on

- Digital Photography
- Microsoft Windows & Office
- Personal Finance & Investing
- Health & Wellness
- Computing, iPods & Cell Phones
- eBay
- Internet
- Food, Home & Garden

Find out “HOW” at Dummies.com

*Sweepstakes not currently available in all countries; visit Dummies.com for official rules.



HTML5 and CSS3

ALL-IN-ONE

FOR
DUMMIES®
A Wiley Brand

3rd Edition

by Andy Harris

FOR
DUMMIES®
A Wiley Brand

HTML5 and CSS3 All-in-One For Dummies®, 3rd Edition

Published by:

John Wiley & Sons, Inc.,

111 River Street,

Hoboken, NJ 07030-5774,

www.wiley.com

Copyright © 2014 by John Wiley & Sons, Inc., Hoboken, New Jersey

Media and software compilation copyright © 2014 by John Wiley & Sons, Inc. All rights reserved.

Published simultaneously in Canada

No part of this publication may be reproduced, stored in a retrieval system or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, scanning or otherwise, except as permitted under Sections 107 or 108 of the 1976 United States Copyright Act, without the prior written permission of the Publisher. Requests to the Publisher for permission should be addressed to the Permissions Department, John Wiley & Sons, Inc., 111 River Street, Hoboken, NJ 07030, (201) 748-6011, fax (201) 748-6008, or online at <http://www.wiley.com/go/permissions>.

Trademarks: Wiley, For Dummies, the Dummies Man logo, Dummies.com, Making Everything Easier, and related trade dress are trademarks or registered trademarks of John Wiley & Sons, Inc. and may not be used without written permission. All other trademarks are the property of their respective owners. John Wiley & Sons, Inc. is not associated with any product or vendor mentioned in this book.

LIMIT OF LIABILITY/DISCLAIMER OF WARRANTY: THE PUBLISHER AND THE AUTHOR MAKE NO REPRESENTATIONS OR WARRANTIES WITH RESPECT TO THE ACCURACY OR COMPLETENESS OF THE CONTENTS OF THIS WORK AND SPECIFICALLY DISCLAIM ALL WARRANTIES, INCLUDING WITHOUT LIMITATION WARRANTIES OF FITNESS FOR A PARTICULAR PURPOSE. NO WARRANTY MAY BE CREATED OR EXTENDED BY SALES OR PROMOTIONAL MATERIALS. THE ADVICE AND STRATEGIES CONTAINED HEREIN MAY NOT BE SUITABLE FOR EVERY SITUATION. THIS WORK IS SOLD WITH THE UNDERSTANDING THAT THE PUBLISHER IS NOT ENGAGED IN RENDERING LEGAL, ACCOUNTING, OR OTHER PROFESSIONAL SERVICES. IF PROFESSIONAL ASSISTANCE IS REQUIRED, THE SERVICES OF A COMPETENT PROFESSIONAL PERSON SHOULD BE SOUGHT. NEITHER THE PUBLISHER NOR THE AUTHOR SHALL BE LIABLE FOR DAMAGES ARISING HEREFROM. THE FACT THAT AN ORGANIZATION OR WEBSITE IS REFERRED TO IN THIS WORK AS A CITATION AND/OR A POTENTIAL SOURCE OF FURTHER INFORMATION DOES NOT MEAN THAT THE AUTHOR OR THE PUBLISHER ENDORSES THE INFORMATION THE ORGANIZATION OR WEBSITE MAY PROVIDE OR RECOMMENDATIONS IT MAY MAKE. FURTHER, READERS SHOULD BE AWARE THAT INTERNET WEBSITES LISTED IN THIS WORK MAY HAVE CHANGED OR DISAPPEARED BETWEEN WHEN THIS WORK WAS WRITTEN AND WHEN IT IS READ.

For general information on our other products and services, please contact our Customer Care Department within the U.S. at 877-762-2974, outside the U.S. at 317-572-3993, or fax 317-572-4002. For technical support, please visit www.wiley.com/techsupport.

Wiley publishes in a variety of print and electronic formats and by print-on-demand. Some material included with standard print versions of this book may not be included in e-books or in print-on-demand. If this book refers to media such as a CD or DVD that is not included in the version you purchased, you may download this material at <http://booksupport.wiley.com>. For more information about Wiley products, visit www.wiley.com.

Library of Congress Control Number: 2013952425

ISBN 978-1-118-28938-9 (pbk); ISBN 978-1-118-42139-0 (ePub); ISBN 978-1-118-41983-0 (ePDF)

Manufactured in the United States of America

10 9 8 7 6 5 4 3 2 1

Contents at a Glance

<i>Introduction</i>	<i>1</i>
<i>Part I: Creating the HTML Foundation</i>	<i>7</i>
Chapter 1: Sound HTML Foundations	9
Chapter 2: It's All About Validation	19
Chapter 3: Choosing Your Tools	33
Chapter 4: Managing Information with Lists and Tables	51
Chapter 5: Making Connections with Links	67
Chapter 6: Adding Images, Sound, and Video	77
Chapter 7: Creating Forms	105
<i>Part II: Styling with CSS</i>	<i>129</i>
Chapter 1: Coloring Your World	131
Chapter 2: Styling Text	149
Chapter 3: Selectors: Coding with Class and Style	175
Chapter 4: Borders and Backgrounds	197
Chapter 5: Levels of CSS	225
Chapter 6: CSS Special Effects	245
<i>Part III: Building Layouts with CSS</i>	<i>263</i>
Chapter 1: Fun with the Fabulous Float	265
Chapter 2: Building Floating Page Layouts	285
Chapter 3: Styling Lists and Menus	309
Chapter 4: Using Alternative Positioning	327
<i>Part IV: Client-Side Programming with JavaScript</i>	<i>353</i>
Chapter 1: Getting Started with JavaScript	355
Chapter 2: Talking to the Page	375
Chapter 3: Decisions and Debugging	399
Chapter 4: Functions, Arrays, and Objects	429
Chapter 5: Getting Valid Input	459
Chapter 6: Drawing on the Canvas	483
Chapter 7: Animation with the Canvas	511

<i>Part V: Server-Side Programming with PHP</i>	527
Chapter 1: Getting Started on the Server	529
Chapter 2: PHP and HTML Forms	549
Chapter 3: Using Control Structures	569
Chapter 4: Working with Arrays	587
Chapter 5: Using Functions and Session Variables	605
Chapter 6: Working with Files and Directories	617
Chapter 7: Exceptions and Objects	639
<i>Part VI: Managing Data with MySQL</i>	653
Chapter 1: Getting Started with Data	655
Chapter 2: Managing Data with MySQL	679
Chapter 3: Normalizing Your Data	705
Chapter 4: Putting Data Together with Joins	719
Chapter 5: Connecting PHP to a MySQL Database	741
<i>Part VII: Integrating the Client and Server with AJAX</i>	759
Chapter 1: AJAX Essentials	761
Chapter 2: Improving JavaScript and AJAX with jQuery	775
Chapter 3: Animating jQuery	795
Chapter 4: Using the jQuery User Interface Toolkit	819
Chapter 5: Improving Usability with jQuery	841
Chapter 6: Working with AJAX Data	859
Chapter 7: Going Mobile	883
<i>Part VIII: Moving from Pages to Sites</i>	909
Chapter 1: Managing Your Servers	911
Chapter 2: Planning Your Sites	933
Chapter 3: Introducing Content Management Systems	953
Chapter 4: Editing Graphics	977
Chapter 5: Taking Control of Content	995
<i>Index</i>	1015

Table of Contents

<i>Introduction</i>	1
About This Book	1
Foolish Assumptions	2
Use Any Computer	3
Don't Buy Any Software	3
How This Book Is Organized	3
New for the Third Edition	4
Icons Used in This Book	5
Beyond the Book	6
Where to Go from	6
 <i>Part I: Creating the HTML Foundation</i>	7
 Chapter 1: Sound HTML Foundations	9
Creating a Basic Page	9
Understanding the HTML in the Basic Page	11
Meeting Your New Friends, the Tags	12
Setting Up Your System	15
Displaying file extensions	15
Setting up your software	16
 Chapter 2: It's All About Validation	19
Somebody Stop the HTML Madness!	19
XHTML had some great ideas	20
Validating Your Page	23
Aesop visits W3C	25
Using Tidy to repair pages	30
 Chapter 3: Choosing Your Tools	33
What's Wrong with the Big Boys: Expression Web and Adobe Dreamweaver	33
How About Online Site Builders?	34
Alternative Web Development Tools	35
Picking a Text Editor	35
Tools to avoid unless you have nothing else	36
Suggested programmer's editors	36
My Personal Choice: Komodo Edit	41
Other text editors	43
The bottom line on editors	44

Finding a Good Web Developer's Browser	44
A little ancient history	44
Overview of the prominent browsers	46
Other notable browsers	48
The bottom line in browsers	49
Chapter 4: Managing Information with Lists and Tables	51
Making a List and Checking It Twice	51
Creating an unordered list	51
Creating ordered lists	53
Making nested lists	54
Building the definition list	57
Building Tables	59
Defining the table	60
Spanning rows and columns	63
Avoiding the table-based layout trap	65
Chapter 5: Making Connections with Links	67
Making Your Text Hyper	67
Introducing the anchor tag	68
Comparing block-level and inline elements	69
Analyzing an anchor	69
Introducing URLs	70
Making Lists of Links	71
Working with Absolute and Relative References	73
Understanding absolute references	73
Introducing relative references	73
Chapter 6: Adding Images, Sound, and Video	77
Adding Images to Your Pages	77
Linking to an image	78
Adding inline images using the tag	80
src (source)	81
height and width	81
alt (alternate text)	81
Choosing an Image Manipulation Tool	82
An image is worth 3.4 million words	82
Introducing IrfanView	84
Choosing an Image Format	85
BMP	85
JPG/JPEG	86
GIF	86
PNG	88
SVG	89
Summary of web image formats	90
Manipulating Your Images	90
Changing formats in IrfanView	90
Resizing your images	91
Enhancing image colors	92

Using built-in effects	93
Other effects you can use	97
Batch processing	98
Working with Audio	99
Adding video	101

Chapter 7: Creating Forms. 105

You Have Great Form	105
Forms must have some form	107
Building Text-Style Inputs	109
Making a standard text field	109
Building a password field	111
Making multi-line text input	112
Creating Multiple Selection Elements	114
Making selections	114
Building check boxes	116
Creating radio buttons	117
Pressing Your Buttons	119
Making input-style buttons	120
Building a Submit button	121
It's a do-over: The Reset button	121
Introducing the <button> tag	121
New form input types	122
date	122
time	123
datetime	123
datetime-local	123
week	124
month	125
color	125
number	125
range	126
search	126
email	127
tel	127
url	127

***Part II: Styling with CSS*..... 129**

Chapter 1: Coloring Your World. 131

Now You Have an Element of Style	131
Setting up a style sheet	133
Changing the colors	134
Specifying Colors in CSS	134
Using color names	135
Putting a hex on your colors	136
Coloring by number	136

Hex education.....	137
Using the web-safe color palette.....	139
Choosing Your Colors.....	140
Starting with web-safe colors	141
Modifying your colors	141
Doing it on your own pages.....	141
Changing CSS on the fly.....	142
Creating Your Own Color Scheme.....	143
Understanding hue, saturation, and lightness.....	143
Using HSL colors in your pages	145
Using the Color Scheme Designer.....	146
Selecting a base hue	147
Picking a color scheme	148

Chapter 2: Styling Text149

Setting the Font Family	149
Applying the font-family style attribute.....	150
Using generic fonts	151
Making a list of fonts	153
The Curse of Web-Based Fonts	154
Understanding the problem	154
Using Embedded Fonts	155
Using images for headlines.....	158
Specifying the Font Size.....	160
Size is only a suggestion!.....	160
Using the font-size style attribute.....	161
Absolute measurement units	162
Relative measurement units.....	163
Determining Other Font Characteristics	164
Using font-style for italics	165
Using font-weight for bold	166
Using text-decoration	167
Using text-align for basic alignment	169
Other text attributes.....	170
Using the font shortcut	171
Working with subscripts and superscripts	172

Chapter 3: Selectors: Coding with Class and Style175

Selecting Particular Segments.....	175
Defining more than one kind of paragraph.....	175
Styling identified paragraphs	176
Using Emphasis and Strong Emphasis.....	177
Modifying the Display of em and strong.....	179
Defining Classes	180
Adding classes to the page.....	181

Using classes	182
Combining classes	182
Introducing div and span.....	184
Organizing the page by meaning.....	185
Why not make a table?	186
Using Pseudo-Classes to Style Links	187
Styling a standard link.....	187
Styling the link states	187
Best link practices	189
Selecting in Context.....	190
Defining Styles for Multiple Elements	191
Using New CSS3 Selectors	193
attribute selection	193
not.....	194
nth-child	194
Other new pseudo-classes.....	195

Chapter 4: Borders and Backgrounds. 197

Joining the Border Patrol	197
Using the border attributes	197
Defining border styles	199
Using the border shortcut	200
Creating partial borders.....	201
Introducing the Box Model.....	202
Borders, margin, and padding.....	203
Positioning elements with margins and padding.....	205
New CSS3 Border Techniques.....	207
Image borders	207
Adding Rounded Corners	209
Adding a box shadow	210
Changing the Background Image.....	212
Getting a background check.....	214
Solutions to the background conundrum.....	215
Manipulating Background Images	218
Turning off the repeat	218
Using CSS3 Gradients	219
Using Images in Lists.....	223

Chapter 5: Levels of CSS 225

Managing Levels of Style	225
Using local styles	225
Using an external style sheet	228
Understanding the Cascading Part of Cascading Style Sheets	233
Inheriting styles	233
Hierarchy of styles.....	234

Overriding styles.....	235
Precedence of style definitions	236
Managing Browser Incompatibility	237
Coping with incompatibility	237
Making Internet Explorer-specific code	238
Using a conditional comment with CSS	240
Checking the Internet Explorer version	242
Using a CSS reset.....	243

Chapter 6: CSS Special Effects245

Image Effects	245
Transparency	245
Reflections	247
Text Effects.....	249
Text stroke.....	249
Text-shadow	251
Transformations and Transitions.....	252
Transformations	253
Three-dimensional transformations.....	254
Transition animation	257
Animations.....	259

Part III: Building Layouts with CSS 263

Chapter 1: Fun with the Fabulous Float265

Avoiding Old-School Layout Pitfalls.....	265
Problems with frames	265
Problems with tables.....	266
Problems with huge images.....	267
Problems with Flash	267
Introducing the Floating Layout Mechanism	268
Using float with images	269
Adding the float property	270
Using Float with Block-Level Elements	271
Floating a paragraph.....	271
Adjusting the width	273
Setting the next margin	275
Using Float to Style Forms.....	276
Using float to beautify the form	279
Adjusting the fieldset width.....	282
Using the clear attribute to control page layout	283

Chapter 2: Building Floating Page Layouts	285
Creating a Basic Two-Column Design	285
Designing the page	285
Building the HTML	287
Using temporary background colors	288
Setting up the floating columns	290
Tuning up the borders	291
Advantages of a fluid layout	292
Using semantic tags	292
Building a Three-Column Design	295
Styling the three-column page	296
Problems with the floating layout	298
Specifying a min-height	299
Using height and overflow	300
Building a Fixed-Width Layout	302
Setting up the HTML	303
Fixing the width with CSS	303
Building a Centered Fixed-Width Layout	305
Making a surrogate body with an all div	306
How the jello layout works	307
Limitations of the jello layout	308
 Chapter 3: Styling Lists and Menus	 309
Revisiting List Styles	309
Defining navigation as a list of links	310
Turning links into buttons	310
Building horizontal lists	313
Creating Dynamic Lists	314
Building a nested list	315
Hiding the inner lists	317
Getting the inner lists to appear on cue	318
Building a Basic Menu System	321
Building a vertical menu with CSS	322
Building a horizontal menu	324
 Chapter 4: Using Alternative Positioning	 327
Working with Absolute Positioning	327
Setting up the HTML	327
Adding position guidelines	328
Making absolute positioning work	330
Managing z-index	331
Handling depth	331
Working with z-index	332
Building a Page Layout with Absolute Positioning	332
Overview of absolute layout	333

Writing the HTML	334
Adding the CSS	335
Creating a More Flexible Layout	336
Designing with percentages	337
Building the layout	339
Exploring Other Types of Positioning	340
Creating a fixed menu system	340
Setting up the HTML	341
Setting the CSS values	342
Flexible Box Layout Model	344
Creating a flexible box layout	345
Viewing a flexible box layout	346
... And now for a little reality	348
Determining Your Layout Scheme	351

Part IV: Client-Side Programming with JavaScript 353

Chapter 1: Getting Started with JavaScript. 355

Working in JavaScript	355
Choosing a JavaScript editor	356
Picking your test browser	356
Writing Your First JavaScript Program	357
Embedding your JavaScript code	358
Creating comments	358
Using the alert() method for output	358
Adding the semicolon	359
Introducing Variables	359
Creating a variable for data storage	360
Asking the user for information	361
Responding to the user	361
Using Concatenation to Build Better Greetings	362
Comparing literals and variables	363
Including spaces in your concatenated phrases	364
Understanding the String Object	364
Introducing object-based programming (and cows)	364
Investigating the length of a string	365
Using string methods to manipulate text	366
Understanding Variable Types	368
Adding numbers	369
Adding the user's numbers	370
The trouble with dynamic data	370
The pesky plus sign	371
Changing Variables to the Desired Type	372
Using variable conversion tools	373
Fixing the addInput code	373

Chapter 2: Talking to the Page	375
Understanding the Document Object Model	375
Previewing the DOM	375
Getting the blues, JavaScript-style	377
Writing JavaScript code to change colors	378
Managing Button Events	379
Adding a function for more ... functionality	381
Making a more flexible function	382
Embedding quotes within quotes	384
Writing the changeColor function	384
Managing Text Input and Output	384
Introducing event-driven programming	385
Creating the HTML form	386
Using getElementById to get access to the page	387
Manipulating the text fields	388
Writing to the Document	388
Preparing the HTML framework	390
Writing the JavaScript	390
Finding your innerHTML	391
Working with Other Text Elements	391
Building the form	392
Writing the function	393
Understanding generated source	395
What if you're not in Chrome?	397
 Chapter 3: Decisions and Debugging	 399
Making Choices with If	399
Changing the greeting with if	401
The different flavors of if	402
Conditional operators	403
Nesting your if statements	403
Making decisions with switch	405
Managing Repetition with for Loops	406
Setting up the web page	407
Initializing the output	408
Creating the basic for loop	409
Introducing shortcut operators	410
Counting backwards	411
Counting by fives	412
Understanding the Zen of for loops	413
Building While Loops	413
Making a basic while loop	413
Getting your loops to behave	415
Managing more complex loops	416

Managing Errors with a Debugger	418
Debugging with the interactive console	420
Debugging strategies	422
Resolving syntax errors	422
Squashing logic bugs	424

Chapter 4: Functions, Arrays, and Objects. 429

Breaking Code into Functions	429
Thinking about structure	430
Building the antsFunction.html program	431
Passing Data to and from Functions	432
Examining the makeSong code	434
Looking at the chorus	434
Handling the verses	435
Managing Scope	437
Introducing local and global variables	437
Examining variable scope	437
Building a Basic Array	439
Accessing array data	440
Using arrays with for loops	441
Revisiting the ants song	442
Working with Two-Dimension Arrays	444
Setting up the arrays	446
Getting a city	447
Creating a main() function	448
Creating Your Own Objects	449
Building a basic object	449
Adding methods to an object	450
Building a reusable object	452
Using your shiny new objects	453
Introducing JSON	454
Storing data in JSON format	454
Building a more complex JSON structure	455

Chapter 5: Getting Valid Input 459

Getting Input from a Drop-Down List	459
Building the form	460
Reading the list box	461
Managing Multiple Selections	462
Coding a multiple selection select object	462
Writing the JavaScript code	463
Check, Please: Reading Check Boxes	465
Building the check box page	466
Responding to the check boxes	467
Working with Radio Buttons	468
Interpreting Radio Buttons	469

Working with Regular Expressions	470
Introducing regular expressions	473
Using characters in regular expressions	475
Marking the beginning and end of the line	476
Working with special characters	476
Conducting repetition operations	477
Working with pattern memory	478
New HTML5/CSS3 Tricks for Validation	479
Adding a pattern	481
Marking a field as required	481
Adding placeholder text	481
Chapter 6: Drawing on the Canvas	483
Canvas Basics	483
Setting up the canvas	484
How <canvas> works	485
Fill and Stroke Styles	486
Colors	486
Gradients	487
Patterns	489
Drawing Essential Shapes	491
Rectangle functions	491
Drawing text	492
Adding shadows	494
Working with Paths	496
Line-drawing options	498
Drawing arcs and circles	500
Drawing quadratic curves	502
Building a Bézier curve	503
Images	505
Drawing an image on the canvas	505
Drawing part of an image	507
Manipulating Pixels	508
Chapter 7: Animation with the Canvas	511
Transformations	511
Building a transformed image	512
A few thoughts about transformations	514
Animation	515
Overview of the animation loop	515
Setting up the constants	516
Initializing the animation	517
Animate the current frame	517
Moving an element	519
Bouncing off the walls	520
Reading the Keyboard	521
Managing basic keyboard input	522
Moving an image with the keyboard	523

***Part V: Server-Side Programming with PHP* 527**

Chapter 1: Getting Started on the Server 529

Introducing Server-Side Programming.....	529
Programming on the server.....	529
Serving your programs.....	530
Picking a language	531
Installing Your Web Server.....	532
Inspecting phpinfo().....	533
Building HTML with PHP	536
Coding with Quotation Marks	539
Working with Variables PHP-Style.....	540
Concatenation	541
Interpolating variables into text	542
Building HTML Output.....	543
Using double quote interpolation.....	543
Generating output with heredocs.....	544
Switching from PHP to HTML.....	546

Chapter 2: PHP and HTML Forms 549

Exploring the Relationship between PHP and HTML.....	549
Embedding PHP inside HTML	550
Viewing the results	551
Sending Data to a PHP Program.....	552
Creating a form for PHP processing	552
Receiving data in PHP	555
Choosing the Method of Your Madness	556
Using get to send data.....	557
Using the post method to transmit form data	559
Getting data from the form	560
Retrieving Data from Other Form Elements.....	563
Building a form with complex elements	563
Responding to a complex form	565

Chapter 3: Using Control Structures 569

Introducing Conditions (Again).....	569
Building the Classic if Statement.....	570
Rolling dice the PHP way	571
Checking your six.....	571
Understanding comparison operators.....	574
Taking the middle road.....	574
Building a program that makes its own form.....	576
Making a switch	578
Looping with for	581
Looping with while.....	584

Chapter 4: Working with Arrays	587
Using One-Dimensional Arrays	587
Creating an array	587
Filling an array.....	588
Viewing the elements of an array	588
Preloading an array	589
Using Loops with Arrays	590
Simplifying loops with foreach.....	591
Arrays and HTML.....	593
Introducing Associative Arrays	594
Using foreach with associative arrays	595
Introducing Multidimensional Arrays	597
We're going on a trip	597
Looking up the distance.....	599
Breaking a String into an Array.....	600
Creating arrays with explode	601
Creating arrays with preg_split.....	602
 Chapter 5: Using Functions and Session Variables	 605
Creating Your Own Functions.....	605
Rolling dice the old-fashioned way.....	606
Improving code with functions	607
Managing variable scope	610
Returning data from functions	610
Managing Persistence with Session Variables	611
Understanding session variables.....	613
Adding session variables to your code.....	614
 Chapter 6: Working with Files and Directories	 617
Text File Manipulation	617
Writing text to files	618
Writing a basic text file.....	620
Reading from the file	625
Using Delimited Data.....	626
Storing data in a CSV file.....	627
Viewing CSV data directly.....	629
Reading the CSV data in PHP.....	630
Working with File and Directory Functions	633
opendir()	633
readdir().....	634
chdir()	634
Generating the list of file links.....	635

Chapter 7: Exceptions and Objects 639

Object-Oriented Programming in PHP	639
Building a basic object	640
Using your brand-new class	642
Protecting your data with access modifiers	644
Using access modifiers	645
You've Got Your Momma's Eyes: Inheritance	647
Building a critter based on another critter	648
How to inherit the wind (and anything else)	649
Catching Exceptions	650
Introducing exception handling	650
Knowing when to trap for exceptions	652

Part VI: Managing Data with MySQL 653

Chapter 1: Getting Started with Data 655

Examining the Basic Structure of Data	655
Determining the fields in a record	657
Introducing SQL data types	657
Specifying the length of a record	658
Defining a primary key	659
Defining the table structure	659
Introducing MySQL	660
Why use MySQL?	661
Understanding the three-tier architecture	662
Practicing with MySQL	662
Setting Up phpMyAdmin	663
Changing the root password	665
Adding a user	670
Using phpMyAdmin on a remote server	672
Implementing a Database with phpMyAdmin	674

Chapter 2: Managing Data with MySQL. 679

Writing SQL Code by Hand	679
Understanding SQL syntax rules	680
Examining the buildContact.sql script	680
Dropping a table	681
Creating a table	681
Adding records to the table	682
Viewing the sample data	683
Running a Script with phpMyAdmin	683
Using AUTO_INCREMENT for Primary Keys	686

Selecting Data from Your Tables	688
Selecting only a few fields	689
Selecting a subset of records	690
Searching with partial information.....	692
Searching for the ending value of a field.....	693
Searching for any text in a field.....	693
Searching with regular expressions	694
Sorting your responses	695
Editing Records.....	696
Updating a record	696
Deleting a record.....	697
Exporting Your Data and Structure.....	697
Exporting SQL code	700
Creating XML data	702
Chapter 3: Normalizing Your Data	705
Recognizing Problems with Single-Table Data.....	705
The identity crisis	706
The listed powers	706
Repetition and reliability	708
Fields with changeable data	709
Deletion problems	709
Introducing Entity-Relationship Diagrams	709
Using MySQL workbench to draw ER diagrams.....	709
Creating a table definition in Workbench	710
Introducing Normalization	713
First normal form	714
Second normal form	715
Third normal form	716
Identifying Relationships in Your Data	717
Chapter 4: Putting Data Together with Joins	719
Calculating Virtual Fields.....	719
Introducing SQL functions.....	720
Knowing when to calculate virtual fields.....	721
Calculating Date Values	721
Using DATEDIFF to determine age.....	722
Adding a calculation to get years	723
Converting the days integer into a date.....	723
Using YEAR() and MONTH() to get readable values.....	724
Concatenating to make one field.....	725
Creating a View	726
Using an Inner Join to Combine Tables	728
Building a Cartesian join and an inner join	729
Enforcing one-to-many relationships	731

Counting the advantages of inner joins	732
Building a view to encapsulate the join	733
Managing Many-to-Many Joins.....	733
Understanding link tables.....	735
Using link tables to make many-to-many joins.....	736

Chapter 5: Connecting PHP to a MySQL Database. 741

PHP and MySQL: A Perfect (but Geeky) Romance	741
Understanding data connections.....	744
Introducing PDO.....	745
Building a connection.....	745
Retrieving data from the database	747
Using HTML tables for output.....	748
Allowing User Interaction	751
Building an HTML search form	753
Responding to the search request.....	753

Part VII: Integrating the Client and Server with AJAX..... 759

Chapter 1: AJAX Essentials 761

AJAX Spelled Out	762
A is for asynchronous	763
J is for JavaScript	763
A is for ... and?	763
And X is for ... data.....	763
Making a Basic AJAX Connection	764
Building the HTML form.....	766
Creating an XMLHttpRequest object.....	767
Opening a connection to the server.....	768
Sending the request and parameters	769
Checking the status	769
All Together Now — Making the Connection Asynchronous	771
Setting up the program	772
Building the getAJAX() function	772
Reading the response.....	773

Chapter 2: Improving JavaScript and AJAX with jQuery 775

Introducing jQuery	776
Installing jQuery.....	777
Importing jQuery from Google.....	777
Your First jQuery App.....	778
Setting up the page.....	779
Meet the jQuery node object.....	780
Creating an Initialization Function	781

Using \$(document).ready()	782
Alternatives to document.ready	783
Investigating the jQuery Object	783
Changing the style of an element	783
Selecting jQuery objects	785
Modifying the style	785
Adding Events to Objects	786
Adding a hover event	787
Changing classes on the fly	788
Making an AJAX Request with jQuery	790
Including a text file with AJAX	791
Building a poor man's CMS with AJAX	791
Chapter 3: Animating jQuery	795
Playing Hide and Seek	795
Getting transition support	797
Writing the HTML and CSS foundation	799
Initializing the page	800
Hiding and showing the content	800
Toggling visibility	801
Sliding an element	801
Fading an element in and out	802
Changing Position with jQuery	802
Creating the framework	804
Setting up the events	805
Building the move() function with chaining	806
Building time-based animation with animate()	806
Move a little bit: Relative motion	808
Modifying Elements on the Fly	808
Building the basic page	813
Initializing the code	813
Adding text	813
Attack of the clones	814
It's a wrap	815
Alternating styles	816
Resetting the page	816
More fun with selectors and filters	817
Chapter 4: Using the jQuery User Interface Toolkit	819
What the jQuery User Interface Brings to the Table	819
It's a theme park	820
Using the themeRoller to get an overview of jQuery	820
Wanna drag? Making components draggable	823
Downloading the library	824
Writing the program	826
Resizing on a Theme	827
Examining the HTML and standard CSS	829

Importing the files.....	829
Making a resizable element	830
Adding themes to your elements.....	830
Adding an icon	833
Dragging, Dropping, and Calling Back.....	834
Building the basic page.....	836
Initializing the page.....	836
Handling the drop	838
Beauty school dropout events	838
Cloning the elements	839

Chapter 5: Improving Usability with jQuery841

Multi-Element Designs	841
Playing the accordion widget.....	842
Building a tabbed interface	845
Using tabs with AJAX.....	848
Improving Usability	849
Playing the dating game.....	851
Picking numbers with the slider	852
Selectable elements	854
Building a sortable list	855
Creating a custom dialog box.....	856

Chapter 6: Working with AJAX Data859

Sending Requests AJAX Style.....	859
Sending the data	859
Building a Multipass Application.....	863
Setting up the HTML framework.....	864
Loading the select element.....	865
Writing the loadList.php program	866
Responding to selections.....	867
Writing the showHero.php script	868
Working with XML Data	870
Review of XML.....	871
Manipulating XML with jQuery	872
Creating the HTML.....	873
Retrieving the data	874
Processing the results.....	874
Printing the pet name.....	875
Working with JSON Data.....	876
Knowing JSON's pros	876
Reading JSON data with jQuery	877
Managing the framework	878
Retrieving the JSON data	879
Processing the results.....	879

Chapter 7: Going Mobile 883

Thinking in Mobile.....	883
Building a Responsive Site	885
Specifying a media type	885
Adding a qualifier.....	885
Making Your Page Responsive.....	888
Building the wide layout	891
Adding the narrow CSS	892
Using jQuery Mobile to Build Mobile Interfaces.....	894
Building a basic jQuery mobile page.....	894
Working with collapsible content.....	897
Building a multi-page document	900
Going from Site to App.....	905
Adding an icon to your program.....	906
Removing the Safari toolbar	906
Storing your program offline	907

Part VIII: Moving from Pages to Sites 909

Chapter 1: Managing Your Servers 911

Understanding Clients and Servers.....	911
Parts of a client-side development system.....	912
Parts of a server-side system	913
Creating Your Own Server with XAMPP	914
Running XAMPP	915
Testing your XAMPP configuration	916
Adding your own files.....	916
Setting the security level	917
Compromising between functionality and security	919
Choosing a Web Host	920
Finding a hosting service	920
Connecting to a hosting service.....	922
Managing a Remote Site.....	922
Using web-based file tools	922
Understanding file permissions	924
Using FTP to manage your site.....	925
Using an FTP client	926
Naming Your Site.....	928
Understanding domain names	928
Registering a domain name	929
Managing Data Remotely.....	931
Creating your database.....	931
Finding the MySQL server name	932

Chapter 2: Planning Your Sites933

Creating a Multipage Web Site.....	933
Planning a Larger Site	934
Understanding the Client.....	934
Ensuring that the client's expectations are clear	935
Delineating the tasks	936
Understanding the Audience	937
Determining whom you want to reach.....	937
Finding out the user's technical expertise	938
Building a Site Plan.....	939
Creating a site overview.....	940
Building the site diagram.....	941
Creating Page Templates.....	943
Sketching the page design	943
Building the HTML template framework.....	945
Creating page styles	947
Building a data framework.....	949
Fleshing Out the Project	950
Making the site live.....	950
Contemplating efficiency	951

Chapter 3: Introducing Content Management Systems953

Overview of Content Management Systems.....	954
Previewing Common CMSs.....	955
Moodle.....	955
WordPress	956
Drupal.....	957
Building a CMS site with WebsiteBaker	958
Installing your CMS.....	958
Getting an overview of WebsiteBaker	962
Adding your content.....	962
Using the WYSIWYG editor.....	963
Changing the template	968
Adding additional templates	969
Building Custom Themes.....	971
Adding new functionality	970
Starting with a prebuilt template.....	971
Changing the info.php file.....	973
Modifying index.php.....	974
Modifying the CSS files	975
Packaging your template	976

Chapter 4: Editing Graphics977

Using a Graphic Editor	977
Choosing an Editor.....	978

Introducing Gimp.....	979
Creating an image	980
Painting tools.....	980
Selection tools	982
Modification tools	984
Managing tool options.....	984
Utilities	985
Understanding Layers.....	986
Introducing Filters	988
Solving Common Web Graphics Problems.....	989
Changing a color	989
Building a banner graphic.....	990
Building a tiled background	992

Chapter 5: Taking Control of Content995

Building a “Poor Man’s CMS” with Your Own Code.....	995
Using Server Side Includes (SSIs)	995
Using AJAX and jQuery for client-side inclusion	998
Building a page with PHP includes	1000
Creating Your Own Data-Based CMS.....	1001
Using a database to manage content	1001
Writing a PHP page to read from the table.....	1004
Allowing user-generated content	1007
Adding a new block	1011
Improving the dbCMS design	1013

***Index* 1015**

Introduction

I love the Internet, and if you picked up this book, you probably do, too. The Internet is dynamic, chaotic, exciting, interesting, and useful, all at the same time. The web is pretty fun from a user's point of view, but that's only part of the story. Perhaps the best part of the Internet is how participatory it is. You can build your own content — free! It's really amazing. There's never been a form of communication like this before. Anyone with access to a minimal PC and a little bit of knowledge can create his or her own home-
stead in one of the most exciting platforms in the history of communication.

The real question is how to get there. A lot of web development books are really about how to use some sort of software you have to buy. That's okay, but it isn't necessary. Many software packages have evolved that purport to make web development easier — and some work pretty well — but regardless what software package you use, there's still a need to know what's really going on under the surface. That's where this book comes in.

About This Book

You'll find out exactly how the web works in this book. You'll figure out how to use various tools, but, more importantly, you'll create your piece of the web. You'll discover:

- ✓ **How web pages are created:** You'll figure out the basic structure of web pages. You'll understand the structure well because you build pages yourself. No mysteries here.
- ✓ **How to separate content and style:** You'll understand the foundation of modern thinking about the Internet — that style should be separate from content.
- ✓ **How to use web standards:** The web is pretty messy, but, finally, some standards have arisen from the confusion. You'll discover how these standards work and how you can use them.
- ✓ **How to create great-looking web pages:** Of course, you want a terrific-looking website. With this book, you'll find out how to use layout, style, color, and images.
- ✓ **How to build modern layouts:** Many web pages feature columns, menus, and other fancy features. You'll figure out how to build all these things.
- ✓ **How to add interactivity:** Adding forms to your pages, validating form data, and creating animations are all possible with the JavaScript language.

- ✓ **How to write programs on the server:** Today's web is powered by programs on web servers. You'll discover the powerful PHP language and figure out how to use it to create powerful and effective sites.
- ✓ **How to harness the power of data:** Every web developer eventually needs to interact with data. You'll read about how to create databases that work. You'll also discover how to connect databases to your web pages and how to create effective and useful interfaces.
- ✓ **How AJAX is changing everything:** The hottest web technology on the horizon is AJAX (Asynchronous JavaScript and XML). You'll figure out how to harness this way of working and use it to create even more powerful and interesting applications.

Foolish Assumptions

I don't have any foolish assumptions: I'm not assuming anything in this book. If you've never built a web page before, you're in the right hands. You don't need any experience, and you don't have to know anything about HTML, programming, or databases. I discuss everything you need.

If you're reasonably comfortable with a computer (you can navigate the web and use a word processor), you have all the skills you need.

If you've been around web development for a while, you'll still find this book handy.

If you've used HTML but not HTML5, see how things have changed and discover the powerful combination of HTML5 and CSS3.

You'll see how new HTML and CSS features can literally make your web pages sing and dance, with support for advanced tools like audio and video embedding, animation, and much more.

If you're already comfortable with HTML and CSS, you're ready to add JavaScript functionality for form validation and animation. If you've never used a programming language before, JavaScript is a really great place to start.

If you're starting to get serious about web development, you've probably already realized that you'll need to work with a server at some point. PHP is a really powerful, free, and easy language that's extremely prominent on the web landscape. You'll use this to have programs send e-mails, store and load information from files, and work with databases.

If you're messing with commercial development, you'll definitely need to know more about databases. I get e-mails every week from companies looking for people who can create a solid relational database and connect it to a website with PHP.

If you're curious about AJAX, you can read about what it is, how it works, and how to use it to add functionality to your site. You'll also read about a very powerful and easy AJAX library that can add tremendous functionality to your bag of tricks.

I wrote this book as the reference I wish I had. If you have only one web development book on your shelf, this should be the one. Wherever you are in your web development journey, you can find something interesting and new in this book.

Use Any Computer

One of the great things about web development is how accessible it can be. You don't need a high-end machine to build websites. Whatever you're using now will probably do fine. I tested most of the examples in this book with Windows 7, Ubuntu Linux, and a Macbook pro. I've tested on computers ranging from cutting-edge platforms to mobile devices to a \$35 Raspberry Pi. Most of the software I use in the book is available free for all major platforms. Similar alternatives for all platforms are available in the few cases when this isn't true.

Don't Buy Any Software

Everything you need for web development is on the companion website. I've used only open-source software for this book. Following are the highlights:

- ✓ **Komodo Edit:** Komodo Edit is my current favorite editor. It's a solid free text editor well suited to the many text-editing tasks you'll run across in your programming travels. It also works exactly the same on every platform, so it doesn't really matter what computer or operating system you're running.
- ✓ **XAMPP:** When you're ready to move to the server, XAMPP is a complete server package that's easy to install and incredibly powerful. This includes the incredible Apache web server, the PHP programming language, the MySQL database manager, and tons of useful utilities.
- ✓ **Useful tools:** Every time I use a tool (such as a data mapper, a diagram tool, or an image editor) in this book, I make it available on the companion website.

There's no need to buy any expensive web development tools. Everything you need is here and no harder than the more expensive web editors.

How This Book Is Organized

Web development is about solving a series of connected but different problems. This book is organized into eight minibooks based on specific technologies. You can read them in any order you wish, but you'll find that the later books tend to rely on topics described in the earlier books. (For example, JavaScript doesn't make much sense without HTML because JavaScript is usually embedded in a web page written with HTML.) The following describes these eight minibooks:

- ✓ **Book I: Creating the HTML Foundation** — Web development incorporates a lot of languages and technologies, but HTML is the foundation. Here I show you *HTML5*, the latest incarnation of HTML, and describe how it's used to form the basic skeleton of your pages.
- ✓ **Book II: Styling with CSS** — In the old days, HTML had a few tags to spruce up your pages, but they weren't nearly powerful enough. Today, developers use Cascading Style Sheets (CSS) to add color and formatting to your pages as well as zing and pizzazz. (I'm pretty sure those are formal computer programming words.)
- ✓ **Book III: Building Layouts with CSS** — Discover the best ways to set up layouts with floating elements, fixed positioning, and absolute positioning. Figure out how to build various multicolumn page layouts and how to create dynamic buttons and menus.
- ✓ **Book IV: Client-Side Programming with JavaScript** — Figure out essential programming skills with the easy and powerful JavaScript language — even if you've never programmed before. Manipulate data in web forms and use powerful regular expression technology to validate form entries. Also discover how to create animations with JavaScript with the powerful new `<canvas>` element.
- ✓ **Book V: Server-Side Programming with PHP** — Move your code to the server and take advantage of this powerful language. Figure out how to respond to web requests; work with conditions, functions, objects, and text files; and connect to databases.
- ✓ **Book VI: Managing Data with MySQL** — Most serious Web projects are eventually about data. Figure out how databases are created, how to set up a secure data server, the basics of data normalization, and how to create a reliable and trustworthy data back end for your site.
- ✓ **Book VII: Integrating the Client and Server with AJAX** — Look forward to the technology that has the web abuzz. AJAX isn't really a language but rather a new way of thinking about web development. Get the skinny on what's going on here, build an AJAX connection or two by hand, and read about some really cool libraries for adding advanced features and functionality to your pages.
- ✓ **Book VIII: Moving from Pages to Sites** — This minibook ties together many of the threads throughout the rest of the book. Discover how to create your own complete web server solution or pick a web host. Walk through the process of designing a complex multipage web site. Discover how to use content management systems to simplify complex websites and, finally, to build your own content management system with skills taught throughout the book.

New for the Third Edition

This is actually the third edition of this book. (The previous editions were called *HTML*, *XHTML*, and *CSS All in One For Dummies*.) I have made a few changes to keep up with advances in technology:

- ✓ **Focus on HTML5:** The first edition of the book used HTML4, the second edition used XHTML, and this edition uses HTML5. I'm very excited about HTML5 because it's easier to use than either of the older versions, and quite a bit more powerful.
- ✓ **Integration with CSS3:** CSS3 is the latest incarnation of CSS, and it has some wonderful new features too, including the ability to use custom fonts, animation, and new layout mechanisms.
- ✓ **Improved PHP coverage:** PHP has had some major updates reflected in this book. I have modified all form input to use the safer `filter_input` mechanism, and all database connectivity now uses the PDO library.
- ✓ **Enhanced jQuery coverage:** jQuery has become even more important as a utility library than it was before. The coverage updates some of the nice new features of this library.
- ✓ **A new mobile chapter:** Mobile web development is increasingly important. I provide a new chapter with tips on making your pages mobile-friendly, including use of the jQuery mobile library and building responsive designs that automatically adjust based on screen size.
- ✓ **Support for the WebsiteBaker CMS:** I use this CMS quite a bit in my web business, and I find it especially easy to modify. I changed Book VIII, Chapter 3 to explain how to use and modify this excellent CMS.
- ✓ **Various tweaks and improvements:** No book is perfect (though I really try). There were a few passages in the previous edition that readers found difficult. I tried hard to clean up each of these areas. Many thanks to those who provided feedback!

Icons Used in This Book



This is a *For Dummies* book, so you have to expect some snazzy icons, right? I don't disappoint. Here's what you'll see:

This is where I pass along any small insights I may have gleaned in my travels.



I can't really help being geeky once in a while. Every so often, I want to explain something a little deeper. Read this to impress people at your next computer science cocktail party or skip it if you really don't need the details.



A lot of details are here. I point out something important that's easy to forget with this icon.



Watch out! Anything I mark with this icon is a place where things have blown up for me or my students. I point out any potential problems with this icon.

Beyond the Book

You can find additional features of this book online. Visit the web to find these extras:

- ✓ **Companion website:** www.aharrisbooks.net/haio

This is my primary site for this book. Every single example in the book is up and running on this site so you can see it in action. When necessary, I've also included source code so you can see the source code of anything you can't look at with the ordinary View Source command. I've also posted a link to every piece of software that I mention in the book. If you find any example is not working on your site, please come to my site. If there was a problem with an example in the book, I'll update the site right away, so check my site to compare your code to mine. I also have links to my other books, a forum where you can ask questions, and a form for emailing me any specific questions you might have.

- ✓ **Cheat Sheet:** Go to www.dummies.com/cheatsheet/html5css3aio to find this book's Cheat Sheet. Here, you can find primers on selected HTML syntax, CSS attributes, JavaScript syntax, and MySQL commands.
- ✓ **Dummies.com online articles:** Go to www.dummies.com/extras/html5css3aio to find the Extras for this book. Here you can find articles on topics such as using HTML entities, resetting and extending CSS, JavaScript libraries, using templates with PHP, SQLite and alternative data strategies, fun with jQuery plug-ins, and what's next for the web.
- ✓ **Updates:** *For Dummies* technology books sometimes have updates. To check for updates to this book, go to www.dummies.com/extras/html5css3aio.

Where to Go from Here

Well, that's really up to you. I sincerely believe you can use this book to turn into a top-notch web developer. That's my goal for you.

Although this is a massive book, there's still more to figure out. If you have questions or just want to chat, feel free to e-mail me at andy@aharrisbooks.net. You can also visit my website at www.aharrisbooks.net/ for code examples, updates, and other good stuff.

I try hard to answer all reader e-mails, but sometimes I get behind. Please be patient with me, and I'll do my best to help.

I can't wait to hear from you and see the incredible websites you develop. Have a great time, discover a lot, and stay in touch!

Part I

Creating the HTML Foundation

getting started
with
**HTML5 and
CSS3**



Visit www.dummies.com for more great content online.

Contents at a Glance

Chapter 1: Sound HTML Foundations	9
Creating a Basic Page	9
Understanding the HTML in the Basic Page	11
Meeting Your New Friends, the Tags	12
Setting Up Your System	15
Chapter 2: It's All About Validation	19
Somebody Stop the HTML Madness!	19
Validating Your Page	23
Using Tidy to repair pages	30
Chapter 3: Choosing Your Tools	33
What's Wrong with the Big Boys: Expression Web and Adobe Dreamweaver	33
How About Online Site Builders?	34
Alternative Web Development Tools	35
Picking a Text Editor	35
Finding a Good Web Developer's Browser	44
Chapter 4: Managing Information with Lists and Tables	51
Making a List and Checking It Twice	51
Building Tables	59
Chapter 5: Making Connections with Links	67
Making Your Text Hyper	67
Making Lists of Links	71
Working with Absolute and Relative References	73
Chapter 6: Adding Images, Sound, and Video	77
Adding Images to Your Pages	77
Choosing an Image Manipulation Tool	82
Choosing an Image Format	85
Manipulating Your Images	90
Working with Audio	99
Chapter 7: Creating Forms	105
You Have Great Form	105
Building Text-Style Inputs	109
Making a standard text field	109
Building a password field	111
Creating Multiple Selection Elements	114
Pressing Your Buttons	119
New form input types	122

Chapter 1: Sound HTML Foundations

In This Chapter

- ✓ Creating a basic web page
- ✓ Understanding the most critical HTML tags
- ✓ Setting up your system to work with HTML
- ✓ Viewing your pages

This chapter is your introduction to building web pages. Before this slim chapter is finished, you'll have your first page up and running. It's a humble beginning, but the basic web technology you learn here is the foundation of everything happening on the web today.

In this minibook, you discover the modern form of web design using HTML5. Your web pages will be designed from the ground up, which makes them easy to modify and customize. Although you figure out more advanced techniques throughout this book, you'll take the humble pages you discover in this chapter and make them do all kinds of exciting things.

Creating a Basic Page

Here's the great news: The most important web technology you need is also the easiest. You don't need any expensive or complicated software, and you don't need a powerful computer. You probably have everything you need to get started already.

No more talking! Fire up a computer and build a web page!

1. Open a text editor.

You can use any text editor you want, as long as it lets you save files as plain text. If you're using Windows, Notepad is fine for now. If you're using Mac, you'll really need to download a text editor. I like Komodo Edit (www.activestate.com/komodo-edit) or TextWrangler (www.barebones.com/products/textwrangler/). It's possible to make TextEdit work correctly, but it's probably easier to just download something made for the job. I explain text editors more completely in Chapter 3 of this mini-book.



Don't use a word processor like Microsoft Word or Mac TextEdit. These are powerful tools, but they don't save things in the right format. The way these tools do things like centering text and changing fonts won't work on the web. I promise that you'll figure out how to do all that stuff soon, but a word processing program won't do it correctly. Even the Save as HTML feature doesn't work right. You really need a very simple text editor, and that's it. In Chapter 3 of this minibook, I show you a few more editors that make your life easier. You should not use Word or TextEdit.

2. Type the following code.

Really. Type it in your text editor so you get some experience writing the actual code. I explain very soon what all this means, but type it now to get a feel for it:

```
<!DOCTYPE HTML>
<html lang="en-US">
<head>
<meta charset="UTF-8">
<!-- myFirst.html -->

<title>My very first web page!</title>
</head>

<body>

<h1>This is my first web page!</h1>

<p>
This is the first web page I've ever made,
and I'm extremely proud of it.
It is so cool!
</p>

</body>
</html>
```

3. Save the file as `myFirst.html`.

It's important that your filename has no spaces and ends with the `.html` extension. Spaces cause problems on the Internet (which is, of course, where all good pages go to live), and the `.html` extension is how most computers know that this file is an HTML file (which is another name for a web page). It doesn't matter where you save the file, as long as you can find it in the next step.

4. Open your web browser.

The *web browser* is the program used to look at pages. After you post your page on a web server somewhere, your Great Aunt Gertrude can use her web browser to view your page. You also need one (a browser, not a Great Aunt Gertrude) to test your page. For now, use whatever browser you ordinarily use. Most Windows users already have Internet Explorer installed. If you're a Mac user, you probably have Safari. Linux folks generally have Chrome or Firefox. Any of these are fine. In Chapter 3 of this minibook, I explain why you probably need more than one browser and how to configure them for maximum usefulness.

5. Load your page into the browser.

You can do this a number of ways. You can use the browser's File menu to open a local file, or you can simply drag the file from your Desktop (or wherever) to the open browser window.

6. Bask in your newfound genius.

Your simple text file is transformed! If all went well, it looks like Figure 1-1.

Understanding the HTML in the Basic Page

The page you created in the previous section uses an extremely simple notation — HTML (HyperText Markup Language), which has been around since the beginning of the web. HTML is a terrific technology for several reasons:

- ◆ **It uses plain text.** Most document systems (like word processors) use special *binary encoding schemes* that incorporate formatting directly into the computer's internal language, which locks a document into a particular computer or software. That is, a document stored in Word format can't be read without a program that understands Word formatting. HTML gets past this problem by storing everything in plain text.

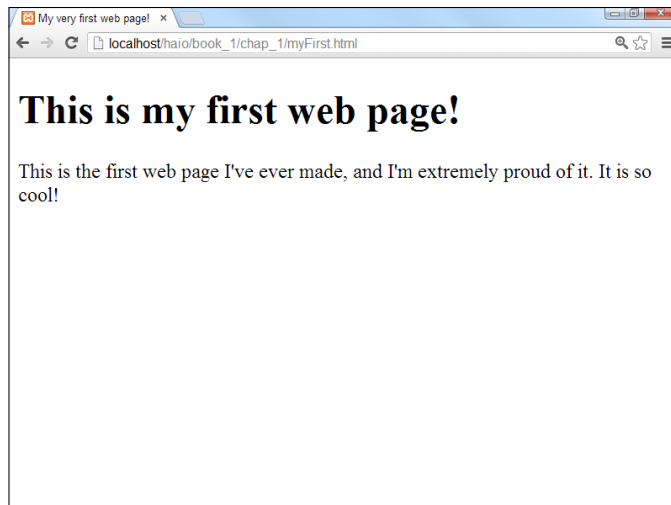


Figure 1-1:
Congratu-
lations!
You're
now a web
developer!

- ◆ **It works on all computers.** The main point of HTML is to have a universal format. Any computer should be able to read and write it. The plain-text formatting aids in this.
- ◆ **It describes what documents *mean*.** HTML isn't really designed to indicate how a page or its elements look. HTML is about describing the meaning of various elements (more on that very soon). This has some distinct advantages when you figure out how to use HTML properly.
- ◆ **It *doesn't* describe how documents *look*.** This one seems strange. Of course, when you look at Figure 1-1, you can see that the appearance of the text on the web page has changed from the way the text looked in your text editor. Formatting a document in HTML does cause the document's appearance to change. That's not the point of HTML, though. You discover in Book II and Book III how to use another powerful technology — CSS — to change the appearance of a page after you define its meaning. This separation of meaning from layout is one of the best features of HTML.
- ◆ **It's easy to write.** Sure, HTML gets a little more complicated than this first example, but you can easily figure out how to write HTML without any specialized editors. You only have to know a handful of elements, and they're pretty straightforward.
- ◆ **It's free.** HTML doesn't cost anything to use, primarily because it isn't owned by anyone. No corporation has control of it (although a couple have tried), and nobody has a patent on it. The fact that this technology is freely available to anyone is a huge advantage.

Meeting Your New Friends, the Tags

The key to writing HTML code is the special text inside angle braces (<>). These special elements are *tags*. They aren't meant to be displayed on the web page, but offer instructions to the web browser about the meaning of the text. The tags are meant to be embedded into each other to indicate the organization of the page. This basic page introduces you to all the major tags you'll encounter. (There are more, but they can wait for a chapter or two.) Each tag has a beginning and an end tag. The end tag is just like the beginning tag, except the end tag has a slash (/):

- ◆ **<!DOCTYPE HTML>:** This special tag is used to inform the browser that the document type is HTML. This is how the browser knows you'll be writing an HTML5 document. You will sometimes see other values for the doctype, but HTML5 is the way to go these days.
- ◆ **<html lang = "en"></html>:** The <html> tag is the foundation of the entire web page. The tag begins the page. Likewise, </html> ends the page. For example, the page begins with <html> and ends with </html>. The <html></html> combination indicates that everything in the page is defined as HTML code. In HTML5, you're expected to tell



the browser which language the page will be written in. Because I write in English, I'm specifying with the code "en."

Some books teach you to write your HTML tags in uppercase letters. This was once a standard, but it is no longer recommended.

- ◆ **<head></head>**: These tags define a special part of the web page called the *head* (or sometimes *header*). This part of the web page reminds me of the engine compartment of a car. This is where you put some great stuff later, but it's not where the main document lives. For now, the only thing you'll put in the header is the document's title. Later, you'll add styling information and programming code to make your pages sing and dance.
- ◆ **<meta charset="UTF-8">**: The meta tag is used to provide a little more information to the browser. This command gives a little more information to the browser, telling it which character set to use. English normally uses a character set called (for obscure reasons) UTF-8. You don't need to worry much about this, but every HTML5 page written in English uses this code.
- ◆ **<!--/-->**: This tag indicates a *comment*, which is ignored by the browser. However, a comment is used to describe what's going on in a particular part of the code.
- ◆ **<title></title>**: This tag is used to determine the page's title. The title usually contains ordinary text. Whatever you define as the title will appear in some special ways. Many browsers put the title text in the browser's title bar. Search engines often use the title to describe the page.

Throughout this book, I use the filename of the HTML code as the title. That way, you can match any figure or code listing to the corresponding file on the web site that accompanies this book. Typically, you'll use something more descriptive, but this is a useful technique for a book like this.



It's not quite accurate to say that the title text always shows up in the title bar because a web page is designed to work on lots of different browsers. Sure, the title does show up on most major browsers that way, but what about cellphones and tablets? HTML never legislates what will happen; it only suggests. This may be hard to get used to, but it's a reality. You trade absolute control for widespread capability, which is a good deal.

- ◆ **<body></body>**: The page's main content is contained within these tags. Most of the HTML code and the stuff the user sees are in the body area. If the header area is the engine compartment, the body is where the passengers go.
- ◆ **<h1></h1>**: H1 stands for *heading level one*. Any text contained within this markup is treated as a prominent headline. By default, most browsers add special formatting to anything defined as H1, but there's no guarantee. An H1 heading doesn't really specify any particular font or formatting, just the *meaning* of the text as a level one heading. When you find out how to use CSS in Book II, you'll discover that you can make your headline look however you want. In this first minibook, keep all the default layouts for now and make sure you understand that HTML is about semantic meaning, not about layout or design. There are other levels of headings, of

course, through `<h6>` where `<h2>` indicates a heading slightly less important than `<h1>`, `<h3>` is less important than `<h2>`, and so on.



Beginners are sometimes tempted to make their first headline an `<h1>` tag and then use an `<h2>` for the second headline and an `<h3>` for the third. That's not how it works. Web pages, like newspapers and books, use different headlines to point out the relative importance of various elements on the page, often varying the point size of the text. You can read more about that in Book II.

- ◆ **`<p></p>`:** In HTML, `p` stands for the paragraph tag. In your web pages, you should enclose each standard paragraph in a `<p></p>` pair. You might notice that HTML doesn't preserve the carriage returns or white space in your HTML document. That is, if you press Enter in your code to move text to a new line, that new line isn't necessarily preserved in the final web page.

The `<p></p>` structure is one easy way to manage spacing before and after each paragraph in your document.



Some older books recommend using `<p>` without a `</p>` to add space to your documents, similar to pressing the Enter key. This way of thinking could cause you problems later because it doesn't accurately reflect the way web browsers work. Don't think of `<p>` as the carriage return. Instead, think of `<p>` and `</p>` as defining a paragraph. The paragraph model is more powerful because soon enough, you'll figure out how to take any properly defined paragraph and give it yellow letters on a green background with daisies (or whatever else you want). If things are marked properly, they'll be much easier to manipulate later.

A few notes about the basic page

Be proud of this first page. It may be simple, but it's the foundation of greater things to come. Before moving on, take a moment to ponder some important HTML principles shown in this humble page you've created:

- ✓ **All tags are lowercase.** Although HTML does allow uppercase tags, modern developers have agreed on lowercase tags in most cases. (`<!DOCTYPE>` is one notable exception to this rule.)
- ✓ **Tag pairs are containers, with a beginning and an end.** Tags contain other tags or text.
- ✓ **Some elements can be repeated.** There's only one `<html>`, `<title>`, and `<body>` tag per page, but a lot of the other elements (`<h1>` and `<p>`) can be repeated as many times as you like.
- ✓ **Carriage returns are ignored.** In the Notepad document, there are a number of carriage returns. The formatting of the original document has no effect on the HTML output. The markup tags indicate how the output looks.

Setting Up Your System

Book I
Chapter 1

Sound HTML
Foundations

You don't need much to make web pages. Your plain text editor and a web browser are about all you need. Still, some things can make your life easier as a web developer.

Displaying file extensions

The method discussed in this section is mainly for Windows users, but it's a big one. Windows uses the *extension* (the part of the filename after the period) to determine what type of file you're dealing with. This is very important in web development. The files you create are simple text files, but if you store them with the ordinary `.txt` extension, your browser can't read them properly. What's worse, the default Windows setting hides these extensions from you, so you have only the icons to tell you what type of file you're dealing with, which causes all kinds of problems. I recommend you have Windows explicitly describe your file extensions. Here's how to set that up in Windows 7:

- 1. Click the Start button.**

This opens the standard Start menu.

- 2. Open the Control Panel.**

The Control Panel application allows you to modify many parts of your operating system.

- 3. Find Appearance and Personalization.**

This section allows you to modify the visual look and feel of your operating system.

- 4. Choose Folder Options.**

This dialog box lets you modify the way folders look throughout the visual interface.

- 5. Find Advanced Settings.**

Click the View tab and then look under Advanced Settings.

- 6. Display file extensions.**

By default, the Hide Extensions for Known File Types check box is selected. Deselect this check box to display file extensions.

The process for displaying file types is similar in Windows 8:

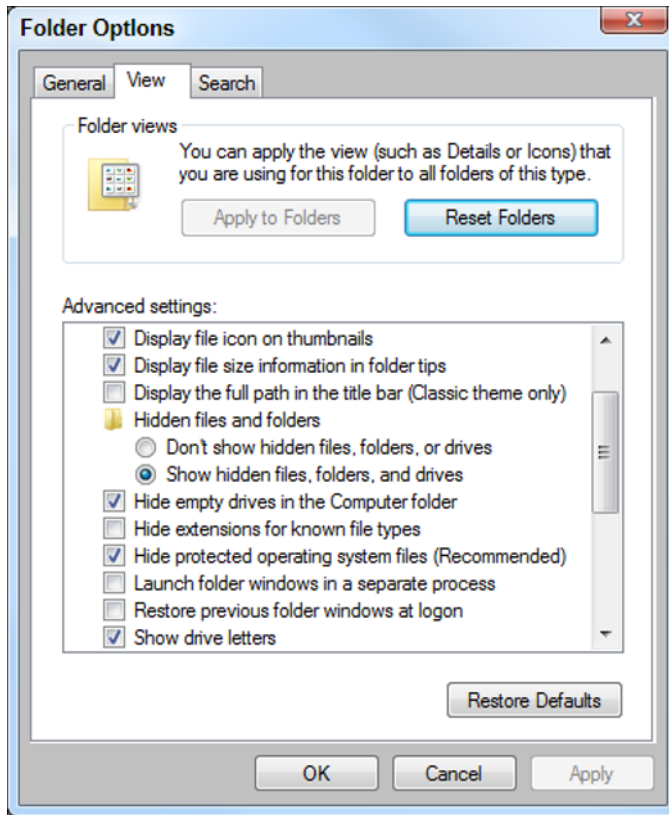
- 1. Go to Windows Explorer.**

Use the Windows Explorer tile to view Windows Explorer — the standard file manager for Windows.

- 2. Click the View tab.**

This tab allows you to modify how directories look.

Figure 1-2:
Don't
hide file
extensions
(deselect
that Hide
Extensions
check box).



3. De-select filename extensions.

If this button is checked, file extensions are shown (which is what you want.) (See Figure 1-2.) Note this is the opposite of Windows 7's behavior.



Although my demonstration uses Windows 7 and 8, the technique is similar in older versions of Windows. Just do a quick search for “displaying file extensions.”

Setting up your software

You'll write a lot of web pages, so it makes sense to set up your system to make that process as easy as possible. I talk a lot more about some software you should use in Chapter 3 of this minibook, but for now, here are a couple of easy suggestions:

- ◆ **Put a Notepad icon on your Desktop.** You'll edit a lot of text files, so it's helpful to have an icon for Notepad (or whatever other text editor you

use) available directly on the Desktop. That way, you can quickly edit any web page by dragging it to the Desktop. When you use more sophisticated editors than Notepad, you'll want links to them, too.

- ◆ **Get another web browser.** You may just *love* your web browser, and that's fine, but you can't assume that everybody likes the same browser you do. You need to know how other browsers interpret your code. Chrome is an incredibly powerful browser, and it's completely free, as well as having a lot of great programmer's features. If you don't already, I suggest having links to at least two browsers directly on your Desktop.

Understanding the magic

Most of the problems people have with the web are from misunderstandings about how this medium really works. Most people are comfortable with word processors, and we know how to make a document look how we want. Modern applications use WYSIWYG technology, promising that *what you see is what you get*. That's a reasonable promise when it comes to print documents, but it doesn't work that way on the web.

How a web page looks depends on a lot of things that you don't control. The user may read your pages on a smaller or larger screen than you. She may use a different operating system than you. She may have a slower connection or may turn off the graphics for speed. She may be blind and use screen-reader technology to navigate web pages. She may be reading your page on a tablet, smart phone,

or even an older (not so smart) cellphone. You can't make a document that looks the same in all these situations.

A good compromise is to make a document that clearly indicates how the information fits together and makes suggestions about the visual design. The user and her browser can determine how much of those suggestions to use.

You get some control of the visual design but never complete control, which is okay because you're trading total control for accessibility. People with devices you've never heard of can visit your page.

Practice a few times until you can easily build a page without looking anything up. Soon enough, you're ready for the next step — building pages like the pros.

Chapter 2: It's All About Validation

In This Chapter

- ✓ Introducing the concept of valid pages
- ✓ Using a doctype
- ✓ Setting the character set
- ✓ Meeting the W3C validator
- ✓ Fixing things when they go wrong
- ✓ Using HTML Tidy to clean your pages

Web development is undergoing a revolution. As the web matures and becomes a greater part of everyday life, it's important to ensure that web pages perform properly — thus, a call for web developers to follow voluntary standards of web development.

Somebody Stop the HTML Madness!

In the bad old days, the web was an informal affair. People wrote HTML pages any way they wanted. Although this was easy, it led to a lot of problems:

- ◆ **Browser manufacturers added features that didn't work on all browsers.** People wanted prettier web pages with colors, fonts, and doodads, but there wasn't a standard way to do these things. Every browser had a different set of tags that supported enhanced features. As a developer, you had no real idea if your web page would work on all the browsers out there. If you wanted to use some neat feature, you had to ensure your users had the right browser.
- ◆ **The distinction between meaning and layout was blurred.** People expected to have some kind of design control of their web pages, so all kinds of new tags popped up that blurred the distinction between describing and decorating a page.
- ◆ **Table-based layout was used as a hack.** HTML didn't have a good way to handle layout, so clever web developers started using tables as a layout mechanism. This worked, after a fashion, but it wasn't easy or elegant.
- ◆ **People started using tools to write pages.** Web development soon became so cumbersome that people began to believe that they couldn't do HTML by hand anymore and that some kind of editor was necessary

to handle all that complexity for them. Although these editing programs introduced new features that made things easier upfront, these tools also made code almost impossible to change without the original editor. Web developers began thinking they couldn't design web pages without a tool from a major corporation.

- ♦ **The nature of the web was changing.** At the same time, these factors were making ordinary web development more challenging. Innovators were recognizing that the web wasn't really about documents but was about applications that could dynamically create documents. Many of the most interesting web pages you visit aren't web pages at all, but programs that produce web pages dynamically every time you visit. This innovation meant that developers had to make web pages readable by programs, as well as humans.
- ♦ **XHTML tried to fix things.** The standards body of the web (there really is such a thing) is called the World Wide Web Consortium (W3C), and it tried to resolve things with a new standard called XHTML. This was a form of HTML that also followed the much stricter rules of XML. If everyone simply agreed to follow the XHTML standard, much of the ugliness would go away.
- ♦ **XHTML didn't work either.** Although XHTML was a great idea, it turned out to be complicated. Parts of it were difficult to write by hand, and very few developers followed the standards completely. Even the browser manufacturers didn't agree exactly on how to read and display XHTML. It doesn't matter how good an idea is if nobody follows it.

In short, the world of HTML was a real mess.

XHTML had some great ideas

In 2000, the World Wide Web Consortium (usually abbreviated as W3C) got together and proposed some fixes for HTML. The basic plan was to create a new form of HTML that complied with a stricter form of markup, or *eXtensible Markup Language (XML)*. The details are long and boring, but essentially, they came up with some agreements about how web pages are standardized. Here are some of those standards:

- ♦ **All tags have endings.** Every tag comes with a beginning and an end tag. (Well, a few exceptions come with their own ending built in. I'll explain when you encounter the first such tag in Chapter 6 of this minibook.) This was a new development because end tags were considered optional in old-school HTML, and many tags didn't even have end tags.
- ♦ **Tags can't be overlapped.** In HTML, sometimes people had the tendency to be sloppy and overlap tags, like this: `<a>my stuff`. That's not allowed in XHTML, which is a good thing because it confuses the browser. If a tag is opened inside some container tag, the tag must be closed before that container is closed.

- ◆ **Everything's lowercase.** Some people wrote HTML in uppercase, some in lowercase, and some just did what they felt like. It was inconsistent and made it harder to write browsers that could read all the variations.
- ◆ **Attributes must be in quotes.** If you've already done some HTML, you know that quotes used to be optional — not anymore. (Turn to Chapter 3 for more about attributes.)
- ◆ **Layout must be separate from markup.** Old-school HTML had a bunch of tags (like `` and `<center>`) that were more about formatting than markup. These were useful, but they didn't go far enough. XHTML (at least the strict version) eliminates all these tags. Don't worry, though; CSS gives you all the features of these tags and a lot more.

This sounds like strict librarian rules, but really they aren't restricting at all. Most of the good HTML coders were already following these guidelines or something similar.

Even though you're moving past XHTML into HTML5, these aspects of XHTML remain, and they are guidelines all good HTML5 developers still use.



HTML5 actually allows a looser interpretation of the rules than XHTML strict did, but throughout this book I write HTML5 code in a way that also passes most of the XHTML strict tests. This practice ensures nice clean code with no surprises.

You validate me

In old-style HTML, you never really knew how your pages would look on various browsers. In fact, you never really knew if your page was even written properly. Some mistakes would look fine on one browser but cause another browser to blow up.

The idea of *validation* is to take away some of the uncertainty of HTML. It's like a spell checker for your code. My regular spell checker makes me feel a little stupid sometimes because I make mistakes. I like it, though, because I'm the only one who sees the errors. I can fix the spelling errors before I pass the document on to you, so I look smart. (Well, maybe.)

It'd be cool if you could have a special kind of checker that does the same things for your web pages. Instead of checking your spelling, it'd test your page for errors and let you know if you made any mistakes. It'd be even cooler if you could have some sort of certification that your page follows a standard of excellence.

That's how page validation works. You can designate that your page will follow a particular standard and use a software tool to ensure that your page meets that standard's specifications. The software tool is a *validator*. I show you two different validators in the upcoming "Validating Your Page" section.

The browsers also promise to follow a particular standard. If your page validates to a given standard, any browser that validates to that same standard can reproduce your document correctly, which is a big deal.

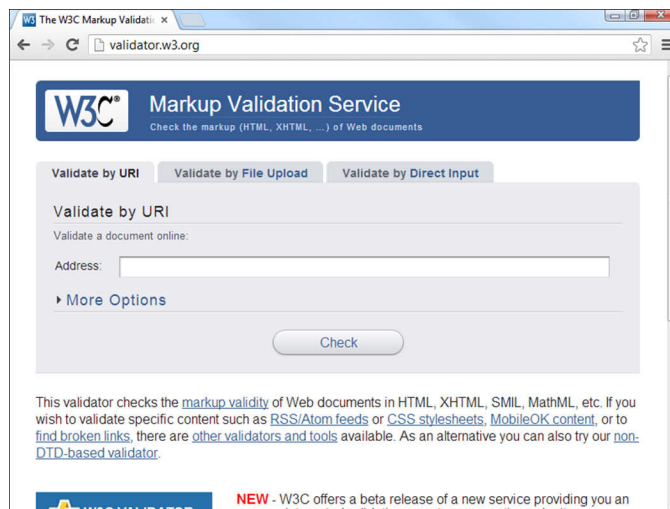
The most important validator is the W3C validator at <http://validator.w3.org>, as shown in Figure 2-1.

A validator is actually the front end of a piece of software that checks pages for validity. It looks at your web page's doctype and sees whether the page conforms to the rules of that doctype. If not, it tells you what might have gone wrong.

You can submit code to a validator in three ways:

- ◆ **Validate by URI.** This option is used when a page is hosted on a web server. Files stored on local computers can't be checked with this technique. Book VIII describes all you need to know about working with web servers, including how to create your own and move your files to it. (A *URI*, or uniform resource identifier, is a more formal term for a web address, which is more frequently seen as URL.)
- ◆ **Validate by file upload.** This technique works fine with files you haven't posted to a web server. It works great for pages you write on your computer but that you haven't made visible to the world. This is the most common type of validation for beginners.
- ◆ **Validate by direct input.** The validator page has a text box you can simply paste your code into. It works, but I usually prefer to use the other methods because they're easier.

Figure 2-1:
The W3C
validator
page isn't
exciting,
but it sure is
useful.



Validation might sound like a big hassle, but it's really a wonderful tool because sloppy HTML code can cause lots of problems. Worse, you might think everything's okay until somebody else looks at your page, and suddenly, the page doesn't display correctly.



As of this writing, the W3C validator can read and test HTML5 code, but the HTML5 validation is still considered experimental. Until HTML5 becomes a bit more mainstream, your HTML5 pages may get a warning about the experimental nature of HTML5. You can safely ignore this warning.

Validating Your Page

To explain all this, I created a web page the way Aesop might have done in ancient Greece. Okay, maybe Aesop didn't write his famous fables as web pages, but if he had, they might have looked like the following code listing:

```
<!DOCTYPE HTML>
<html lang="en-US">
<head>
  <meta charset="UTF-8">

<!-- oxWheels1.html -->

<!-- note this page has deliberate errors! Please see the text
      and oxWheelsCorrect.html for a corrected version.
-->

</head>
<body>
<title>The Oxen and the Wheels</title>
<h1>The Oxen and the Wheels
<h2></h1>From Aesop's Fables</h2>

<p>
  A pair of Oxen were drawing a heavily loaded wagon along a
  miry country road. They had to use all their strength to pull
  the wagon, but they did not complain.
</p>

<p>
  The Wheels of the wagon were of a different sort. Though the
  task they had to do was very light compared with that of the
  Oxen, they creaked and groaned at every turn. The poor Oxen,
  pulling with all their might to draw the wagon through the
  deep mud, had their ears filled with the loud complaining of
  the Wheels. And this, you may well know, made their work so
  much the harder to endure.
</p>

<p>
  "Silence!" the Oxen cried at last, out of patience. "What have
  you Wheels to complain about so loudly? We are drawing all the
  weight, not you, and we are keeping still about it besides."
</p>

<h2>
  They complain most who suffer least.
```

```
</h2>

</body>
</html>
```

The code looks okay, but actually has a number of problems. Aesop may have been a great storyteller, but from this example, it appears he was a sloppy coder. The mistakes can be hard to see, but trust me, they're there. The question is, how do you find the problems before your users do?

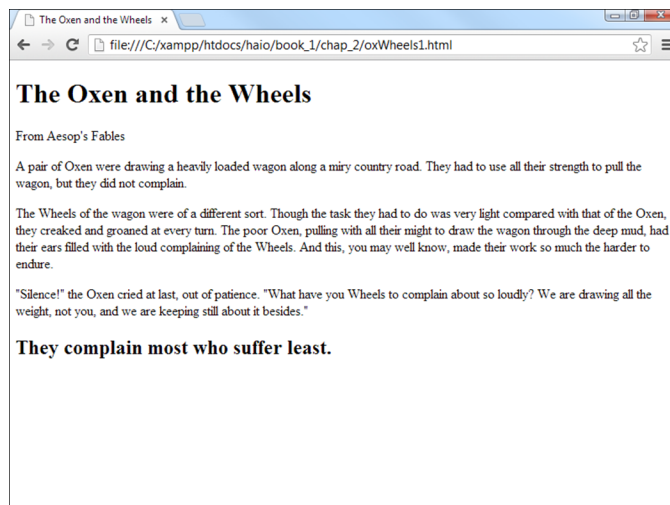
You might think that the problems would be evident if you viewed the page in a web browser. The various web browsers seem to handle the page decently, even if they don't display it in an identical way. Figure 2-2 shows oxWheels1.html in a browser.

Chrome appears to handle the page pretty well, but From Aesop's Fables is supposed to be a headline level two, or *H2*, and it appears as plain text. Other than that, there's very little indication that something is wrong.

If it looks fine, who cares if it's exactly right? You might wonder why we care if there are mistakes in the underlying code, as long as everything works okay. After all, who's going to look at the code if the page displays properly?

The problem is, you don't know if it'll display properly, and mistakes in your code will eventually come back to haunt you. If possible, you want to know immediately what parts of your code are problematic so you can fix them and not worry.

Figure 2-2:
The page
looks okay,
but the
headings
are strange.



Aesop visits W3C

To find out what's going on with this page, pay a visit to the W3C validator at <http://validator.w3.org>. Figure 2-3 shows me visiting this site and uploading a copy of `oxWheels1.html` to it.

Hold your breath and click the Check button. You might be surprised at the results shown in Figure 2-4.

The validator is a picky beast, and it doesn't seem to like this page at all. The validator does return some useful information and gives enough hints that you can decode things soon enough.

Figure 2-3:
I'm
checking
the
oxWheels
page to
look for any
problems.

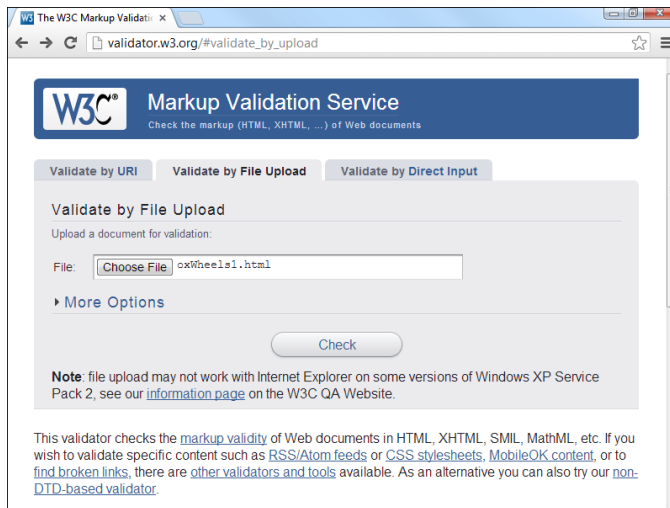
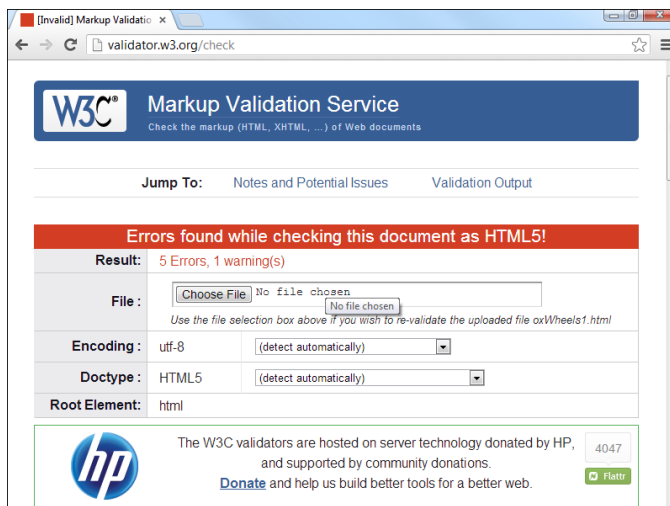


Figure 2-4:
Five errors?
That can't
be right!



Examining the overview

Before you look at the specific complaints, take a quick look at the web page the validator sends you. The web page is chock-full of handy information. The top of the page tells you a lot of useful things:

- ◆ **Result:** This is really the important thing. You'll know the number of errors remaining by looking at this line. Don't panic, though. The errors in the document are probably fewer than the number you see here.
- ◆ **File:** The name of the file you're working on.
- ◆ **Encoding:** The text encoding you've set. If you didn't explicitly set text encoding, you may see a warning here.
- ◆ **Doctype:** This is the doctype extracted from your document. It indicates the rules that the validator is using to check your page. This should usually say `HTML5`.
- ◆ **The dreaded red banner:** Experienced web developers don't even have to read the results page to know if there is a problem. If everything goes well, there's a green congratulatory banner. If there are problems, the banner is red. It doesn't look good, Aesop.



Don't panic because you have errors. The mistakes often overlap, so one problem in your code often causes more than one error to pop up. Most of the time, you have far fewer errors than the page says, and a lot of the errors are repeated, so after you find the error once, you'll know how to fix it throughout the page.

Validating the page

The validator doesn't always tell you everything you need to know, but it does give you some pretty good clues. Page validation is tedious but not as difficult as it might seem at first. Here are some strategies for working through page validation:

- ◆ **Focus only on the first error.** Sure, 100 errors might be on the page, but solve them one at a time. The only error that matters is the first one on the list. Don't worry at all about other errors until you've solved the first one.
- ◆ **Note where the first error is.** The most helpful information you get is the line and column information about where the validator recognized the error. This isn't always where the error is, but it does give you some clues.
- ◆ **Look at the error message.** It's usually good for a laugh. The error messages are sometimes helpful and sometimes downright mysterious.
- ◆ **Look at the verbose text.** Unlike most programming error messages, the W3C validator tries to explain what went wrong in something like English. It still doesn't always make sense, but sometimes the text gives you a hint.

- ◆ **Scan the next couple of errors.** Sometimes, one mistake shows up as more than one error. Look over the next couple of errors, as well, to see if they provide any more insight; sometimes, they do.
- ◆ **Try a change and revalidate.** If you've got an idea, test it out (but only solve one problem at a time.) Check the page again after you save it. If the first error is now at a later line number than the previous one, you've succeeded.
- ◆ **Don't worry if the number of errors goes up.** The number of perceived errors will sometimes go up rather than down after you successfully fix a problem. This is okay. Sometimes, fixing one error uncovers errors that were previously hidden. More often, fixing one error clears up many more. Just concentrate on clearing errors from the beginning to the end of the document.
- ◆ **Lather, rinse, and repeat.** Look at the new top error and get it straightened out. Keep going until you get the coveted Green Banner of Validation. (If I ever write an HTML adventure game, the Green Banner of Validation will be one of the most powerful talismans.)

Examining the first error

Look again at the results for the oxWheels1.html page. The first error message looks like Figure 2-5.

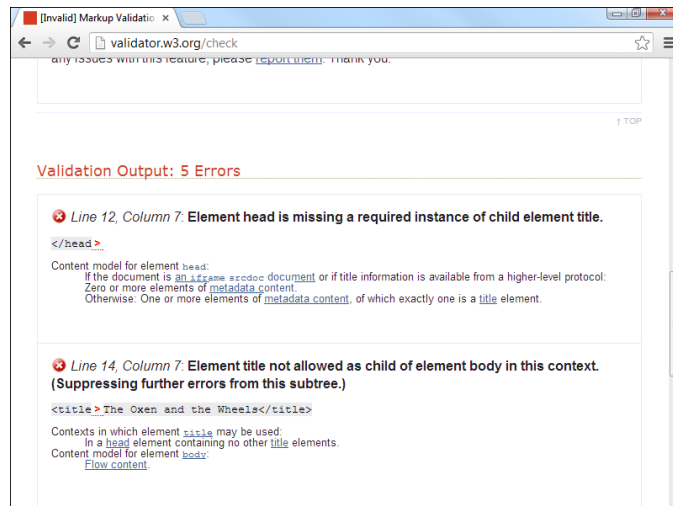


Figure 2-5:
Well, that
clears
every-
thing up.

Figure 2-5 shows the first two error messages. The first complains that the head is missing a title. The second error message is whining about the title being in the body. The relevant code is repeated here:

```
<!DOCTYPE HTML>
<html lang="en-US">
```

```
<head>
  <meta charset="UTF-8">

<!-- oxWheels1.html -->

<!-- note this page has deliberate errors! Please see the text
      and oxWheelsCorrect.html for a corrected version.
-->

</head>
<body>
<title>The Oxen and the Wheels</title>
```

Look carefully at the `head` and `title` tag pairs and review the notes in the error messages, and you'll probably see the problem. The `<title>` element is supposed to be in the heading, but I accidentally put it in the body! (Okay, it wasn't accidental; I made this mistake deliberately here to show you what happens. However, I have made this mistake for real in the past.)

Fixing the title

If the `title` tag is the problem, a quick change in the HTML should fix it. `oxWheels2.html` shows another form of the page with my proposed fix:

```
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />

<!-- oxWheels2.html -->

<!-- Moved the title tag inside the header -->
<title>The Oxen and the Wheels</title>

</head>
<body>
```

Note: I'm only showing the parts of the page that I changed. The entire page is available on this book's website. See this book's Introduction for more on the website.

The fix for this problem is pretty easy:

1. **Move the title inside the head.** I think the problem here is having the `<title>` element inside the body, rather than in the head where it belongs. If I move the title to the body, the error should be eliminated.
2. **Change the comments to reflect the page's status.** It's important that the comments reflect what changes I make.
3. **Save the changes.** Normally, you simply make a change to the same document, but I've elected to change the filename so you can see an archive of my changes as the page improves. This can actually be a good idea because you then have a complete history of your document's changes, and you can always revert to an older version if you accidentally make something worse.

4. **Note the current first error position.** Before you submit the modified page to the validator, make a mental note of the position of the current first error. Right now, the validator's first complaint is on line 12, column 7. I want the first mistake to be somewhere later in the document.
5. **Revalidate by running the validator again on the modified page.**
6. **Review the results and do a happy dance.** It's likely you still have errors, but that's not a failure! Figure 2-6 shows the result of my revalidation. The new first error is on line 17, and it appears to be very different from the last error. I solved it!

Solving the next error

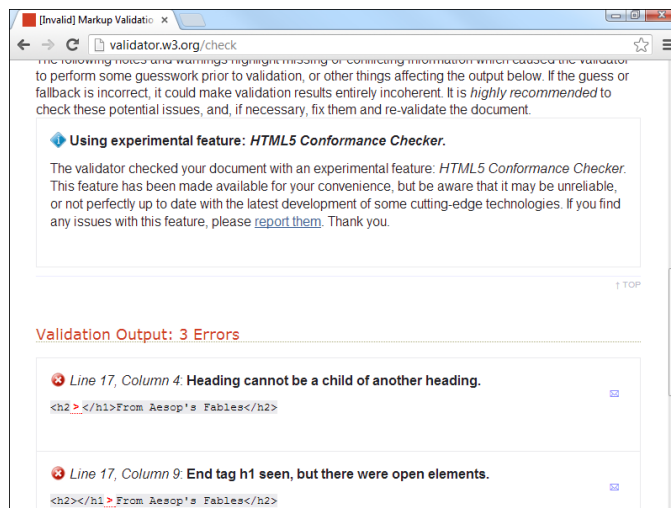
One down, but more to go. The next error (refer to Figure 2-6) looks strange, but it makes sense when you look over the code.

This type of error is very common. What it usually means is you forgot to close something or you put something in the wrong place. The error message indicates a problem in line 17. The next error is line 17, too. See if you can find the problem here in the relevant code:

```
<body>
<h1>The Oxen and the Wheels
<h2></h1>From Aesop's Fables</h2>
```

After you know where to look, the problem becomes a bit easier to spot. I got sloppy and started the `<h2>` tag before I finished the `<h1>`. In many cases, one tag can be completely embedded inside another, but you can't have tag definitions overlap as I've done here. The `<h1>` has to close before I can start the `<h2>` tag.

Figure 2-6:
Heading
cannot be
a child of
another
heading.
Huh?



This explains why browsers might be confused about how to display the headings. It isn't clear whether this code should be displayed in H1 or H2 format, or perhaps with no special formatting at all. It's much better to know the problem and fix it than to remain ignorant until something goes wrong.

The third version — `oxWheels3.html` — fixes this part of the program:

```
<!-- oxWheels3.html -->
<!-- sort out the h1 and h2 tags at the top -->
<title>The Oxen and the Wheels</title>
</head>
<body>
<h1>The Oxen and the Wheels</h1>
<h2>From Aesop's Fables</h2>
```

The validator has fixed a number of errors, but there's one really sneaky problem still in the page. See if you can find it, and then read ahead.

Using Tidy to repair pages

The W3C validator isn't the only game in town. Another great resource — HTML Tidy — can be used to fix your pages. You can download Tidy or just use the online version at <http://infohound.net/tidy>. Figure 2-7 illustrates the online version.

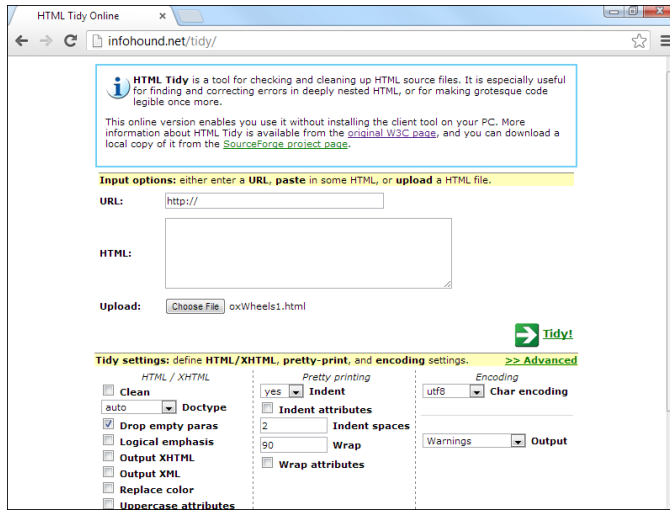
Is validation really that big a deal?

I can hear the angry e-mails coming in. "Andy, I've been writing web pages since 1998, and I never used a validator." Okay, it's true. A lot of people, even some professional web developers, work without validating their code. Some of my older web pages don't validate at all. (You can run the W3C validator on any page you want, not just one you wrote. This can be a source of great joy if you like feeling superior to sloppy coders.) When I became more proficient and more prolific in my web development, I found that those little errors often caused a whole lot of grief down the road. I really believe you should validate every single page you write. Get into the habit now, and it'll pay huge dividends. When you're figuring out this stuff for the first time, do it right.

If you already know some HTML, you're gonna hate the validator for a while because it rejects coding habits that you might think are perfectly fine. Unlearning a habit is a lot harder than learning a new practice, so I feel your pain. It's still worth it.

After you discipline yourself to validate your pages, you'll find you've picked up good habits, and validation becomes a lot less painful. Experienced programmers actually like the validation process because it becomes much easier and prevents problems that could cause lots of grief later. You may even want to re-validate a page you've been using for a while. Sometimes a content update can cause mistakes.

Figure 2-7:
HTML
Tidy is an
alternative
to the W3C
validator.

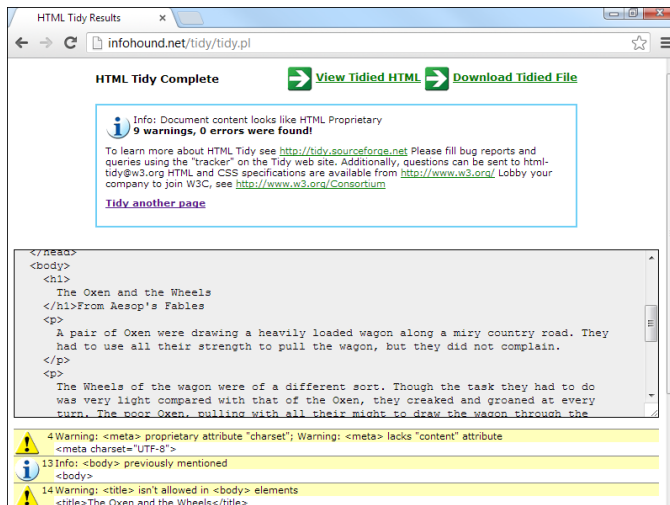


Unlike W3C's validator, Tidy actually attempts to fix your page. Figure 2-8 displays how Tidy suggests the oxWheels1.html page be fixed.

Tidy examines the page for a number of common errors and does its best to fix the errors. However, the result is not quite perfect:

- ◆ **It outputs XHTML by default.** XHTML is fine, but because we're doing HTML here, deselect the Output XHTML box. The only checkbox you need selected is Drop Empty Paras.

Figure 2-8:
Tidy fixes
the page,
but the fix
is a little
awkward.



- ◆ **Tidy got confused by the headings.** Tidy correctly fixed the level one heading, but it had trouble with the level two heading. It removed all the tags, so it's valid, but the text intended to be a level two heading is just sort of hanging there.
- ◆ **Sometimes, the indentation is off.** I set Tidy to indent every element, so it is easy to see how tag pairs are matched up. If I don't set up the indentation explicitly, I find Tidy code very difficult to read.
- ◆ **The changes aren't permanent.** Anything Tidy does is just a suggestion. If you want to keep the changes, you need to save the results in your editor. Click the Download Tidied File button to do this easily.

I sometimes use Tidy when I'm stumped because I find the error messages are easier to understand than the W3C validator. However, I never trust it completely. Until it's updated to truly understand HTML5, it sometimes deletes perfectly valid HTML5 tags. There's really no substitute for good old detective skills and the official W3C validator.

Did you figure out that last error? I tried to close a paragraph with `<p>` rather than `</p>`. That sort of thing freaks out an XHTML validator, but HTML takes it in stride, so you might not even know there is a problem. Tidy does notice the problem and repairs it. Remember this when you're working with a complex page and something doesn't seem right. It's possible there's a mistake you can't even see, and it's messing you up. In that case, consider using a validator and Tidy to figure out what's going wrong and fix it.

Chapter 3: Choosing Your Tools

In This Chapter

- ✓ **Choosing a text editor**
- ✓ **Using a dedicated HTML editor**
- ✓ **Comparing common browsers**

Web development is a big job. You don't go to a construction site without a belt full of tools (and a cool hat), and the same thing is true with web development (except you don't normally need a hard hat for web development). An entire industry has evolved trying to sell tools that help make web development easier. The funny thing is that the tools you need might not be the ones that people are trying to sell you. Some of the very best web development tools are free, and some of the most expensive tools aren't that helpful.

This chapter tells you what you need and how to set up your workshop with great programs that simplify web development.

What's Wrong with the Big Boys: Expression Web and Adobe Dreamweaver

Many web development books are really books about how to use a particular type of software. Microsoft's Expression Web and Adobe Dreamweaver are the two primary applications in this category. These tools are powerful and offer some *seemingly* great features:

- ◆ **WYSIWYG editing:** *What you see is what you get* is an idea borrowed from word processors. You can create a web page much like a word-processing document and use menus as well as tools to handle all the formatting. The theory is that you don't have to know any icky codes.
- ◆ **Templates:** You can create a template that stays the same and build several pages from that template. If you need to change the template, everything else changes automatically.
- ◆ **Site management:** The interaction between the various pages on your site can be maintained automatically.

These sound like pretty good features, and they are. The tools (and the newer replacements, like Microsoft's Expression suite) are very powerful and can be an important part of your web development toolkit. However, the same powerful programs introduce problems, such as the following:

- ◆ **Code maintenance:** The commercial editors that concentrate on visual design tend to create pretty unmanageable code. If you find there's something you need to change by hand, it's pretty hard to fix the code.
- ◆ **Vendor lock-in:** These tools are written by corporations that want you to buy other tools from them. If you're using Dreamweaver, you'll find it easy to integrate with other Adobe applications (like ColdFusion), but it's not as simple to connect to non-Adobe technology. Likewise, Microsoft's offerings are designed to work best with other Microsoft technologies.
- ◆ **Cost:** The cost of these software packages keeps going up. Although there are free versions of Microsoft's web development tools, the commercial versions are very expensive. Likewise, Dreamweaver weighs in at \$400. Both companies encourage you to buy the software as part of a package, which can easily cost more than hundreds more.
- ◆ **Complexity:** They're complicated. You can take a full class or buy a huge book on how to use only one of these technologies. If it's that hard to figure out, is it really saving you any effort?
- ◆ **Code:** You still need to understand it. No matter how great your platform is, at some point, you have to dig into your code. After you plunk down all that money and spend all that time figuring out an application, you still have to understand how the underlying code works because things still go wrong. For example, if your page fails to work with Safari, you'll have to find out why and fix the problem yourself.
- ◆ **Spotty standards compliance:** The tools are getting better here, but if you want your pages to comply with the latest standards, you have to edit them heavily after the tool is finished.
- ◆ **Display variations:** WYSIWYG is a lie. This is really the big problem. WYSIWYG works for word processors because it's possible to make the screen look like the printed page. After a page is printed, it stays the same. You don't know what a web page will look like because that depends on the browser. What if the user loads your page on a cell-phone or handheld device? The editors tend to perpetuate the myth that you can treat a web page like a printed document when, in truth, it's a very different kind of beast.
- ◆ **Incompatibility with other tools:** Web development is now moving toward content management systems (CMS) — programs that create websites dynamically. Generally, CMS systems provide the same ease-of-use as a visual editor but with other benefits. However, transitioning code created in a commercial editor to a CMS is very difficult. I describe CMS systems in detail in Book VIII.

How About Online Site Builders?

A lot of modern websites are built with a content management system (CMS). Content management systems are software programs that allow you to build and modify a page right in your web browser. Some CMS systems are free, and

some cost money to use. I go over how to install and modify a CMS (and even build your own) in Book VIII. A CMS system can be nice because it allows you to build a website visually without any special tools or knowledge.

The CMS approach is a very good solution, but I still recommend you discover how to build things by hand. Ultimately even a CMS uses HTML and CSS, and you'll need these skills to make your site look and perform well even if you have help.

Alternative Web Development Tools

For web development, all you really need is a text editor and a web browser. You probably already have a basic set of tools on your computer. If you read Chapters 1 and 2 of this minibook, you've already written a couple of web pages. However, the very basic tools that come with every computer might not be enough for serious work. Web development requires a specialized kind of text editor, and a number of tools have evolved that make the job easier.

I've found uses for four types of programs in web development:

- ◆ **Enhanced text editors:** These tools are text editors, but they're souped-up with all kinds of fancy features, like syntax checkers, code-coloring tools, macro tools, and multiple document interfaces.
- ◆ **Browsers and plug-ins:** Some browsers are better than others for development. You'll also need a full suite of browsers to ensure your code works in all of them. Some browsers can be extended with plug-ins for advanced performance.
- ◆ **Programming technologies:** This book covers all pertinent info about incorporating other technologies, like Apache, PHP, and MySQL. I show you how to install everything you need for these technologies in Book VIII, Chapter 1. You don't need to worry about these things yet, but you should develop habits that are compatible with these enhanced technologies from the beginning.
- ◆ **Multimedia tools:** It's very common for a web page to feature various types of images, as well as other multimedia like custom fonts, sound effects, and video. You'll need some tools to manage these resources.

Picking a Text Editor

As a programmer, you come to see your text editor as a faithful companion. You spend a lot of time with this tool, so use one that works with you.

A text editor should save plain text without any formatting at all. You don't want anything that saves colors, font choices, or other text formatting because these things don't automatically translate to HTML.

Fortunately, you have several choices, as the following sections reveal.

Tools to avoid unless you have nothing else

A text editor may be a simple program, but that doesn't mean they're all the same. Some programs have a history of causing problems for beginners (and experienced developers, too). There's usually no need to use some of these weaker choices.



Microsoft Word

Just don't use it for web development. Word is a word processor. Even though, theoretically, it can create web pages, the HTML code it writes is absolutely horrific. As an example, I created a blank document, wrote "Hello World" in it, changed the font, and saved it as HTML. The resulting page was non-compliant code, was not quite HTML or XHTML, and was 114 lines long. Word is getting better, but it's just not a good web development tool. In fact, don't use any word processor. They're just not designed for this kind of work.

Windows Notepad

Notepad is everywhere, and it's free. That's the good news. However, Notepad doesn't have a lot of the features you might need, such as line numbers, multiple documents, or macros. Use it if you're on an unfamiliar machine, but try something else if you can. Many people begin with Notepad, but it won't be long until you outgrow its limitations.

Mac TextEdit

Mac has a simple text editor built in — TextEdit — that's similar to Notepad, but closer to a word processor than a programmer's text editor. TextEdit saves files in a number of formats. If you want to use it to write web pages, you must save your files in plain-text format, and you must not use any of TextEdit's formatting features. It's probably best not to use TextEdit unless you really have to.

Suggested programmer's editors

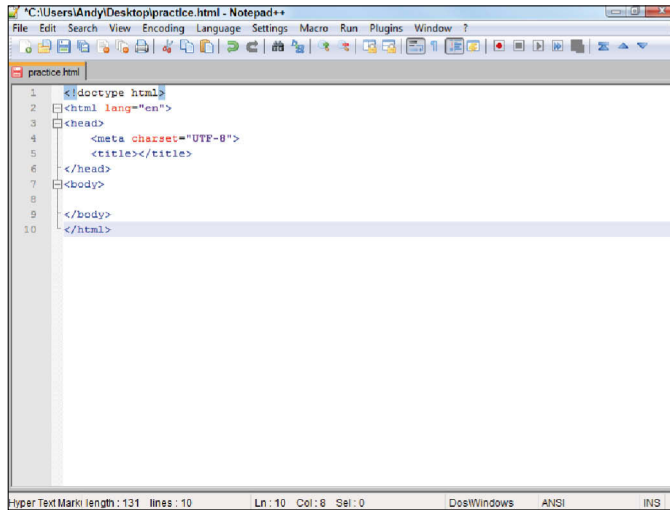
If Notepad, Word, and TextEdit aren't the best choices, what are some better options?

Good question. Because a text editor is such an important tool, it might depend a bit on your preferences, so I'll highlight a few of my favorites. Note that every editor I mention here is entirely free, so don't go paying for something until you've tried some of these first.

A noteworthy editor: Notepad++

A number of developers have come up with good text editors. Some of the best are free, such as Notepad++ by Don Ho. Notepad++ is designed for text editing, especially in programming languages. Figure 3-1 shows Notepad++ with an HTML file loaded.

Figure 3-1:
Notepad++
has many of
the features
you need in
a text editor.



Notepad++ has a lot of interesting features. Here are a few highlights:

- ◆ **Syntax highlighting:** Notepad++ can recognize key HTML terms and put different types of terms in different colors. For example, all HTML tags are rendered blue, and text is black, making it easy to tell if you've made certain kinds of mistakes, such as forgetting to end a tag. Note that the colors aren't saved in the document. The coloring features are there to help you understand the code.
- ◆ **Multiple files:** You'll often want to edit more than one document at a time. You can have several different documents in memory at the same time.
- ◆ **Multi-language support:** Currently, your pages consist of nothing but HTML. Soon enough, you'll use some other languages, like SQL, CSS, and PHP. Notepad++ is smart enough to recognize these languages, too.
- ◆ **Macros:** Whenever you find yourself doing something over and over, consider writing a keyboard macro. Notepad++ has a terrific macro feature. Macros are easy to record and play back a series of keystrokes, which can save you a lot of work.
- ◆ **Page preview:** When you write a page, test it. Notepad++ has short-cut keys built in to let you quickly view your page in Internet Explorer (Ctrl+Alt+Shift+I) and Firefox (Ctrl+Alt+Shift+X).
- ◆ **TextFX:** The open-source design of Notepad++ makes it easy to add features. The TextFX extension (built into Notepad++) allows you to do all sorts of interesting things. One especially handy set of tools runs HTML Tidy on your page and fixes any problems.



Sadly, Notepad++ is a Windows-only editor. If you're using Mac or Linux, you need to find something else. The closest alternative in the Mac and Linux world is gedit.

gedit

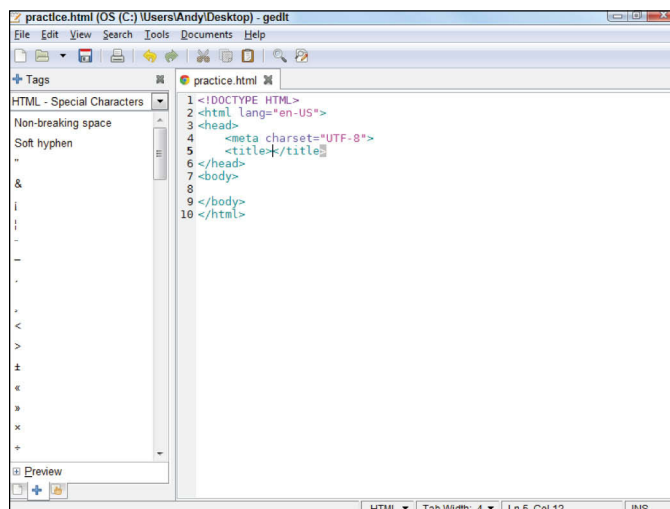
One simple but effective editor available free for all major operating systems is gedit. It is the default editor for many versions of Linux, but you can download it for Mac and Windows from <http://projects.gnome.org/gedit/>.

It has all the standard features including syntax highlighting (which colors different parts of code in different colors to help with debugging), line numbers, and a tag list, which is a special menu which allows you to pick common HTML tags from a list if you forget some syntax. (You may need to play with the plugins from the edit-preferences menu to activate all these features.)

Sadly, gedit does not have a macro editor. This may not be a deal-breaker for you, but often I find a macro tool to be extremely useful, and I'm happiest when my editor has this feature. (If you're especially geeky, it does expose the entire Python language and allow you to modify anything with this language, but that's a topic for another day.) If you need a very nice general-purpose editor, consider gedit. It does much of what you might want without getting terribly complicated.

Figure 3-2 shows gedit in action.

Figure 3-2:
gedit is a
very nice
but simple
tool.



The old standards: VI and Emacs

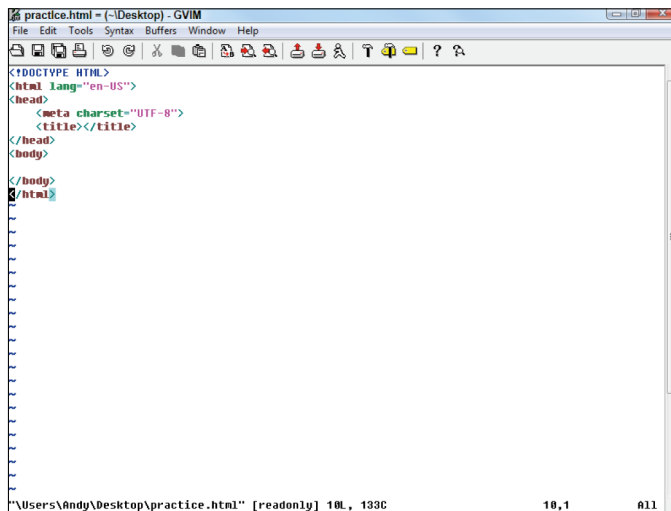
No discussion of text editors is complete without a mention of the venerable UNIX editors that were the core of the early Internet experience. Most of the pioneering work on the web was done in the UNIX and Linux operating systems, and these environments had two extremely popular text-editor families.

Both might seem obscure and difficult to modern sensibilities, but they still have passionate adherents, even in the Windows community. (Besides, Linux is more popular than ever!)

VI and VIM

VI stands for Visual Editor. That name seems strange now because most developers can't imagine an editor that's *not* visual. Back in the day, it was a very big deal that VI could use the entire screen for editing text. Before that time, line-oriented editors were the main way to edit text files. Trust me, you have it good now. Figure 3-3 shows a modern variant of VI (called GVIM) in action.

Figure 3-3:
VI isn't
pretty, but
after you
know it,
it's very
powerful.



VI is a *modal* editor, which means that the same key sometimes has more than one job, depending on the editor's current mode. For example, the I key is used to indicate where you want to insert text. The D key is used to delete text, and so on. Of course, when you're inserting text, the keys have their normal meanings. This multimode behavior is baffling to modern users, but it can be amazingly efficient after you get used to it. Skilled VI users swear by it and often use nothing else.

VI is a little too obscure for some users, so a number of variants are floating around, such as VIM, or VI Improved. (Yeah, it should be VII but maybe they were afraid people would call it the Roman numeral seven.) VIM is a little friendlier than VI, and GVIM is friendlier yet. It tells you which mode it's in and includes such modern features as mouse support, menus, and icons. Even with these features, VIM is not intuitive for most people.

Versions of VI are available for nearly any operating system being used. If you already know VI, you might enjoy using it for web page development

because it has all the features you might need. If you don't already know VI, it's probably more efficient for you to start with a more standard text editor, such as Notepad++.

Emacs

The other popular editor from the UNIX world is Emacs. Like VI, you probably don't need this tool if you never use Linux or UNIX. Also like VI, if you know it already, you probably don't need anything else. Emacs has been a programmer's editor for a very long time (it has been in continuous development since 1976) and has nearly every feature you can think of.



Emacs also has a lot of features you haven't thought of, including a built-in text adventure game and even a psychotherapist simulator. I really couldn't make this stuff up if I tried.

Emacs has very powerful customization and macro features and allows you to view and edit more than one file at a time. Emacs also has the ability to view and manipulate the local file system, manage remote files, access the local operating system (OS) shell, and even browse the web or check e-mail without leaving the program. If you're willing to invest in a program that takes some effort to understand, you'll have an incredibly powerful tool in your kit. Versions of Emacs are available for most major operating systems. Emacs is one of the first programs I install on any new computer because it's so powerful. A version of Emacs is shown in Figure 3-4.

An enhanced version — XEmacs — (shown in the figure) uses standard menus and icons like modern programs, so it's reasonably easy to get started with.

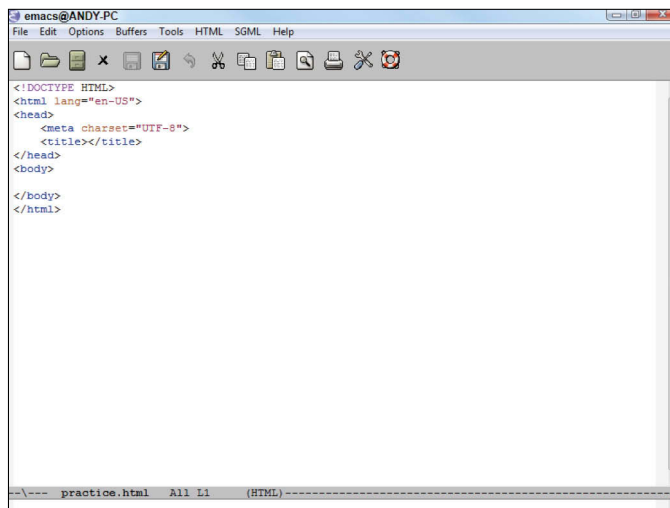


Figure 3-4:
Emacs is
powerful but
somewhat
eccentric.



Emacs has an astonishing number of options and a nonstandard interface, so it can be challenging for beginners. However, those who have made the investment (like me) swear by it.

My personal choice: Komodo Edit

Personally I really like Komodo Edit (www.activestate.com/komodo-edit). This editor is extremely powerful, but is not quite as intimidating as some of the older tools. It has a modern streamlined interface, but more power than you might realize at first. Komodo Edit is actually the open-source cousin to a commercial Integrated Development Environment (IDE) called Komodo IDE. Komodo IDE costs hundreds of dollars, but Komodo Edit has almost as many features, and is entirely free. Figure 3-5 illustrates Komodo Edit.

Komodo Edit has a number of really intriguing features that make it stand out in my mind:

- ◆ **All the standard features:** Komodo Edit has all the features I've mentioned as necessary for a programmer's editor, including syntax highlighting, line numbers, and saving in plain text format.
- ◆ **Code completion:** A number of higher-end programmer's editors have this feature, but it's not as common in text editors. Here's how it works: When you set up a page as HTML5 (by choosing from the menu on the bottom right), Komodo "watches" as you type and provides hints. So, if you begin typing `<h`, Komodo pops up a little dialog box showing all the tags that begin with `h`. If you pick `<html>` and then move to the next line and type an angle bracket (`<`) character, you'll get a pop-up menu with `<head>` and `<body>` listed because these are the two tags valid in this context. Komodo is pretty smart about knowing what tags you can use when. This can be a helpful feature when you're starting out.

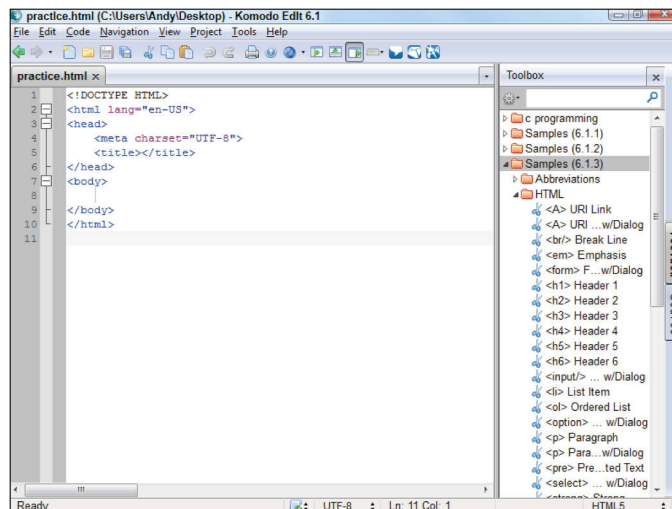


Figure 3-5:
Komodo Edit
is a really
powerful
editor.

- ◆ **Multiple file support:** Your first few web pages will be single documents, but most websites incorporate many pages. Komodo allows you to have several pages at once and to compare any two pages at the same time.
- ◆ **Page Preview:** Just use `ctrl-K-V` to preview the current web page in a second tab. This is a quick way to see how your page is going.
- ◆ **Multiple language support:** This book (and web development in general) requires a whole bunch of different languages. Komodo Edit is just as good at the languages you'll be using as it is with HTML. Komodo has native support for HTML, CSS, JavaScript, PHP, MySQL and many more. (In fact, I also use it for working in other languages like Python, C++, and Java, so you might end up using it beyond even web development.)
- ◆ **Multi-platform:** It might not be a big deal to you right now, but Komodo works the same on all major operating systems – Windows, Mac, and Linux. This really matters in web development because you will encounter new operating systems in your web travels. I use all three major OS types and use Komodo on all of them.
- ◆ **Remote file support:** Eventually, you'll be posting your sites on a remote web server. (See Book VIII for details on how to set up a server.) Komodo makes it easy to edit a web page even when it's not on your own machine!
- ◆ **Page templates:** If you don't remember exactly how to start a page, you can choose New ⇨ File from Template from the File menu to start a file with some starter code in it. Note that the HTML5 code provided with Komodo does not include everything the validator wants, but you can add the features you want and save it as your own template (File ⇨ Save As ⇨ Template).
- ◆ **Code sample library:** Komodo comes with a complete code sample library. To see it, pick View ⇨ Tabs and Sidebars⇨Toolbox. The toolbox appears and contains a number of interesting tools. Choose `samples-HTML` from the tree structure and you'll see several useful HTML snippets. You can double-click on any of these to add a code snippet directly to your page. This can be helpful when you don't remember exactly how to type something.
- ◆ **Powerful macro system:** As you spend more time with your editor, you'll probably want to add some custom features. The Macro and command feature is especially powerful. This system allows you to record a series of keystrokes and play them back. This is handy when you find yourself doing something repetitive (for example, if you have a list of filenames and you want to turn them into links). I love a good macro system. If you create a particularly good macro, you can save it for later reuse and even attach a keystroke to it so it becomes a permanent part of your Komodo system.
- ◆ **Tools and commands:** Explore the Tools panel to see some very useful tools that are installed by default. These tools are often used to send commands to the underlying operating system. You can use the tool system to view the contents of a particular directory, preview the current document in a specific browser, or pretty much anything you can do from the command line.

Super-charging Komodo with Emmet

As you begin coding, the basic features of Komodo Edit are more than enough for your needs. However, you'll soon become more adept at coding, you may want some tools to improve your efficiency. My favorite add-on for Komodo is a tool called Emmet (formerly known as Zen Coding). It's a neat tool for writing HTML and CSS super-quickly.

Essentially, this tool allows you to enter a code snippet and Emmet expands it to complete code. For example, take a look at the following code:

```
html:5>h1{my page}+ul>li*5>{item $}
```

With Emmet installed, you can simply invoke Emmet's `expand abbreviation` command, and the following HTML snippet is created:

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Document</title>
</head>
<body>
```

```
<h1>my page</h1>
<ul>
  <li>item 1</li>
  <li>item 2</li>
  <li>item 3</li>
  <li>item 4</li>
  <li>item 5</li>
</ul>
</body>
</html>
```

Of course, you might not understand the Emmet code or the HTML it generates yet, so don't worry about installing Emmet until you're a little more fluent with HTML and CSS. However, when you're ready, you'll find that Emmet is one of the most powerful tools in your library. You can install Emmet (and many other interesting add-ons) by searching for it in the Tools ⇄ Add-ons menu.

I actually use Emmet more often than the code snippets built into Komodo because I find it faster and more flexible. With this tool and a little practice, I can build a web page far more quickly and accurately in a text editor than I ever would with a graphical tool like Dreamweaver.

- ◆ **Extensions and add-ons:** Komodo uses the same general architecture as the Firefox web browser. The developers of Komodo made it very easy to extend, so there are hundreds of really great add-ons you can install quite easily. After you have a feel for the stock version of Komodo, you may want to investigate some add-ons to make it even better. See the nearby sidebar "Super-charging Komodo with Emmet" to find out about my favorite add-on.

Other text editors

Many other text editors are used in web development. The most important thing is to find one that matches the way you work. If you don't like any of the editors I've suggested so far, here are a few more you might want to try:

- ◆ **SynEdit:** Much like Notepad++ and very popular with web developers
- ◆ **Scintilla:** Primarily a programming editor, but has nice support for HTML coding
- ◆ **jEdit:** A popular text editor written in Java with nice features, but some developers consider it slower than the other choices

The bottom line on editors

There is a dizzying array of editors for you to choose from. Which is the best for you is something of a personal decision. As your coding style develops, you'll know more about which is the best editor for you. If you're not sure, I recommend starting with gedit (if you want simple and fast) or Komodo Edit (if you're ready for a bit more power). Then as you spend more time with an editor, try some of the others out to see what best fits your needs.

Finding a Good Web Developer's Browser

Web pages are meant to display in a browser; so, of course, you need browsers for testing. Not all browsers are the same, though, so you need more than one. There are a number of important browsers in use right now, and you need to understand how they are related because they are how the user will see your work.

A little ancient history

You've probably already noticed that browsers are inconsistent in the way they display and handle web pages. It's useful to understand how we got into this mess.

Mosaic/Netscape: The killer application

In the beginning, browsers were written by small teams. The most important early browser was Mosaic, written by a team based at the National Center for Supercomputing Applications (NCSA) in Champaign-Urbana, Illinois.

Several members of that NCSA team decided to create a completely commercial web browser. Netscape was born and it quickly became the most prominent and important browser, with 97 percent market share at the peak of its popularity.

Microsoft enters (and wins) the battle

Microsoft came onto the scene with Internet Explorer (IE). A bitter fight (sometimes called the First Browser Wars) ensued between Microsoft and Netscape. Each browser added new features regularly. Eventually, entire sets of tags evolved, so a web page written for IE would not always work in Netscape and vice versa. Developers had three bad choices: pick only one

browser to support, write two versions of the page, or stick with the more limited set of features common to both browsers.

Netscape 6.0 was a technical disappointment, and Microsoft capitalized, earning a nearly complete lock on the browser market. Microsoft's version of standards became the *only* standards because there was virtually no competition. After Microsoft won the fight, there was a period of stability but very little innovation.

Firefox shakes up the world

A new browser rose from the ashes of Netscape (in fact, its original name was Firebird, after the mythical birds that rise from their own ashes). The name was later changed to Firefox, and it breathed new life into the web. Firefox has several new features that are very appealing to web developers:

- ◆ **Solid compliance to standards:** Firefox followed the W3C standards almost perfectly.
- ◆ **Tabbed browsing:** One browser window can have several panels, each with its own page.
- ◆ **Easy customization:** Firefox developers encouraged people to add improvements and extensions to Firefox. This led to hundreds of interesting add-ons.
- ◆ **Improved security:** By this time, a number of security loopholes in IE were publicized. Although Firefox has many of the same problems, it has a much better reputation for openness and quick solutions.

WebKit messes things up again

The next shakeup happened with a rendering engine called *WebKit*. This tool is the underlying engine shared by Apple's Safari and Google's Chrome browser. These browsers changed things again by being even more aggressive about standards-compliance and by emphasizing the programming capabilities built into a browser. Chrome and Safari are each extensions of the same essential technology. It gets messier. Recently Google announced that they are developing a new rendering engine called 'blink' based on WebKit. It's still not clear what this will mean, but for the time being, WebKit is a solid place to start.

HTML5 ushers in the second browser war

It is now becoming clear that the web is far more than a document mechanism. It is really becoming more like an operating system in its own right, and increasingly the web is about applications more than documents. HTML5 is at the center of this innovation, and today there are again many browser choices. It's a better situation, as developers are insisting on compliance with HTML5 standards, and any browser that follows these

standards will be acceptable. The real question today isn't which browser the user prefers, but does the user have a browser that's reasonably complaint with today's standards?

Overview of the prominent browsers

The browser is the primary tool of the web. All your users view your page with one browser or another, so you need to know a little about each of them.

Microsoft Internet Explorer 10

Microsoft Internet Explorer (IE) remains a dominant player on the Internet. Explorer is still extremely prevalent because it comes installed with Microsoft Windows. Of course, it also works exclusively with Microsoft Windows. Mac and Linux aren't supported (users don't seem too upset about it, though).

Version 10 of IE finally has respectable (if not complete support) for the major parts of the HTML5 standard. If you write pages according to the version of HTML5 described in this book (using a reasonably universal subset of the HTML5 standard), you can expect your page to work well in IE10. Most features will also work in IE9, but not all.

Older versions of Internet Explorer

The earlier versions of IE are still extremely important because so many computers out there don't have 10 installed yet. Version 6 was the dominant player in the Internet for some time, and it refuses to die. However, it will not play well with modern standards, so it's considered obsolete by most developers. (There are some software packages built on the proprietary features of IE6, so it refuses to die away completely, but there is no need for consumers to use this version.)

Mozilla Firefox

Firefox is a major improvement on IE from a programmer's point of view, for the following reasons:

- ◆ **Better code view:** If you view the HTML code of a page, you see the code in a special window. The code has syntax coloring, which makes it easy to read. Some versions of IE display code in Notepad, which is confusing because you think you can edit the code, but you're simply editing a copy.
- ◆ **Better error-handling:** You'll make mistakes. Generally, Firefox does a better job of pointing out errors than IE, especially when you begin using JavaScript and other advanced technologies.
- ◆ **Great extensions:** Firefox has some wonderful extensions that make web development a lot easier. These extensions allow you to modify your code on the fly, automatically validate your code, and explore the structure of your page dynamically.

- ◆ **Multi-platform support:** IE works only on the Windows operating system, so it isn't available to Mac or Linux users. Even if you're a Windows-only developer, your users may use something else, so you need to know how the other browsers see things.

WebKit/Safari

The default browser for Mac and the iPhone/iPad Operating System (iOS) is called Safari. It's a very powerful browser built on the WebKit rendering engine. Safari was designed with standards-compliance and speed in mind, and it shows. Your Mac and iOS users will almost certainly be using Safari, so you should know something about it. Fortunately, Chrome uses WebKit (or a variant) as well, so if things look good on Chrome, you're likely to be fine with your Apple users.

Google Chrome

Google sees the future of computing in browser-based applications using AJAX technologies. (AJAX is described in Book VII.) The Chrome browser is extremely fast, especially in the JavaScript technology that serves as the foundation to this strategy. Chrome complies quite well with common standards. In addition, Chrome has a number of developer toolkits that makes it the hands-down favorite browser for many web developers (including me). Many of the features of the developer tools make sense only when you have a bit more experience, but here are the highlights:

- ◆ **Real-time page editing:** You can go to any web page, right click 'inspect this element' and modify the text of that element in real time. You can then see what the element looks like with new content. You can select a part of the page to see which page corresponds to the code, and you can select the code and see which part of the page that code represents. Figure 3-6 illustrates this feature in action.

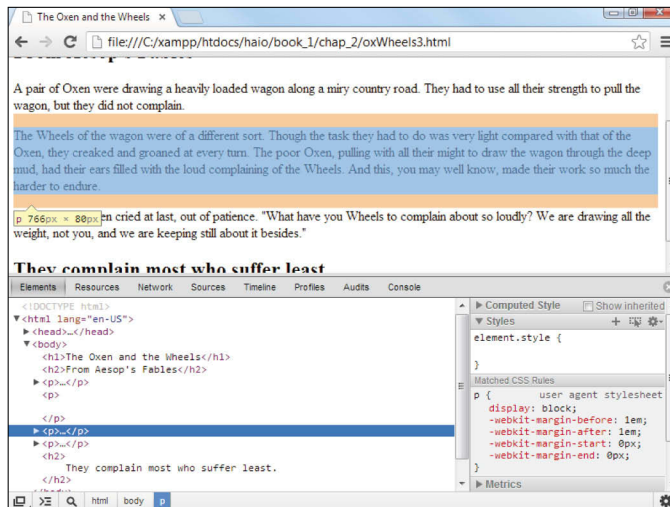


Figure 3-6:
The ability to
inspect an
element is
a powerful
feature of
Chrome.

- ◆ **Page Outline:** A well-designed web page is created in outline form, with various elements nested inside each other. The elements view allows you to see the web page in this format, with the ability to collapse and expand elements to see your page's structure clearly.
- ◆ **Realtime CSS Edit:** As you discover how to apply CSS styles in Books II and III, you'll want to be able to see how various CSS rules change your page. In the Inspect Element view, you can highlight a part of your page and change the CSS while seeing how the change affects your page in real time.
- ◆ **Network Tab:** This feature allows you to examine how long each piece of your page takes to load. It can be helpful for troubleshooting a slow-loading page.
- ◆ **Sources View:** This allows you to see the complete code of your page. It's especially useful when you get to JavaScript programming (in Book IV) because it includes a powerful debugging suite.
- ◆ **Console:** The console view is a little command-line tool integrated directly into your browser. This can be very helpful because it often shows errors that are otherwise hidden from view. The console is most useful when using JavaScript, so it is described in more detail in Book IV.

Other notable browsers

Firefox and IE are the big players in the browser world, but they certainly aren't the only browsers you will encounter.

Opera

The Opera web browser, one of the earliest standards-compliant browsers, is a technically solid browser that has never been widely used. If you design your pages with strict compliance in mind, users with Opera have no problems accessing them. Opera has very good HTML5 compliance. Many gaming consoles and mobile devices have browsers based on Opera, so it's worth looking into.

WebKit/Safari

Apple includes a web browser in all recent versions of Mac OS. The current incarnation — Safari — is an excellent standards-compliant browser. Safari was originally designed only for the Mac, but a Windows version is also available. The WebKit framework, the foundation for Safari, is used in a number of other online applications, mainly on the Mac. A modified version of Safari is the foundation of the browsers on the iPhone and iPad.

Text-only browsers

Some browsers that don't display any graphics at all (such as Lynx) are intended for the old command-line interfaces. This may seem completely irrelevant today, but these browsers are incredibly fast because they don't

display graphics. Auditory browsers read the contents of web pages. They were originally intended for people with visual disabilities, but people without any disabilities often use them as well. Fire Vox is a variant of Firefox that reads web pages aloud.



Worrying about text-only readers may seem unnecessary because people with visual disabilities are a relatively small part of the population, and you may not think they're part of your target audience. You probably should think about these users anyway because it isn't difficult to help them (and if you're developing for certain organizations, support for folks with disabilities is required). There's another reason, too. The search engines (Google is the main game in town) read your page just like a text-only browser. Therefore, if an element is invisible to a text-based browser, it won't appear on the search engine.

The bottom line in browsers

Really, you need to have access to a couple browsers, but you can't possibly have them all. I tend to do my initial development testing with Chrome. I look over my page in IE version 10 and I try to keep an older computer around with IE7 or 8 just to see what will happen.

I also check the built-in browser on an Android phone and iOS tablet to see how the pages look there. Generally, if you follow the subset of HTML5 outlined in this book, you can be satisfied that it works on most browsers. However, there's still no guarantee. If you follow the standards, your page displays on any browser, but you might not get the exact layout you expect.

Chapter 4: Managing Information with Lists and Tables

In This Chapter

- ✓ Understanding basic lists
- ✓ Creating unordered, ordered, and nested lists
- ✓ Building definition lists
- ✓ Building basic tables
- ✓ Using rowspan and colspan attributes

You'll often need to present large amounts of organized information, and HTML has some wonderful tools to manage this task. HTML has three kinds of lists and a powerful table structure for organizing the content of your page. Figure out how these tools work, and you can manage complex information with ease.

Making a List and Checking It Twice

HTML supports three types of lists. *Unordered* lists generally contain bullet points. They're used when the order of elements in the list isn't important. *Ordered* lists usually have some kind of numeric counter preceding each list item. *Definition* lists contain terms and their definitions.

Creating an unordered list

All the list types in HTML are closely related. The simplest and most common kind of list is an unordered list.

Looking at an unordered list

Look at the simple page shown in Figure 4-1. In addition to a couple of headers, it has a list of information.

This list of browsers has some interesting visual characteristics:

- ◆ **The items are indented.** There's some extra space between the left margin and the beginning of each list item.
- ◆ **The list elements have bullets.** That little dot in front of each item is a *bullet*. Bullets are commonly used in unordered lists like this one.

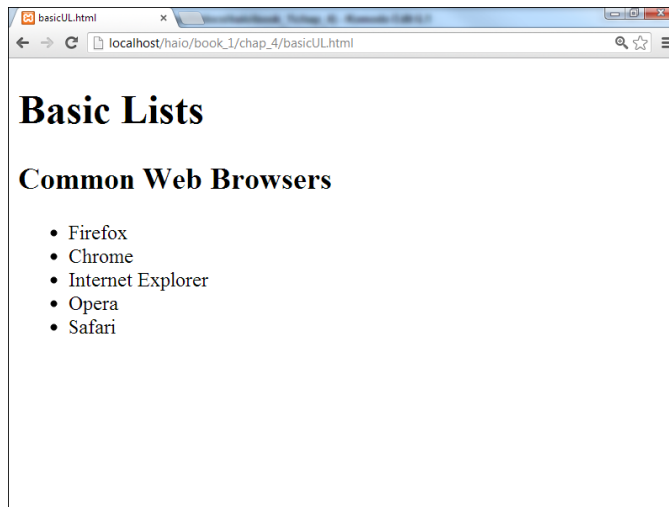


Figure 4-1:
An
unordered
list of web
browsers.

◆ **Each item begins a new line.** When a list item is displayed, it's shown on a new line.

These characteristics help you see that you have a list, but they're just default behaviors. Defining something as a list doesn't force it to look a particular way; the defaults just help you see that these items are indeed part of a list.



Remember the core idea of HTML here. You aren't really describing how things *look*, but what they *mean*. You can change the appearance later when you figure out CSS, so don't get too tied up in the particular appearance of things. For now, just recognize that HTML can build lists, and make sure you know how to use the various types.

Building an unordered list

Lists are made with two kinds of tags. One tag surrounds the entire list and indicates the general type of list. This first example demonstrates an unordered list, which is surrounded by the `` pair.

Note: Indenting all the code inside the `` set is common. The unordered list can go in the main body.

Inside the `` set is a number of list items. Each element of the list is stored between a `` (list item) and a `` tag. Normally, each `` pair goes on its own line of the source code, although you can make a list item as long as you want.



Look to Book II, Chapter 4 for information on how to change the bullet to all kinds of other images, including circles, squares, and even custom images.

The code for the unordered list is pretty straightforward:

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="UTF-8">
<title>basicUL.html</title>
</head>
<body>
  <h1>Basic Lists</h1>
  <h2>Common Web Browsers</h2>
  <ul>
    <li>Firefox</li>
    <li>Chrome</li>
    <li>Internet Explorer</li>
    <li>Opera</li>
    <li>Safari</li>
  </ul>
</body>
</html>
```

Creating ordered lists

Ordered lists are almost exactly like unordered lists. Ordered lists traditionally have numbers rather than bullets (although you can change this through CSS if you want; see Book II, Chapter 4).

Viewing an ordered list

Figure 4-2 demonstrates a page with a basic ordered list — basicOL.html.

Figure 4-2 shows a list where the items are numbered. When your data is a list of steps or information with some type of numerical values, an ordered list is a good choice.

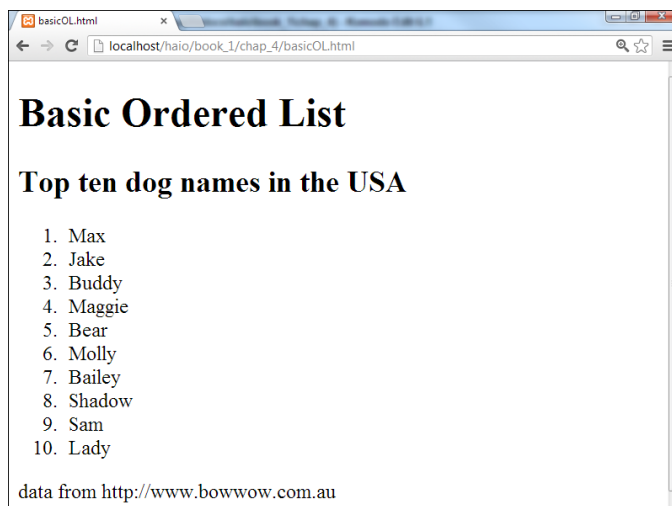


Figure 4-2:
A simple
ordered list.

Building the ordered list

The code for basicOL.html is remarkably similar to the previous unordered list:

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>basicOL.html</title>
</head>
<body>
  <h1>Basic Ordered List</h1>
  <h2>Top ten dog names in the USA</h2>
  <ol>
    <li>Max</li>
    <li>Jake</li>
    <li>Buddy</li>
    <li>Maggie</li>
    <li>Bear</li>
    <li>Molly</li>
    <li>Bailey</li>
    <li>Shadow</li>
    <li>Sam</li>
    <li>Lady</li>
  </ol>

  <p>
    data from http://www.bowwow.com.au
  </p>
</body>
</html>
```

The only change is the list tag itself. Rather than the `` tag, the ordered list uses the `` indicator. The list items are the same `` pairs used in the unordered list.

You don't indicate the item number anywhere; it generates automatically based on the position of each item within the list. Therefore, you can change the order of the items, and the numbers are still correct.



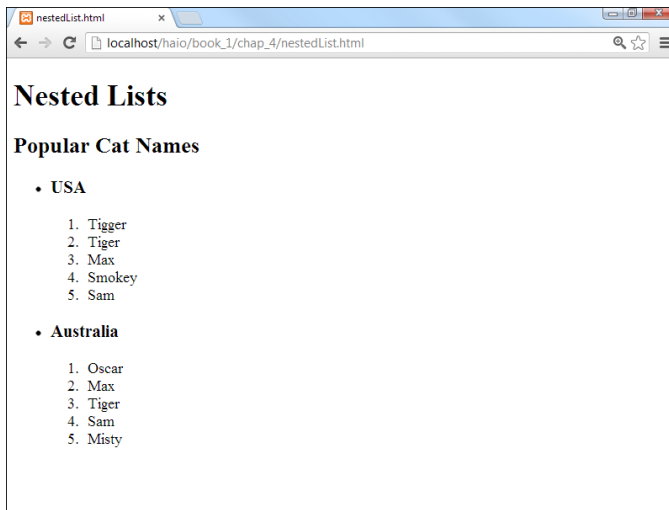
This is where it's great that HTML is about meaning, not layout. If you specified the actual numbers, it'd be a mess to move things around. All that really matters is that the element is inside an ordered list.

Making nested lists

Sometimes, you'll want to create outlines or other kinds of complex data in your pages. You can easily nest lists inside each other, if you want. Figure 4-3 shows a more complex list describing popular cat names in the U.S. and Australia.

Figure 4-3 uses a combination of lists to do its work. This figure contains a list of two countries: the U.S. and Australia. Each country has an H3 heading and another (ordered) list inside it. You can nest various elements inside a list, but you have to do it carefully if you want the page to validate.

Figure 4-3:
An ordered
list inside an
unordered
list!



In this example, there's an unordered list with only two elements. Each of these elements contains an `<h3>` heading and an ordered list. The page handles all this data in a relatively clean way and validates correctly.

Examining the nested list example

The entire code for `nestedList.html` is reproduced here:

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>nestedList.html</title>
</head>
<body>
  <h1>Nested Lists</h1>

  <h2>Popular Cat Names</h2>
  <ul>
    <li>
      <h3>USA</h3>
      <ol>
        <li>Tigger</li>
        <li>Tiger</li>
        <li>Max</li>
        <li>Smokey</li>
        <li>Sam</li>
      </ol>
    </li>

    <li>
      <h3>Australia</h3>
      <ol>
        <li>Oscar</li>
        <li>Max</li>
        <li>Tiger</li>
```

```

        <li>Sam</li>
        <li>Misty</li>
      </ol>
    </li>
  </ul>
</body>
</html>

```

Here are a few things you might notice in this code listing:

- ◆ There's a large `` set surrounding the entire main list.
- ◆ The main list has only two list items.
- ◆ Each of these items represents a country.
- ◆ Each country has an `<h3>` element, describing the country name inside the ``.
- ◆ Each country also has an `` set with a list of names.
- ◆ The indentation really helps you see how things are connected.

Indenting your code

You might have noticed that I indent all the HTML code in this book. The browsers ignore all indentation, but it's still an important coding habit.

There are many opinions about how code should be formatted, but the standard format I use in this book will serve you well until you develop your own style.

Generally, I use the following rules to indent HTML code:

- ◆ **Indent each nested element.** Because the `<head>` tag is inside the `<html>` element, I indent to indicate this. Likewise, the `` elements are always indented inside `` or `` pairs.
- ◆ **Line up your elements.** If an element takes up more than one line, line up the ending tag with the beginning tag. This way, you know what ends what.
- ◆ **Use spaces, not tabs.** The tab character often causes problems in source code. Different editors format tabs differently, and a mixture of tabs and spaces can make your carefully formatted page look awful when you view it in another editor.

Most editors have the ability to interpret the tab key as spaces. It's a great idea to find this feature on your editor and turn it on, so any time you hit the tab key, it's interpreted as spaces. In Komodo Edit, you do this in Edit ⇨ Preferences ⇨ Editor ⇨ Indentation.

- ◆ **Use two spaces.** Most coders use two or four spaces per indentation level. HTML elements can be nested pretty deeply. Going seven or eight layers deep is common. If you use tabs or too many spaces, you'll have so much white space that you can't see the code.

- ◆ **End at the left margin.** If you finish the page and you're not back at the left margin, you've forgotten to end something. Proper indentation makes seeing your page organization easy. Each element should line up with its closing tag.

Building a nested list

When you look over the code for the nested list, it can look intimidating, but it isn't really that hard. The secret is to build the list *outside in*:

1. **Create the outer list first.** Build the primary list (whether it's ordered or unordered). In my example, I began with just the unordered list with the two countries in it.
2. **Add list items to the outer list.** If you want text or headlines in the larger list (as I did), you can put them here. If you're putting nothing but a list inside your primary list, you may want to put some placeholder `` tags in there just so you can be sure everything's working.
3. **Validate before adding the next list level.** Nested lists can confuse the validator (and you). Validate your code with the outer list to make sure there are no problems before you add inner lists.
4. **Add the first inner list.** After you know the basic structure is okay, add the first interior list. For my example, this was the ordered list of cat names in the U.S.
5. **Repeat until finished.** Keep adding lists until your page looks right.
6. **Validate frequently.** It's much better to validate as you go than to wait until everything's finished. Catch your mistakes early so you don't replicate them.

Building the definition list

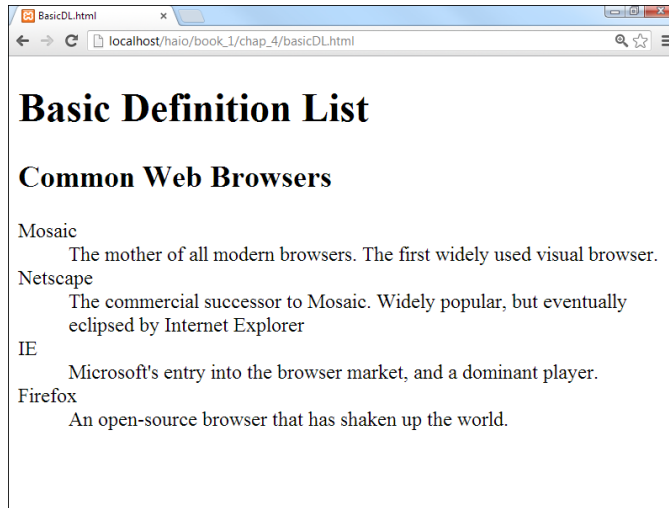
One more type of list — the definition list — is very useful, even if it's used infrequently. The definition list was originally designed to format dictionary-style definitions, but it's really useful any time you have name and value pairs. Figure 4-4 shows a sample definition list in action.

Definition lists don't use bullets or numbers. Instead, they have two elements. *Definition terms* are usually words or short phrases. In Figure 4-4, the browser names are defined as definition terms. *Definition descriptions* are the extended text blocks that contain the actual definition.

The standard layout of definition lists indents each definition description. Of course, you can change the layout to what you want after you understand the CSS in Books II and III.

You can use definition lists any time you want a list marked by key terms, rather than bullets or numbers. The definition list can also be useful in other situations, such as forms, figures with captions, and so on.

Figure 4-4:
A basic
definition
list.



Here's the code for basicDL.html:

```
<!DOCTYPE HTML>
<html lang="en-US">
  <head>
    <meta charset="UTF-8">
    <title>BasicDL.html</title>
  </head>
  <body>
    <h1>Basic Definition List</h1>
    <h2>Common Web Browsers</h2>
    <dl>
      <dt>Mosaic</dt>
      <dd>
        The mother of all modern browsers. The first widely used
        visual browser.
      </dd>

      <dt>Netscape</dt>
      <dd>
        The commercial successor to Mosaic. Widely popular, but
        eventually eclipsed by Internet Explorer
      </dd>

      <dt>IE</dt>
      <dd>
        Microsoft's entry into the browser market, and a dominant
        player.
      </dd>

      <dt>Firefox</dt>
      <dd>
        An open-source browser that has shaken up the world.
      </dd>
    </dl>
  </body>
</html>
```

As you can see, the definition list uses three tag pairs:

- ◆ `<dl></dl>` defines the entire list.
- ◆ `<dt></dt>` defines each definition term.
- ◆ `<dd></dd>` defines the definition data.

Definition lists aren't used often, but they can be extremely useful. Any time you have a list that will be a combination of terms and values, a definition list is a good choice.

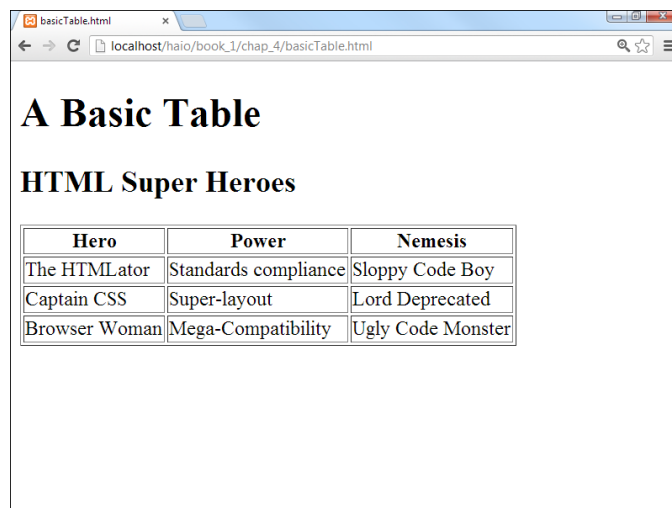
Building Tables

Sometimes, you'll encounter data that fits best in a tabular format. HTML supports several table tags for this kind of work. Figure 4-5 illustrates a very basic table.

Sometimes, the best way to show data in a meaningful way is to organize it in a table. HTML defines a table with the (cleverly named) `<table>` tag. The table contains a number of table rows (defined with the `<tr>` tag). Each table row can consist of a number of table data (`<td>`) or table header (`<th>`) tags.

Compare the output in Figure 4-5 with the code for `basicTable.html` that creates it:

```
<!doctype html>
<html lang="en">
```



Hero	Power	Nemesis
The HTMLator	Standards compliance	Sloppy Code Boy
Captain CSS	Super-layout	Lord Deprecated
Browser Woman	Mega-Compatibility	Ugly Code Monster

Figure 4-5: Tables are useful for certain kinds of data representation.

```

<head>
  <meta charset="UTF-8">
  <title>basicTable.html</title>
</head>
<body>
  <h1>A Basic Table</h1>
  <h2>HTML Super Heroes</h2>
  <table border = "1">
    <tr>
      <th>Hero</th>
      <th>Power</th>
      <th>Nemesis</th>
    </tr>

    <tr>
      <td>The HTMLator</td>
      <td>Standards compliance</td>
      <td>Sloppy Code Boy</td>
    </tr>

    <tr>
      <td>Captain CSS</td>
      <td>Super-layout</td>
      <td>Lord Deprecated</td>
    </tr>

    <tr>
      <td>Browser Woman</td>
      <td>Mega-Compatibility</td>
      <td>Ugly Code Monster</td>
    </tr>

  </table>
</body>
</html>

```

Defining the table

The HTML table is defined with the `<table></table>` pair. It makes a lot of sense to indent and space your code carefully so you can see the structure of the table in the code. Just by glancing at the code, you can guess that the table consists of three rows and each row consists of three elements.

In a word processor, you typically create a blank table by defining the number of rows and columns, and then fill it in. In HTML, you define the table row by row, and the elements in each row determine the number of columns. It's up to you to make sure each row has the same number of elements.

By default (in most browsers, anyway), tables don't show their borders. If you want to see basic table borders, you can turn on the table's `border` attribute. (An *attribute* is a special modifier you can attach to some tags.)

```
<table border = "1">
```

This tag creates a table and specifies that it will have a border of size 1. If you leave out the `border = "1"` business, some browsers display a border



and some don't. You can set the border value to 0 or to a larger number. The larger number makes a bigger border, as shown in Figure 4-6.



Although this method of making table borders is perfectly fine, I show a much more flexible and powerful technique in Book II, Chapter 4.

Setting a table border is a good idea because you can't count on browsers to have the same default. Additionally, the border value is always in quotes. When you read about CSS in Book II (are you getting tired of hearing that yet?), you discover how to add more complex and interesting borders than this simple attribute allows.

Adding your first row

After you define a table, you need to add some rows. Each row is indicated by a `<tr></tr>` pair.

Inside the `<tr></tr>` set, you need some table data. The first row often consists of *table headers*. These special cells are formatted differently to indicate that they're labels, rather than data.



Table headers have some default formatting to help you remember they're headers, but you can change the way they look. You can change the table header's appearance in all kinds of great ways in Books II and III. Define the table header so when you discover formatting and decide to make all your table headers chartreuse, you'll know where in the HTML code all the table headers are.

Indent your headers inside the `<tr>` set. If your table contains three columns, your first row might begin like this:

Hero	Power	Nemesis
The HTMLator	Standards compliance	Sloppy Code Boy
Captain CSS	Super-layout	Lord Deprecated
Browser Woman	Mega-Compatibility	Ugly Code Monster

Figure 4-6:
I set the
border
attribute
to 10.

```
<tr>
  <th></th>
  <th></th>
  <th></th>
</tr>
```

Place the text you want shown in the table headers between the `<th>` and `</th>` elements. The contents appear in the order they're defined.



Headings don't have to be on the top row. If you want headings on the left, just put a `<th></th>` pair as the first element of each row. You can have headings at both the top and the left, if you want. In fact, you can have headings anywhere, but it usually makes sense to put headings only at the top or left.

Making your data rows

The next step is to create another row. The data rows are just like the heading row, except they use `<td></td>` pairs, rather than `<th></th>` pairs, to contain the data elements. Typically, a three-column table has blank rows that look like this:

```
<tr>
  <td></td>
  <td></td>
  <td></td>
</tr>
```

Place the data elements inside the `<td></td>` segments and you're ready to go.

Building tables in the text editor

Some people think that tables are a good reason to use WYSIWYG (what you see is what you get) editors because they think it's hard to create tables in text mode. You have to plan a little, but it's really quite quick and easy to build an HTML table without graphical tools if you follow this plan:

1. **Plan ahead.** Know how many rows and columns will be in the table. Sketching it on paper first might be helpful. Changing the number of rows later is easy, but changing the number of columns can be a real pain after some of the code has been written.
2. **Create the headings.** If you're going to start with a standard headings-on-top table, begin by creating the heading row. Save, check, and validate. You don't want mistakes to multiply when you add more complexity. This heading row tells how many columns you'll need.
3. **Build a sample empty row.** Make a sample row with the correct number of `td` elements with one `<td></td>` pair per line. Build one `td` set and use copy and paste to copy this data cell as many times as you need. Make sure the number of `<td>` pairs equals the number of `<th>` sets in the heading row.

4. **Copy and paste the empty row to make as many rows as you need.**
5. **Save, view, and validate.** Be sure everything looks right and validates properly before you put a lot of effort into adding data.
6. **Populate the table with the data you need.** Go row by row, adding the data between the `<td></td>` pairs.
7. **Test and validate again to make sure you didn't accidentally break something.**

Spanning rows and columns

Sometimes, you need a little more flexibility in your table design. Figure 4-7 shows a page from an evil overlord's daily planner.

Being an evil overlord is clearly a complex business. From a code standpoint, the items that take up more than one cell are the most interesting. Designing traps takes two mornings, and improving the hideout takes three. All Friday afternoon and evening are spent on world domination. Take a look at the code, and you'll see how it works:

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>tableSpan.html</title>
</head>
<body>
  <h1>Using colspan and rowspan</h1>
  <table border = "1">
    <caption><p>My Schedule</p></caption>
    <tr>
      <th></th>
      <th>Monday</th>
      <th>Tuesday</th>
      <th>Wednesday</th>
      <th>Thursday</th>
      <th>Friday</th>
    </tr>

    <tr>
      <th>Breakfast</th>
      <td>In lair</td>
      <td>with cronies</td>
      <td>In lair</td>
      <td>in lair</td>
      <td>in lair</td>
    </tr>

    <tr>
      <th>Morning</th>
      <td colspan = "2">Design traps</td>
      <td colspan = "3">Improve Hideout</td>
    </tr>

    <tr>
      <th>Afternoon</th>
      <td>train minions</td>
```

```

        <td>train minions</td>
        <td>train minions</td>
        <td>train minions</td>
        <td rowspan = "2">world domination</td>
    </tr>

    <tr>
        <th>Evening</th>
        <td>manaical laughter</td>
        <td>manaical laughter</td>
        <td>manaical laughter</td>
        <td>manaical laughter</td>
    </tr>

</table>
</body>
</html>

```

The secret to making cells larger than the default is two special attributes: `rowspan` and `colspan`.

Figure 4-7:
Some
of these
activities
take up
more than
one cell.

The screenshot shows a web browser window with the title 'tableSpan.html'. The address bar shows 'localhost/haio/book_1/chap_4/tableSpan.html'. The main content area has the heading 'Using colspan and rowspan' and a subheading 'My Schedule'. Below the subheading is a table with 6 columns: an empty header cell, 'Monday', 'Tuesday', 'Wednesday', 'Thursday', and 'Friday'. The table body contains the following rows:

	Monday	Tuesday	Wednesday	Thursday	Friday
Breakfast	In lair	with cronies	In lair	in lair	in lair
Morning	Design traps		Improve Hideout		
Afternoon	train minions	train minions	train minions	train minions	world domination
Evening	manaical laughter	manaical laughter	manaical laughter	manaical laughter	

Spanning multiple columns

The morning activities tend to happen over several days. Designing traps will take both Monday and Tuesday mornings, and improving the hideout will occupy the remaining three mornings. Take another look at the Morning row; here's how this is done:

```

<tr>
    <th>Morning</th>
    <td colspan = "2">Design traps</td>
    <td colspan = "3">Improve Hideout</td>
</tr>

```

The Design Traps cell spans over two normal columns. The `colspan` attribute tells how many columns this cell will take. The Improve Hideout cell has a `colspan` of 3.

The Morning row still takes up six columns. The `<th>` is one column wide, like normal, but the Design Traps cell spans two columns and the Improve Hideout cell takes three, which totals six columns wide. If you increase the width of a cell, you need to eliminate some other cells in the row to compensate.

Spanning multiple rows

A related property — `rowspan` — allows a cell to take up more than one row of a table. Look back at the Friday column in Figure 4-7, and you'll see the World Domination cell takes up two time slots. (If world domination was easy, everybody would do it.) Here's the relevant code:

```
<tr>
  <th>Afternoon</th>
  <td>train minions</td>
  <td>train minions</td>
  <td>train minions</td>
  <td>train minions</td>
  <td rowspan = "2">world domination</td>
</tr>

<tr>
  <th>Evening</th>
  <td>maniacal laughter</td>
  <td>maniacal laughter</td>
  <td>maniacal laughter</td>
  <td>maniacal laughter</td>
</tr>
```

The Evening row has only five entries because the World Domination cell extends into the space that would normally be occupied by a `<td>` pair.



If you want to use `rowspan` and `colspan`, don't just hammer away at the page in your editor. Sketch out what you want to accomplish first. I'm pretty good at this stuff, and I still needed a sketch before I was able to create the `tableSpan.html` code.

Avoiding the table-based layout trap

Tables are pretty great. They're a terrific way to present certain kinds of data. When you add the `colspan` and `rowspan` concepts, you can use tables to create some pretty interesting layouts. In fact, because old-school HTML didn't really have any sort of layout technology, a lot of developers came up with some pretty amazing layouts based on tables. You still see a lot of web pages today designed with tables as the primary layout mechanism.

Using tables for layout causes some problems though, such as

- ◆ **Tables aren't meant for layout.** Tables are designed for data presentation, not layout. To make tables work for layout, you have to do a lot of sneaky hacks, such as tables nested inside other tables or invisible images for spacing.
- ◆ **The code becomes complicated fast.** Tables involve a lot of HTML markup. If the code involves tables nested inside each other, it's very difficult to remember which `<td>` element is related to which row of which table. Table-based layouts are very difficult to modify by hand.
- ◆ **Formatting is done cell by cell.** A web page could be composed of hundreds of table cells. Making a change in the font or color often involves making changes in hundreds of cells throughout the page. This makes your page less flexible and harder to update.
- ◆ **Presentation is tied tightly to data.** A table-based layout tightly intertwines the data and its presentation. This runs counter to a primary goal of web design — separation of data from its presentation.
- ◆ **Table-based layouts are hard to change.** After you create a layout based on tables, it's very difficult to make modifications because all the table cells have a potential effect on other cells.
- ◆ **Table-based layouts cause problems for screen-readers.** People with visual disabilities use special software to read web pages. These screen-readers are well adapted to read tables as they were intended (to manage tabular data), but the screen-readers have no way of knowing when the table is being used as a layout technique rather than a data presentation tool. This makes table-based layouts less compliant to accessibility standards.
- ◆ **Table-based layouts do not adapt well.** Modern users expect to run pages on cell phones and tablets as well as desktop machines. Table-based designs do not easily scale to these smaller form-factors.

Resist the temptation to use tables for layout. Use tables to do what they're designed for: data presentation. Book III is entirely about how to use CSS to generate any kind of visual layout you might want. The CSS-based approaches are easier, more dependable, and much more flexible.

Chapter 5: Making Connections with Links

In This Chapter

- ✓ Understanding hyperlinks
- ✓ Building the anchor tag
- ✓ Recognizing absolute and relative links
- ✓ Building internal links
- ✓ Creating lists of links

The basic concept of the hyperlink is common today, but it was a major breakthrough back in the day. The idea is still pretty phenomenal, if you think about it: When you click a certain piece of text (or a designated image, for that matter), your browser is instantly transported somewhere else. The new destination might be on the same computer as the initial page, or it could be literally anywhere in the world.

Any page is theoretically a threshold to any other page, and all information has the ability to be linked. This is still a profound idea. In this chapter, you discover how to add links to your pages.

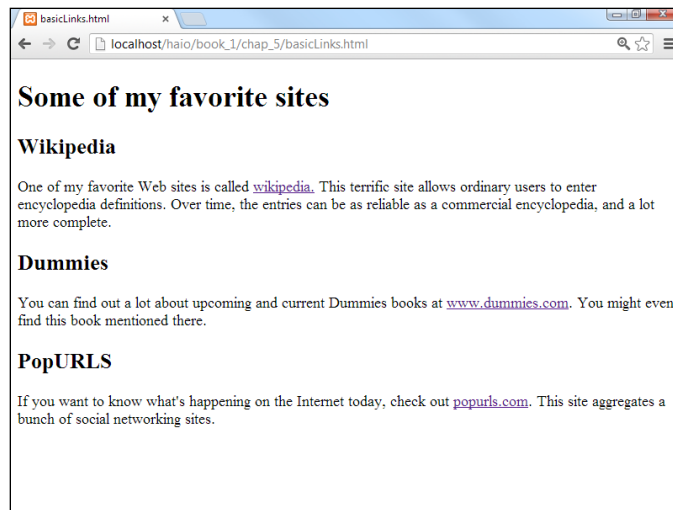
Making Your Text Hyper

The hyperlink is truly a wonderful thing. Believe it or not, there was a time when you had to manually type in the address of the web page you wanted to go to. Not so anymore. Figure 5-1 illustrates a page that describes some of my favorite websites.

In Figure 5-1, the underlined words are hyperlinks. Clicking a hyperlink takes you to the indicated website. Although this is undoubtedly familiar to you as a web user, a few details are necessary to make this mechanism work:

- ◆ **Something must be linkable.** Some text or other element must provide a trigger for the linking behavior.
- ◆ **Things that are links should look like links.** This is actually easy to do when you write plain HTML because all links have a standard (if ugly) appearance. Links are usually underlined blue text. When you can create color schemes, you may no longer want links to look like the default appearance, but they should still be recognizable as links.

Figure 5-1:
You can
click the
links to visit
the other
sites.



- ◆ **The browser needs to know where to go.** When the user clicks the link, the browser is sent to some address somewhere on the Internet. Sometimes that address is visible on the page, but it doesn't need to be.
- ◆ **It should be possible to integrate links into text.** In this example, each link is part of a sentence. It should be possible to make some things act like links without necessarily standing on their own (like heading tags do).
- ◆ **The link's appearance sometimes changes.** Links sometimes begin as blue underlined text, but after a link has been visited, the link is shown in purple, instead. After you know CSS, you can change this behavior.



Of course, if your web page mentions some other website, you should provide a link to that other website.

Introducing the anchor tag

The key to hypertext is an oddly named tag called the *anchor* tag. This tag is encased in an `<a>` set of tags and contains all the information needed to manage links between pages.

The code for the `basicLinks.html` page is shown here:

```
<!DOCTYPE html>
<html lang = "en-US">
  <head>
    <meta charset = "UTF-8">
    <title>basicLinks.html</title>
```

```

</head>

<body>
  <h1>Some of my favorite sites</h1>
  <h2>Wikipedia</h2>
  <p>
    One of my favorite websites is called
    <a href = "http://www.wikipedia.org">wikipedia.</a>
    This terrific site allows ordinary users to enter
    encyclopedia definitions. Over time, the entries
    can be as reliable as a commercial encyclopedia,
    and a lot more complete.
  </p>

  <h2>Dummies</h2>
  <p>
    You can find out a lot about upcoming and current
    Dummies books at <a href = "http://www.dummies.com">
    www.dummies.com</a>. You might even find this
    book mentioned there.
  </p>

  <h2>PopURLS</h2>
  <p>
    If you want
    to know what's happening on the Internet today,
    check out <a href = "http://popurls.com">
    popurls.com</a>. This site aggregates a bunch of
    social networking sites.
  </p>
</body>
</html>

```

As you can see, the anchor tag is embedded into paragraphs. The text generally flows around an anchor, and you can see the anchor code is embedded inside the paragraphs.

Comparing block-level and inline elements

All the tags described so far in this book have been *block-level* tags. Block-level tags typically begin and end with carriage returns. For example, three `<h1>` tags occupy three lines. Each `<p></p>` set has implied space above and below it. Most HTML tags are block-level.

Some tags are meant to be embedded inside block-level tags and don't interrupt the flow of the text. The anchor tag is one such tag. Anchors never stand on their own in the HTML body. This type of tag is an *inline* tag. They're meant to be embedded inside block-level tags, such as list items, paragraphs, and headings.

Analyzing an anchor

The first link shows all the main parts of an anchor in a pretty straightforward way:

```
<a href = "http://www.wikipedia.org">wikipedia.</a>
```

- ◆ **The anchor tag itself:** The anchor tag is simply the `<a>` pair. You don't type the entire word *anchor*, just the *a*.
- ◆ **The hypertext reference (`href`) attribute:** Almost all anchors contain this attribute. It's very rare to write `<a` without `href`. The `href` attribute indicates a web address will follow.
- ◆ **A web address in quotes:** The address that the browser will follow is encased in quotes. See the next section in this chapter for more information on web addresses. In this example, `http://www.wikipedia.org` is the address.
- ◆ **The text that appears as a link:** The user will typically expect to click specially formatted text. Any text that appears between the `<a href>` part and the `` part is visible on the page and formatted as a link. In this example, the word *wikipedia* is the linked text.
- ◆ **The `` marker:** This marker indicates that the text link is finished.

Introducing URLs

The special link addresses are a very important part of the web. You probably already type web addresses into the address bar of your browser (`http://www.google.com`), but you may not be completely aware of how they work. Web addresses are technically URLs (Uniform Resource Locators), and they have a very specific format.



Sometimes, you'll see the term *URI* (Uniform Resource Identifier) instead of URL. URI is technically a more correct name for web addresses, but the term URL has caught on. The two terms are close enough to be interchangeable.

A URL usually contains the following parts:

- ◆ **Protocol:** A web *protocol* is a standardized agreement on how communication occurs. The web primarily uses HTTP (hypertext transfer protocol), but occasionally, you encounter others. Most addresses begin with `http://` because this is the standard on the web. Protocols usually end with a colon and two slashes (`://`).
- ◆ **Host name:** It's traditional to name your primary web server **www**. There's no requirement for this, but it's common enough that users expect to type **www** right after the `http://` stuff. Regardless, the text right after `http://` (and up to the first period) is the name of the actual computer you're linking to.
- ◆ **Domain name:** The last two or three characters indicate a particular type of web server. These letters can indicate useful information about the type of organization that houses the page. Three-letter domains usually indicate the type of organization, and two-letter domains indicate a country. Sometimes, you'll even see a combination of the two.

- ◆ **Subdomain:** Everything between the host name (usually `www`) and the domain name (often `.com`) is the subdomain. This is used so that large organizations can have multiple servers on the same domain. For example, my department web page is `http://www.cs.iupui.edu`. `www` is the name of the primary server, and this is the computer science department at IUPUI (Indiana University–Purdue University Indianapolis), which is an educational organization.
- ◆ **Page name:** Sometimes, an address specifies a particular document on the web. This page name follows the address and usually ends with `.html`. Sometimes, the page name includes subdirectories and username information, as well. For example, my web development course is in the N241 directory of my (aharris) space at IUPUI, so the page's full address is `http://www.cs.iupui.edu/~aharris/n241/index.html`.
- ◆ **Username:** Some web servers are set up with multiple users. Sometimes, an address will indicate a specific user's account with a tilde (~) character. My address has `~aharris` in it to indicate the page is found in my (aharris) account on the machine.



The page name is sometimes optional. Many servers have a special name set up as the default page, which appears if no other name is specified. This name is usually `index.html` but sometimes `home.htm`. On my server, `index.html` is the default name, so I usually just point to `www.cs.iupui.edu/~aharris/n241`, and the index page appears.

<i>Domain</i>	<i>Explanation</i>
<code>.org</code>	Non-profit institution
<code>.com</code>	Commercial enterprise
<code>.edu</code>	Educational institution
<code>.gov</code>	Governing body
<code>.ca</code>	Canada
<code>.uk</code>	United Kingdom
<code>.tv</code>	Tuvalu

Making Lists of Links

Many web pages turn out to be lists of links. Because lists and links go so well together, it's good to look at an example. Figure 5-2 illustrates a list of links to books written by a certain (cough) devilishly handsome author.

This example has no new code to figure out, but the page shows some interesting components:

- ◆ **The list:** An ordinary unordered list.

- ◆ **Links:** Each list item contains a link. The link has a reference (which you can't see immediately) and linkable text (which is marked like an ordinary link).
- ◆ **Descriptive text:** After each link is some ordinary text that describes the link. Writing some text to accompany the actual link is very common.



Figure 5-2:
Putting links
in a list is
common.

This code shows the way the page is organized:

```
<!DOCTYPE html>
<html lang = "en-US">

  <head>
    <meta charset = "UTF-8">
    <title>listLinks.html</title>
  </head>

  <body>
    <h1>Some nice programming books</h1>
    <ul>
      <li><a href = "http://www.aharrisbooks.net/haio">
        HTML / CSS / JavaScript ALL in One for Dummies</a>
        A complete resource to web development</li>
      <li><a href = "http://www.aharrisbooks.net/jad">
        JavaScript / AJAX for Dummies</a>
        Using JavaScript, AJAX, and jQuery</li>
      <li><a href="http://www.aharrisbooks.net/pythonGame">
        Game Programming - the L Line</a>
        Game development in Python</li>
      <li><a href="http://www.aharrisbooks.net/h5g">
        HTML5 Game Development for Dummies</a>
        Building web and mobile games in HTML5</li>
    </ul>
  </body>
</html>
```

The indentation is interesting here. Each list item contains an anchor and some descriptive text. To keep the code organized, web developers tend to place the anchor inside the list item. The address sometimes goes on a new line if it's long, with the anchor text on a new line and the description on succeeding lines. I normally put the `` tag at the end of the last line, so the beginning `` tags look like the bullets of an unordered list. This makes it easier to find your place when editing a list later.

Working with Absolute and Relative References

There's more than one kind of address. So far, you've seen only absolute references, used for links to outside pages. Another kind of reference — a relative reference — links multiple pages inside your own website.

Understanding absolute references

The type of link used in `basicLinks.html` is an *absolute reference*. Absolute references always begin with the protocol name (usually `http://`). An absolute reference is the complete address to a web page, just as you'd use in the browser's address bar. Absolute references are used to refer to a site somewhere else on the Internet. Even if your website moves (say, from your desktop machine to a web server somewhere on the Internet), all the absolute references will work fine because they don't rely on the current page's position for any information.

Introducing relative references

Relative references are used when your website includes more than one page. You might choose to have several pages and a link mechanism for moving among them. Figure 5-3 shows a page with several links on it.

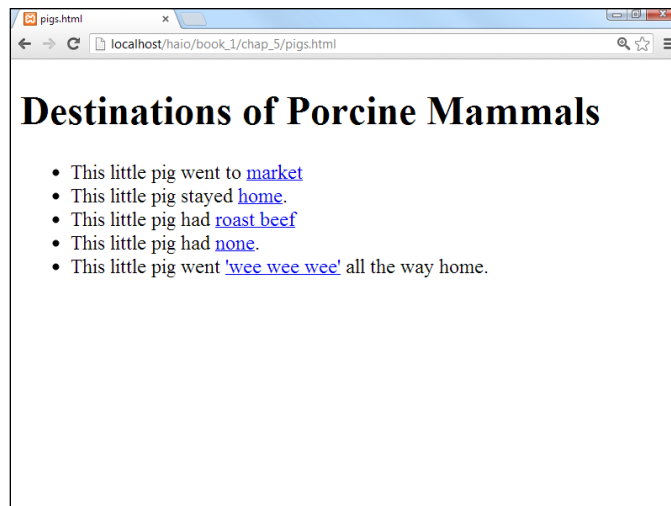


Figure 5-3:
These
little piggies
sure get
around...

The page isn't so interesting on its own, but it isn't meant to stand alone. When you click one of the links, you go to a brand-new page. Figure 5-4 shows what happens when you click the market link.

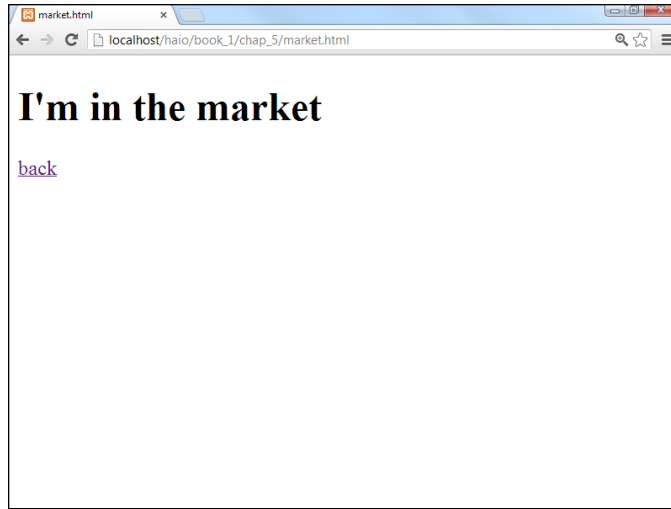


Figure 5-4:
The market
page lets
you move
back.

The market page is pretty simple, but it also contains a link back to the initial page. Most websites aren't single pages at all, but an interconnected web of pages. The relative reference is very useful when you have a set of pages with interlacing links.

The code for pigs.html shows how relative references work:

```
<!DOCTYPE html>
<html lang = "en-US">

  <head>
    <meta charset = "UTF-8">
    <title>pigs.html</title>
  </head>

  <body>
    <h1>Destinations of Porcine Mammals</h1>
    <ul>
      <li>This little pig went to
        <a href = "market.html">market</a></li>
      <li>This little pig stayed
        <a href = "home.html">home</a>.</li>
      <li>This little pig had
        <a href = "roastBeef.html">roast beef</a></li>
      <li>This little pig had
        <a href = "none.html">none</a>.</li>
      <li>This little pig went
        <a href = "wee.html">'wee wee wee'</a>
        all the way home.</li>
    </ul>
  </body>
</html>
```

Most of the code is completely familiar. The only thing surprising is what's *not* there. Take a closer look at one of the links:

```
<a href = "market.html">home</a>.</li>
```

There's no protocol (the `http://` part) and no address at all, just a file-name. This is a *relative reference*. Relative references work by assuming the address of the current page. When the user clicks `market.html`, the browser sees no protocol, so it assumes that `market.html` is in the same directory on the same server as `pigs.html`.

Relative references work like directions. For example, if you're in my lab and ask where the water fountain is, I'd say, "Go out into the hallway, turn left, and turn left again at the end of the next hallway." Those directions get you to the water fountain if you start in the right place. If you're somewhere else and you follow the same directions, you don't really know where you'll end up.

Relative references work well when you have a bunch of interconnected web pages. If you create a lot of pages about the same topic and put them in the same directory, you can use relative references between the pages. If you decide to move your pages to another server, all the links still work correctly.



In Book VIII, you discover how to set up a permanent web server. It's often most convenient to create and modify your pages on the local machine and then ship them to the web server for the world to see. If you use relative references, it's easy to move a group of pages together and know the links will still work.

If you're referring to a page on somebody else's site, you have to use an absolute reference. If you're linking to another page on your site, you typically use a relative reference.

Chapter 6: Adding Images, Sound, and Video

In This Chapter

- ✓ Understanding the main uses of images
- ✓ Choosing an image format
- ✓ Creating inline images
- ✓ Using IrfanView and other image software
- ✓ Changing image sizes
- ✓ Adding audio clips
- ✓ Working with video

You have the basics of text, but pages with nothing but text are... well, a little boring. Pictures do a lot for a web page, and they're pretty easy to work with. Today's web is really a multimedia environment, and HTML5 finally offers great support for audio and video. Find out how to add all these great features to your web pages.

Adding Images to Your Pages

Every time you explore the web, you're bound to run into tons of pictures on just about every page you visit. Typically, images are used in four ways on web pages:

- ◆ **External link:** The page has text with a link embedded in it. When the user clicks the link, the image replaces the page in the web browser. To make an externally linked image, just make an ordinary link (as I describe in Chapter 5 of this minibook), but point toward an image file, rather than an HTML (HyperText Markup Language) file.
- ◆ **Embedded images:** The image is embedded into the page. The text of the page usually flows around the image. This is the most common type of image used on the web.
- ◆ **Background images:** An image can be used as a background for the entire page or for a specific part of the page. Images usually require some special manipulation to make them suitable for background use.
- ◆ **Custom bullets:** With CSS, you can assign a small image to be a bullet for an ordered or unordered list. This allows you to make any kind of customized list markers you can draw.

The techniques you read about in this chapter apply to all type of images, but a couple of specific applications (such as backgrounds and bullets) use CSS. For details on using images in CSS, see Book II, Chapter 4.

Linking to an image

The easiest way to incorporate images is to link to them. Figure 6-1 shows the `externalImage.html` page.

The page's code isn't much more than a simple link:

```
<!DOCTYPE html>
<html lang = "en-US">
  <head>
    <meta charset = "UTF-8">
    <title>externalImage.html</title>
  </head>
  <body>
    <h1>Linking to an External Image</h1>
    <p>
      <a href = "shipStandard.jpg">
        Susan B. Constant
      </a>
    </p>
  </body>
</html>
```

The `href` points to an image file, not an HTML page. You can point to any type of file you want in an anchor tag. If the browser knows the file type (for example, HTML and standard image formats), the browser displays the file. If the browser doesn't know the file format, the user's computer tries to display the file using whatever program it normally uses to open that type of file.

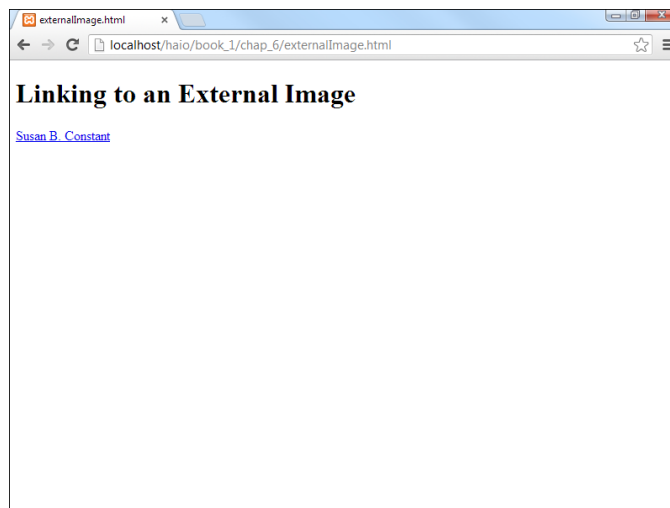


Figure 6-1:
This page
has a link to
an image.



See Chapter 5 of this minibook for a discussion of anchor tags if you need a refresher.

This works fine for most images because the image is displayed directly in the browser.



You can use this anchor trick with any kind of file, but the results can be very unpredictable. If you use the link trick to point to some odd file format, there's no guarantee the user has the appropriate software to view it. Generally, save this trick for very common formats, like GIF and JPG. (If these formats are unfamiliar to you, they are described later in this chapter.)

Most browsers automatically resize the image to fit the browser size. This means a large image may appear to be smaller than it really is, but the user still has to wait for the entire image to download.

Because this is a relative reference, the indicated image must be in the same directory as the HTML file. When the user clicks the link, the page is replaced by the image, as shown in Figure 6-2.

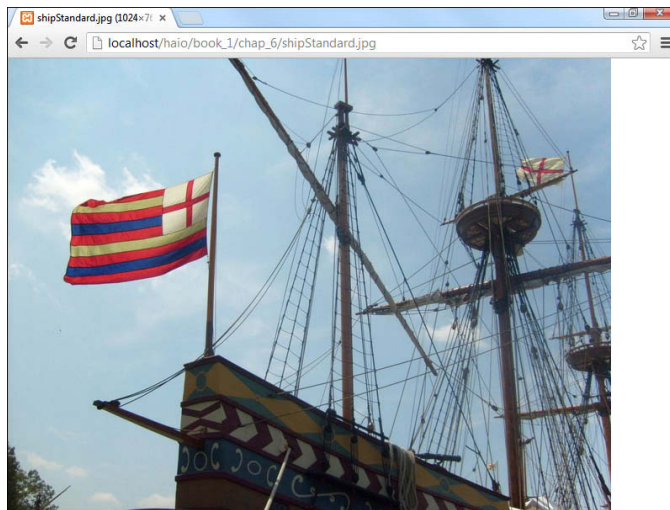


Figure 6-2:
The image
appears in
place of the
page.

External links are easy to create, but they have some problems:

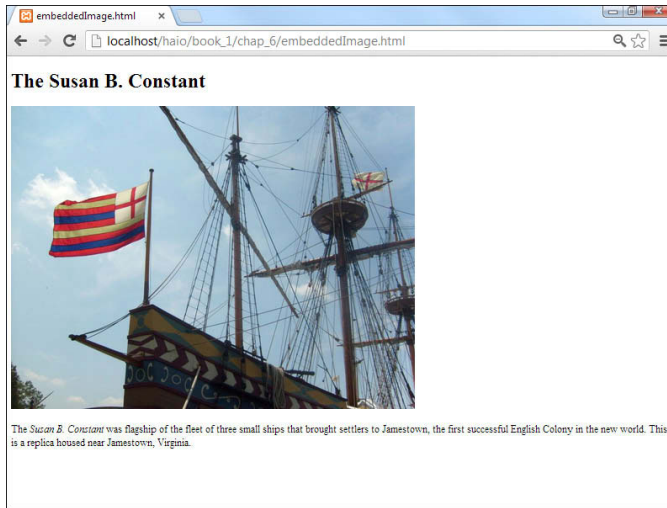
- ◆ **They don't preview the image.** The user has only the text description to figure out what the picture might be.
- ◆ **They interrupt the flow.** If the page contains a series of images, the user has to keep leaving the page to view images.
- ◆ **The user must back up to return to the main page.** The image looks like a web page, but it isn't. No link or other explanatory text in the

image indicates how to get back to the web page. Most users know to click the browser's Back button, but don't assume all users know what to do.

Adding inline images using the `` tag

The alternative to providing links to images is to embed your images into the page. Figure 6-3 displays an example of this technique.

Figure 6-3:
The ship
image is
embedded
into the
page.



The code shows how this image was included into the page:

```
<!DOCTYPE html>
<html lang = "en-US">
  <head>
    <meta charset = "UTF-8">
    <title>embeddedImage.html</title>
  </head>
  <body>
    <h1>The Susan B. Constant</h1>
    <p>
      <img src = "shipStandard.jpg"
           height = "480"
           width = "640"
           alt = "Susan B. Constant" />
    </p>
    <p>
      The <em>Susan B. Constant</em> was flagship of the
      fleet of three small ships that brought settlers to Jamestown, the first
      successful English Colony in the new world. This is a replica housed
      near Jamestown, Virginia.
    </p>
  </body>
</html>
```

The image (`img`) tag is the star of this page. This tag allows you to grab an image file and incorporate it into the page directly. The image tag is a one-shot tag. It doesn't end with ``. Instead, use the `/>` characters at the end of the `img` definition to indicate that this tag doesn't have content.



You might have noticed that I italicized *Susan B. Constant* in the page, and I used the `` tag to get this effect. `` stands for *emphasis*, and `` means *strong emphasis*. By default, any text within an `` pair is italicized, and `` text is boldfaced. Of course, you can change this behavior with CSS.

The image tag has a number of important attributes, which I discuss in the following sections.

src (source)

The `src` attribute allows you to indicate the URL (Uniform Resource Locator) of the image. This can be an absolute or relative reference. Linking to an image in your own directory structure is generally best because you can't be sure an external image will still be there when the user gets to the page. (For more on reference types, turn to Chapter 5 of this minibook.)

height and width

The `height` and `width` attributes are used to indicate the size of the image. The browser uses this information to indicate how much space to reserve on the page.



The `height` and `width` attributes should *describe* the size of an image. You can use these attributes to actually change the size of an image, but it's a bad idea. Change the image size with your image editor (I show you how later in this chapter). If you use the `height` and `width` attributes, the user has to wait for the full image, even if she'll see a smaller version. Don't make the user wait for information she won't see. If you use these attributes to make the image larger than its default size, the resulting image has poor resolution. Find the image's actual size by looking at it in your image tool and use these values. If you leave out `height` and `width`, the browser determines the size automatically, but you aren't guaranteed to see the text until all the images have downloaded. Adding these attributes lets the browser format the page without waiting for the images.

alt (alternate text)

The `alt` attribute gives you an opportunity to specify alternate text describing the image. Alternate text information is used when the user has images turned off and by screen-readers. Information in alt tags is also used in image-searching software like Google Images.



The `alt` attribute is required on all images.

Additionally, the `` tag is an inline tag, so it needs to be embedded inside a block-level tag, like a `<p>` or ``.

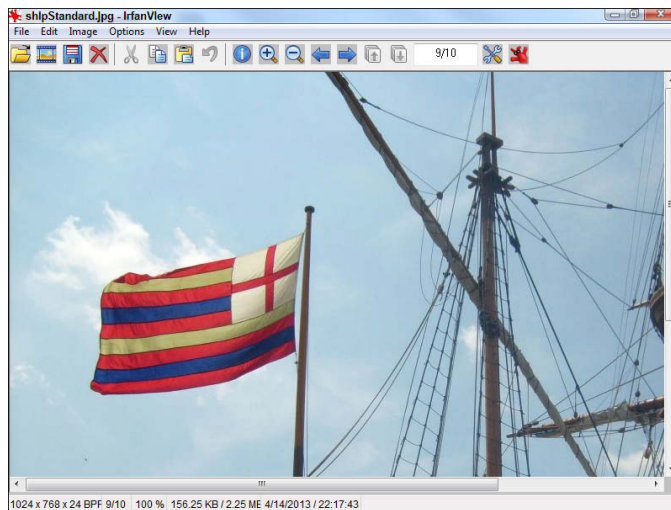
Choosing an Image Manipulation Tool

You can't just grab any old picture off your digital camera and expect it to work on a web page. The picture might work, but it could cause problems for your viewers. It's important to understand that *digital images* (any kind of images you see on a computer or similar device) are different from the kind of images you see on paper.

An image is worth 3.4 million words

Digital cameras and scanners are amazing these days. Even moderately priced cameras can now approach the resolution of old-school analog cameras. Scanners are also capable of taking traditional images and converting them into digital formats that computers use. In both cases, though, the default image can be in a format that causes problems. Digital images are stored as a series of dots, or *pixels*. In print, the dots are very close together, but computer screens have larger dots. Figure 6-4 shows how the ship image looks straight from the digital camera.

Figure 6-4:
Wow. That
doesn't look
like much.



My picture (taken on an older digital camera) registers at 6 megapixels (MP). That's a pretty good resolution, but modern digital cameras are much higher. If I print that picture on paper, all those dots are very tiny, and I get a nice picture. If I try to show the same picture on the computer screen, I see only

one corner. This actual picture came out at 2,816 pixels wide by 2,112 pixels tall. You only see a small corner of the image because the screen shots for this book are taken at 1024×768 pixels. Less than a quarter of the image is visible.

When you look at a large image in most browsers, it's automatically resized to fit the page. The cursor usually turns into some kind of magnifying glass, and if you click the image, you can see it in its full size or the smaller size.



Some image viewers take very large images and automatically resize them so they fit the screen. (This is the default behavior of Windows' default image viewer and most browsers.) The image may appear to be a reasonable size because of this feature, but it'll be huge and difficult to download in an actual web page. Make sure you know the actual size of an image before you use it.

Although shrinking an image so that it can be seen in its entirety is obviously beneficial, there's an even more compelling reason to do so. Each pixel on the screen requires 3 bytes of computer memory. (A *byte* is the basic unit of memory in a computer.) For comparison purposes, one character of text requires roughly 1 byte. The uncompressed image of the ship weighs a whopping 17 megabytes (MB). If you think of a word as five characters long, one picture straight from the digital camera takes up the same amount of storage space and transmission time as roughly 3,400,000 words. This image requires nearly three minutes to download on a 56K modem!

In a web page, small images are often shown at about 320×240 pixels, and larger images are often 640×480 pixels. If I use software to resample the image to the size I actually need and use an appropriate compression algorithm, I can get the image to look like Figure 6-5.



Figure 6-5:
The resized
image is
a lot more
manageable.