



ELOQUENT RUBY

Foreword by **Obie Fernandez**, *Series Editor*

RUSS OLSEN

Praise for *Eloquent Ruby*

“Reading *Eloquent Ruby* is like programming in Ruby itself: fun, surprisingly deep, and you’ll find yourself wishing it was always done this way. Wherever you are in your Ruby experience from novice to Rails developer, this book is a must read.”

—Ethan Roberts
Owner, Monkey Mind LLC

“*Eloquent Ruby* lives up to its name. It’s a smooth introduction to Ruby that’s both well organized and enjoyable to read, as it covers all the essential topics in the right order. This is the book I wish I’d learned Ruby from.”

—James Kebinger
Senior Software Engineer, PatientsLikeMe
www.monkeyatlarge.com

“Ruby’s syntactic and logical aesthetics represent the pinnacle for elegance and beauty in the ALGOL family of programming languages. *Eloquent Ruby* is the perfect book to highlight this masterful language and Russ’s blend of wit and wisdom is certain to entertain and inform.”

—Michael Fogus
Contributor to the Clojure programming
language and author of *The Joy of Clojure*

This page intentionally left blank

ELOQUENT RUBY

ELOQUENT RUBY

Russ Olsen

◆◆ Addison-Wesley

Upper Saddle River, NJ • Boston • Indianapolis • San Francisco
New York • Toronto • Montreal • London • Munich • Paris • Madrid
Capetown • Sydney • Tokyo • Singapore • Mexico City

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and the publisher was aware of a trademark claim, the designations have been printed with initial capital letters or in all capitals.

The author and publisher have taken care in the preparation of this book, but make no expressed or implied warranty of any kind and assume no responsibility for errors or omissions. No liability is assumed for incidental or consequential damages in connection with or arising out of the use of the information or programs contained herein.

The publisher offers excellent discounts on this book when ordered in quantity for bulk purchases or special sales, which may include electronic versions and/or custom covers and content particular to your business, training goals, marketing focus, and branding interests. For more information, please contact:

U.S. Corporate and Government Sales
(800) 382-3419
corpsales@pearsontechgroup.com

For sales outside the United States please contact:

International Sales
international@pearson.com

Visit us on the Web: www.informit.com/aw

Library of Congress Cataloging-in-Publication Data

Olsen, Russ.

Eloquent Ruby / Russ Olsen.
p. cm.

Includes index.

ISBN-13: 978-0-321-58410-6 (pbk. : alk. paper)

ISBN-10: 0-321-58410-4 (pbk. : alk. paper)

1. Ruby (Computer program language) I. Title.

QA76.73.R83O47 2011

005.13'3—dc22

2010048388

Copyright © 2011 Pearson Education, Inc.

All rights reserved. Printed in the United States of America. This publication is protected by copyright, and permission must be obtained from the publisher prior to any prohibited reproduction, storage in a retrieval system, or transmission in any form or by any means, electronic, mechanical, photocopying, recording, or likewise. For information regarding permissions, write to:

Pearson Education, Inc.
Rights and Contracts Department
501 Boylston Street, Suite 900
Boston, MA 02116
Fax: (617) 671-3447

ISBN-13: 978-0-321-58410-6

ISBN-10: 0-321-58410-4

Text printed in the United States on recycled paper at RR Donnelley in Crawfordsville, Indiana.

First printing, February 2011

*To My Dad
Charles J. Olsen
Who never had a chance to write a book of his own,
which is a shame because it would have been
hilarious*

This page intentionally left blank

Contents

Foreword xix

Preface xxi

Acknowledgments xxv

About the Author xxvii

PART I: The Basics 1

Chapter 1: Write Code That Looks Like Ruby 3

The Very Basic Basics 4

Go Easy on the Comments 6

Camels for Classes, Snakes Everywhere Else 8

Parentheses Are Optional but Are Occasionally Forbidden 9

Folding Up Those Lines 10

Folding Up Those Code Blocks 11

Staying Out of Trouble 12

In the Wild 13

Wrapping Up 15

Chapter 2: Choose the Right Control Structure 17

If, Unless, While, and Until 17

Use the Modifier Forms Where Appropriate 19

Use each, Not for 20

A Case of Programming Logic 21

Staying Out of Trouble 23
In the Wild 25
Wrapping Up 27

Chapter 3: Take Advantage of Ruby's Smart Collections 29

Literal Shortcuts 29
Instant Arrays and Hashes from Method Calls 30
Running Through Your Collection 33
Beware the Bang! 36
Rely on the Order of Your Hashes 38
In the Wild 38
Staying Out of Trouble 40
Wrapping Up 42

Chapter 4: Take Advantage of Ruby's Smart Strings 43

Coming Up with a String 44
Another API to Master 47
The String: A Place for Your Lines, Characters, and Bytes 49
In the Wild 50
Staying Out of Trouble 51
Wrapping Up 52

Chapter 5: Find the Right String with Regular Expressions 53

Matching One Character at a Time 54
Sets, Ranges, and Alternatives 55
The Regular Expression Star 57
Regular Expressions in Ruby 58
Beginnings and Endings 60
In the Wild 62
Staying Out of Trouble 63
Wrapping Up 64

Chapter 6: Use Symbols to Stand for Something 65

The Two Faces of Strings 65
Not Quite a String 66
Optimized to Stand for Something 67

In the Wild 69
Staying Out of Trouble 70
Wrapping Up 71

Chapter 7: Treat Everything Like an Object—Because Everything Is 73

A Quick Review of Classes, Instances, and Methods 74
Objects All the Way Down 76
The Importance of Being an Object 77
Public, Private, and Protected 79
In the Wild 81
Staying Out of Trouble 82
Wrapping Up 84

Chapter 8: Embrace Dynamic Typing 85

Shorter Programs, But Not the Way You Think 85
Extreme Decoupling 89
Required Ceremony Versus Programmer-Driven Clarity 92
Staying Out of Trouble 93
In the Wild 94
Wrapping Up 96

Chapter 9: Write Specs! 97

Test::Unit: When Your Documents Just Have to Work 98
A Plethora of Assertions 101
Don't Test It, Spec It! 101
A Tidy Spec Is a Readable Spec 104
Easy Stubs 105
... And Easy Mocks 107
In the Wild 108
Staying Out of Trouble 110
Wrapping Up 113

PART II: Classes, Modules, and Blocks 115

Chapter 10: Construct Your Classes from Short, Focused Methods 117

Compressing Specifications 117
Composing Methods for Humans 121

| | |
|------------------------|-----|
| Composing Ruby Methods | 122 |
| One Way Out? | 123 |
| Staying Out of Trouble | 126 |
| In the Wild | 127 |
| Wrapping Up | 128 |

Chapter 11: Define Operators Respectfully 129

| | |
|----------------------------|-----|
| Defining Operators in Ruby | 129 |
| A Sampling of Operators | 131 |
| Operating Across Classes | 134 |
| Staying Out of Trouble | 135 |
| In the Wild | 137 |
| Wrapping Up | 139 |

Chapter 12: Create Classes That Understand Equality 141

| | |
|--|-----|
| An Identifier for Your Documents | 141 |
| An Embarrassment of Equality | 142 |
| Double Equals for Everyday Use | 143 |
| Broadening the Appeal of the == Method | 145 |
| Well-Behaved Equality | 146 |
| Triple Equals for Case Statements | 149 |
| Hash Tables and the eql? Method | 150 |
| Building a Well-Behaved Hash Key | 152 |
| Staying Out of Trouble | 153 |
| In the Wild | 154 |
| Wrapping Up | 156 |

Chapter 13: Get the Behavior You Need with Singleton and Class Methods 157

| | |
|--|-----|
| A Stubby Puzzle | 158 |
| A Hidden, but Real Class | 160 |
| Class Methods: Singletons in Plain Sight | 162 |
| In the Wild | 164 |
| Staying Out of Trouble | 165 |
| Wrapping Up | 167 |

Chapter 14: Use Class Instance Variables 169

- A Quick Review of Class Variables 169
- Wandering Variables 171
- Getting Control of the Data in Your Class 174
- Class Instance Variables and Subclasses 175
- Adding Some Convenience to Your Class Instance Variables 176
- In the Wild 177
- Staying Out of Trouble 179
- Wrapping Up 179

Chapter 15: Use Modules as Name Spaces 181

- A Place for Your Stuff, with a Name 181
- A Home for Those Utility Methods 184
- Building Modules a Little at a Time 185
- Treat Modules Like the Objects That They Are 186
- Staying Out of Trouble 189
- In the Wild 190
- Wrapping Up 191

Chapter 16: Use Modules as Mixins 193

- Better Books with Modules 193
- Mixin Modules to the Rescue 195
- Extending a Module 197
- Staying Out of Trouble 198
- In the Wild 202
- Wrapping Up 205

Chapter 17: Use Blocks to Iterate 207

- A Quick Review of Code Blocks 207
- One Word after Another 209
- As Many Iterators as You Like 210
- Iterating over the Ethereal 211
- Enumerable: Your Iterator on Steroids 213
- Staying Out of Trouble 215
- In the Wild 217
- Wrapping Up 218

Chapter 18: Execute Around with a Block 219

- Add a Little Logging 219
- When It Absolutely Must Happen 224
- Setting Up Objects with an Initialization Block 225
- Dragging Your Scope along with the Block 225
- Carrying the Answers Back 227
- Staying Out of Trouble 228
- In the Wild 229
- Wrapping Up 231

Chapter 19: Save Blocks to Execute Later 233

- Explicit Blocks 233
- The Call Back Problem 234
- Banking Blocks 236
- Saving Code Blocks for Lazy Initialization 237
- Instant Block Objects 239
- Staying Out of Trouble 240
- In the Wild 243
- Wrapping Up 244

PART III: Metaprogramming 247**Chapter 20: Use Hooks to Keep Your Program Informed 249**

- Waking Up to a New Subclass 250
- Modules Want To Be Heard Too 253
- Knowing When Your Time Is Up 255
- . . . And a Cast of Thousands 256
- Staying Out of Trouble 257
- In the Wild 259
- Wrapping Up 261

Chapter 21: Use `method_missing` for Flexible Error Handling 263

- Meeting Those Missing Methods 264
- Handling Document Errors 266
- Coping with Constants 267
- In the Wild 268

Staying Out of Trouble 270

Wrapping Up 271

Chapter 22: Use `method_missing` for Delegation 273

The Promise and Pain of Delegation 274

The Trouble with Old-Fashioned Delegation 275

The `method_missing` Method to the Rescue 277

More Discriminating Delegation 278

Staying Out of Trouble 279

In the Wild 281

Wrapping Up 283

Chapter 23: Use `method_missing` to Build Flexible APIs 285

Building Form Letters One Word at a Time 286

Magic Methods from `method_missing` 287

It's the Users That Count—All of Them 289

Staying Out of Trouble 289

In the Wild 290

Wrapping Up 292

Chapter 24: Update Existing Classes with Monkey Patching 293

Wide-Open Classes 294

Fixing a Broken Class 295

Improving Existing Classes 296

Renaming Methods with `alias_method` 297

Do Anything to Any Class, Anytime 299

In the Wild 299

Staying Out of Trouble 303

Wrapping Up 303

Chapter 25: Create Self-Modifying Classes 305

Open Classes, Again 305

Put Programming Logic in Your Classes 308

Class Methods That Change Their Class 309

In the Wild 310

Staying Out of Trouble 314
Wrapping Up 315

Chapter 26: Create Classes That Modify Their Subclasses 317

A Document of Paragraphs 317
Subclassing Is (Sometimes) Hard to Do 319
Class Methods That Build Instance Methods 321
Better Method Creation with `define_method` 324
The Modification Sky Is the Limit 324
In the Wild 327
Staying Out of Trouble 330
Wrapping Up 332

PART IV: Pulling It All Together 333

Chapter 27: Invent Internal DSLs 335

Little Languages for Big Problems 335
Dealing with XML 336
Stepping Over the DSL Line 341
Pulling Out All the Stops 344
In the Wild 345
Staying Out of Trouble 347
Wrapping Up 349

Chapter 28: Build External DSLs for Flexible Syntax 351

The Trouble with the Ripper 352
Internal Is Not the Only DSL 353
Regular Expressions for Heavier Parsing 356
Treetop for Really Big Jobs 358
Staying Out of Trouble 360
In the Wild 362
Wrapping Up 364

Chapter 29: Package Your Programs as Gems 367

Consuming Gems 367
Gem Versions 368

| | |
|------------------------------------|-----|
| The Nuts and Bolts of Gems | 369 |
| Building a Gem | 370 |
| Uploading Your Gem to a Repository | 374 |
| Automating Gem Creation | 375 |
| In the Wild | 376 |
| Staying Out of Trouble | 377 |
| Wrapping Up | 380 |

Chapter 30: Know Your Ruby Implementation 381

| | |
|--|-----|
| A Fistful of Rubies | 381 |
| MRI: An Enlightening Experience for the C Programmer | 382 |
| YARV: MRI with a Byte Code Turbocharger | 385 |
| JRuby: Bending the “J” in the JVM | 387 |
| Rubinius | 388 |
| In the Wild | 389 |
| Staying Out of Trouble | 389 |
| Wrapping Up | 390 |

Chapter 31: Keep an Open Mind to Go with Those Open Classes 391**Appendix: Going Further 393**

| | |
|--------------|-----|
| <i>Index</i> | 397 |
|--------------|-----|

This page intentionally left blank

Foreword

Do you know why experienced Ruby programmers tend to reach for basic collections and hashes while programmers from other languages go for more specialized classes? Do you know the difference between `strip`, `chop`, and `chomp`, and why there are three such similar methods when apparently one might suffice? (Not to mention `lstrip` and `rstrip`!) Do you know the downsides of dynamic typing? Do you know why the differences between strings and symbols get so blurry, even to experienced Ruby developers? How about metaprogramming? What the heck is an `eigenclass`? How about protected methods? Do you know what they're really about? Really? Are you sure?

Russ knows all that stuff and more. And if books are like babies, then Russ is that experienced mom who pops out her second child after a couple of hours of labor and is back at work a week later in her pre-pregnancy clothes as if nothing out of the ordinary happened. You know: the one all the other moms talk about in hushed tones of disbelief and reverence. That's the way my series authors discuss Russ.

Not that there's anything small or insignificant about Russ' bouncing new baby . . . eh, I mean book. On the contrary, weighing in at just over 400 pages, this tome is slightly larger than its older sibling *Design Patterns in Ruby*. The family resemblance is crystal clear: Russ is first and foremost your friend. His approachable writing style makes even the driest Ruby language topics engaging and funny. Like the way that symbols remind Russ "of the eyes peering out from the tilted head of a confused but friendly dog."

Truth is, we need this kind of book now more than ever. Ruby has hit the mainstream with the force of a Hulk Smash, and the masses are paddling along well-known routes without full (heck, sometimes any) understanding of what makes their favorite

frameworks and library APIs so vibrant and navigable. So for those not content with the basics, those who want to go beyond shallow understanding, this book goes deep. It helps readers achieve true mastery of Ruby, a programming language with some of the deepest, darkest pools of nuance and texture of all the major languages of modern times.

I know you're going to enjoy this book, just like I did. And if you do, please join me in encouraging Russ to get knocked up again soon.

—Obie Fernandez, Professional Ruby Series Editor

Preface

I've taught a fair number of Ruby classes over the years, but one particular class stands out in my mind. Class was over, and as I was going out the door one of my students, an experienced Java programmer, stopped me and voiced a complaint that I have heard many times since. He said that the hardest part of learning Ruby wasn't the syntax or the dynamic typing. Oh, he could write perfectly correct Ruby, sans semicolons and variable declarations. The problem was that something was missing. He constantly found himself falling back into his same old Java habits. Somehow his Ruby code always ended up looking much like what he would have written in Java. My answer to him was not to worry, you haven't missed anything—you just aren't done learning Ruby.

What does it mean to learn a new programming language? Clearly, like my frustrated student, you need to understand the basic rules of the grammar. To learn Ruby you need to be aware that a new line usually starts a new statement, that a class definition starts with the word `class`, and that variable names start with a lowercase letter—unless they start with an `@`. But you can't really stop there. Again, like my erstwhile student you will also need to know what all of that code does. You'll need to know that those statements are really expressions (since they all return a value) and that all of those classes starting with the `class` keyword can change over time. And you'll need to know why those `@variables` are different from the plain vanilla variables.

But the punch line is that even after you master all of this, you are still not quite there. It turns out that computer languages share something fundamental with our everyday order-a-pizza human tongues: Both kinds of languages are embedded in a culture, a way of thinking about the world, an approach to solving problems. A formal

understanding of the mechanics of Ruby isn't the same as really looking at the programming world through Ruby-colored glasses. You need to absorb the cultural part of Ruby, to see how real Rubyists use the language to solve problems.

This is a book about making that final leap, about absorbing the Ruby programming culture, about becoming truly fluent in Ruby. The good news is that for most people the final step is the best part of learning Ruby—a series of “Ah ha!” moments—as it suddenly becomes clear why those funny symbol things exist, why classes are never final, and how this wonderful language works so hard to just stay out of your way.

Who Is This Book For?

This book is for you if you have a basic understanding of Ruby but feel that you haven't quite gotten your arms around the language. If you find yourself wondering what anyone could possibly do with all those odd language features that seem so important to Ruby, keep reading.

This book is also for you if you are a programmer with experience in other object oriented languages, perhaps Java or C# or Python, and you want to see what this Ruby thing is all about. While I'm not going to explain the basic details of Ruby in this book, the basics of Ruby are really very basic indeed. So, if your learning style involves simply jumping into the deep end, welcome to the pool.

How Is This Book Organized?

Mostly, this book works from small to large. We will start with the most tactical questions and work our way up to the grand strategy behind pulling whole Ruby projects together. Thus the first few chapters will concentrate on one statement, one method, one test, and one bug-sized issue:

- How do you write code that actually looks like Ruby?
- Why does Ruby have such an outsized collection of control structures?
- Why do Ruby programmers use so many hashes and arrays in their code?
- How do I get the most out of Ruby's very powerful strings and regular expressions?
- What are those symbol things, and what do you do with them?

- Is everything in Ruby really an object?
- How do I take advantage of dynamic typing?
- How can I make sure that my code actually works?

From there we will move on to the bigger questions of building methods and classes:

- Why are Ruby classes so full of tiny little methods?
- Why would you overload an operator? And, more importantly, why would you not?
- Do I really need to care about object equality?
- What good is a module?
- Can I really assign a method to an individual object? And what does that have to do with class methods?
- How do I hang some data on a class?
- How do you use blocks to good effect?
- Why would you ever call a method that doesn't actually exist?
- Can I really get notified when a class gets created? Why would I do that?
- Can I really modify classes on the fly? Why would I do that?
- Can I really write code that writes code? Why would I do that?

Finally, we will look at some of the techniques you can use to pull your programming project together into a unified whole:

- Would you really build a whole language simply to solve an ordinary programming problem?
- How do I make it easy for others to use my work?
- How does my Ruby implementation work?
- Where do I go from here?

About the Code Examples

The trouble with writing books about programming is that all the interesting stuff is in a constant state of flux. This is, after all, what makes it interesting. Certainly Ruby is something of a moving target these days: Although the great bulk of the Ruby code base was written for Ruby 1.8.X, version 1.9 has been out for some time and is clearly the future. In the pages that follow I have tried to split the coding difference by writing all of the examples in the 1.9 dialect,¹ taking care to note where Ruby 1.8 would be different. The good news is that there aren't all that many differences.

I have also consistently used the traditional `pp` command to print out more complex objects. However, to keep from driving everyone² crazy, I'm not going to endlessly repeat the `require 'pp'` call needed to make `pp` work. Just assume it is there at the top of each example.

1. Specifically, the examples all use Ruby-1.9.1-p430.

2. Especially me!

- [**read The Reformed Vampire Support Group pdf, azw \(kindle\), epub**](#)
- [*click The Laughing Monsters: A Novel*](#)
- [Step-by-step Knife Making pdf, azw \(kindle\)](#)
- [Them: A Novel here](#)
- [download online Minecraft: The Unlikely Tale of Markus "Notch" Persson and the Game that Changed Everything](#)

- <http://tuscalaural.com/library/The-Reformed-Vampire-Support-Group.pdf>
- <http://drmurphreesnewsletters.com/library/Meteor-in-Action.pdf>
- <http://nautickim.es/books/Step-by-step-Knife-Making.pdf>
- <http://aneventshop.com/ebooks/Burning-Paradise.pdf>
- <http://fitnessfatale.com/freebooks/Staying-Strong--365-Days-a-Year.pdf>