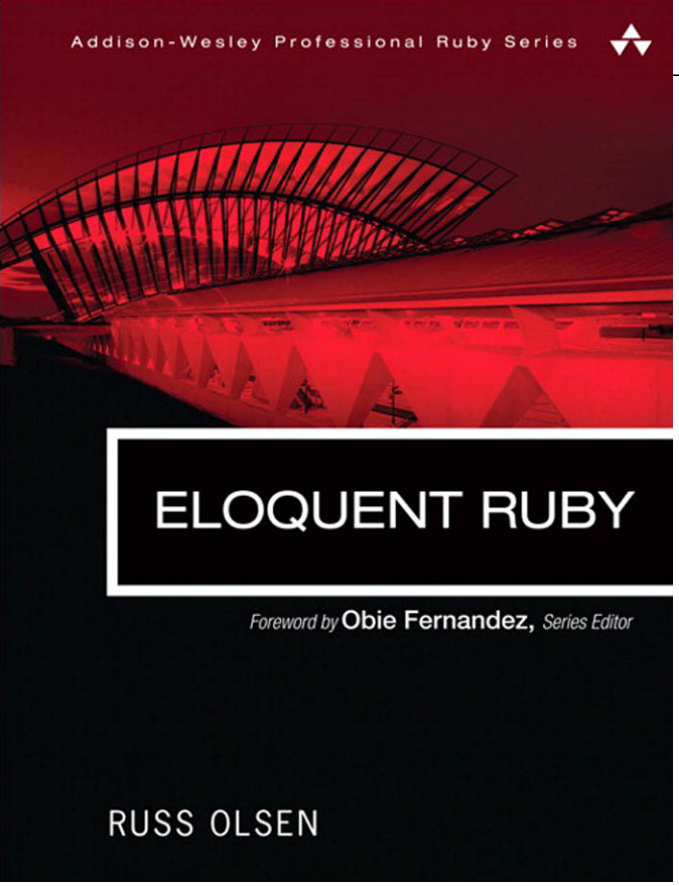
Addison-Wesley Professional Ruby Series
ELOQUENT RUBY
Foreword by Obie Fernandez, Series Editor
RUSS OLSEN

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

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

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

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 everyone[2] 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!