Practical Ruby Projects: Ideas for the Eclectic Programmer

ISBN-13 (pbk): 978-1-59059-911-2

ISBN-10 (pbk): 1-59059-911-X

ISBN-13 (electronic): 978-1-4302-0470-1

ISBN-10 (electronic): 1-4302-0470-2

Printed and bound in the United States of America 9 8 7 6 5 4 3 2 1

Lead Editors: Chris Mills and Tom Welsh
Technical Reviewer: Ben Matasar
Editorial Board: Steve Anglin, Ewan Buckingham, Tony Campbell, Gary Cornell, Jonathan Gennick, Jason Gilmore, Kevin Goff, Jonathan Hassell, Matthew Moodie, Joseph Ottinger, Jeffrey Pepper, Ben Renow-Clarke, Dominic Shakeshaft, Matt Wade, Tom Welsh
Project Manager: Candace English
Copy Editor: Kim Benbow
Associate Production Director: Kari Brooks-Copony
Production Editor: Laura Esterman
Compositor: Molly Sharp, ContentWorks
Proofreader: Martha Whitt
Indexer: Carol Burbo
Cover Designer: Kurt Krames
Manufacturing Director: Tom Debolski

Distributed to the book trade worldwide by Springer-Verlag New York, Inc., 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax 201-348-4505, e-mail `orders-ny@springer-sbm.com`, or visit `http://www.springeronline.com`.

For information on translations, please contact Apress directly at 2855 Telegraph Avenue, Suite 600, Berkeley, CA 94705. Phone 510-549-5930, fax 510-549-5939, e-mail `info@apress.com`, or visit `http://www.apress.com`.

The source code for this book is available to readers at `http://www.apress.com`.

CHAPTER 1

Introduction

This book is titled *Practical Ruby Projects*. And let me start by saying that the projects *are* practical. But they might not be quite what you're used to. Flip through the book. You won't find any references to enterprise deployment. Not a word about business logic. In fact, hard as it is to believe, there's no web programming! But if you exclude those things, what's left? Why, everything else, of course!

Each chapter in this book turns Ruby loose on a new interesting problem or project. They range from creative endeavors to investigative simulations to the exploration of computer programming languages themselves. Ruby is a programming language, but it's also a tool to create, understand, and entertain. This book is all about Ruby.

Why Ruby?

Since this book was written with the assumption that you have a basic knowledge of Ruby, odds are you already know about Ruby's strengths.

The Language

You know that Ruby's blocks are a joy to use. You know how Ruby's programmer-oriented core API can make programming feel effortless. Despite what the popular press sometimes says, Ruby isn't the final word in programming languages. But Ruby holds a unique position in the current landscape.

Borrowing from the Smalltalk tradition, Ruby brings a new level of purity to the world of contemporary object-oriented languages that includes Java and Python. It has also brought the concision and utility of Perl to the world of structured development. Finally, it's captured some of the dynamism of Smalltalk and introduced it to the current programming landscape.

It's a wonderful language for hacking, design, and programming, not to mention an excellent tool for scripting, text-processing, and system administration. Combined with the Ruby on Rails web development buzz, Ruby's future is promising, particularly with progress toward a faster runtime environment.

The Community

I first encountered Ruby in 2004 while working at Intel in Hillsboro, Oregon. The approved higher-level languages were Perl and Ruby. I was a Python programmer at the time and felt a little bit threatened by Ruby's supposed elegance. But I knew Perl well enough to know I was going to want to learn Ruby.

It was an exhilarating experience. In between maintaining legacy Perl modules, I started plowing through the pickaxe book (*Programming Ruby: The Pragmatic Programmer's Guide* by Dave Thomas with Chad Fowler and Andy Hunt [Pragmatic Bookshelf, 2004, 2nd Edition]). And, at some point, I stopped reaching for Python in my personal projects and started turning to Ruby.

That's when I went to my first Portland Ruby Brigade (PDX.rb) meeting. Which brings me to Ruby's second strength: its community. Now, every language community has its own flavor and culture. Maybe it is just because it's a fresh language with the right set of features, but the programmers you meet in the Ruby track at conferences, the hackers at your local Ruby Brigade, and the guy down the hall at work sneaking Ruby into the system all seem to have something in common. They're curious, reflective, and lighthearted, but they're also highly effective programmers. And they're all working on some kind of project. It'll be born of personal interest, but odds are it will be shared—and adopted. That's just the community standard around here!

Why do Rubyists choose Ruby? Probably because it gets their work done. But I suspect that the project culture is part of it. This book was inspired by the amazing Rubyists out there hacking on their own projects and sharing them with the world.

Why This Book?

Whether you maintain a host of Ruby libraries, simply tinker on your own code at night, or are just getting started with Ruby and looking for new ideas, you're part of this select and curious project culture. This book is a collection of ideas that excite me, which are interesting to code and understand on their own. They're also great stepping stones for deeper work or even potential sources of ideas to mine for your projects, not to mention that most of the chapters touch on the strange and interesting corner cases of the Ruby programming language.

Unlike an introductory book, this is a project book, and the chapters are designed to be mostly independent (although a few are complementary). So if a chapter looks good to you, skip right to it! Here's what to expect.

In Chapter 2, you'll use Ruby to play and compose music and briefly discuss live coding music as a performance art. In the process, you'll use Ruby's dynamic linking interface to call directly into C code, letting you build a cross-platform MIDI library that works on Windows, Mac, and Linux.

Chapter 3 focuses on using Ruby to build animations programmatically. You'll use scalable vector graphics (SVG) to describe shapes and pictures that will be rendered into frames and ultimately combined into movies. By the end you'll have a distinctly pro-Ruby animation.

Chapter 4 uses simulation to explore the world of pocket change. Ever wondered if we could make better change and carry fewer coins if we had a different system of denominations? You'll use Ruby to build a simulator to answer that question. In the process, you'll look at how Ruby can help you learn about the world.

Chapter 5 is all about games, turn-based strategy games to be specific. You'll experiment using a very loosely coupled system to model the complex rules of a strategy game and build the core game engine.

In Chapter 6, you'll take the game engine from Chapter 5 and put a beautiful interface on it using RubyCocoa for Mac OS X. You'll learn about Objective-C, Cocoa, runtime bridges, and, of course, do a lot of GUI programming.

Chapter 7 focuses on genetic algorithms. Inspired by the process of evolution, genetic algorithms are an interesting technique for exploring large search spaces when solving problems. You'll cook up an implementation in Ruby and then turn it loose on the coin problem from Chapter 4. It will let you tackle much larger problems than you could previously.

Chapter 8 explores what it is that makes a programming language, while also delving into Lisp. By the end of the chapter, not only will you have your own Lisp interpreter (written in Ruby), but also an improved understanding of both languages! And, of course, you'll have insight into how to build your very own programming language.

Chapter 9 looks at the art of parsing text. This often overlooked skill is an indispensible part of any programmer's toolbox. You'll address it in the context of programming languages (building on Chapter 8) as well as exploring new syntactic ground, but the tricks learned will be applicable to a wide range of everyday text-processing problems.

And as I mentioned, each chapter is designed to be explored on its own, extended for future work, or even mined for ideas related to other original, independent concepts.

Getting Set Up

You're obviously going to need Ruby installed! This book was written using the Ruby 1.8 series. The code was tested on Ruby 1.8.5, but it should work on any 1.8 release. Time will tell how well it bridges the gap to 2.0. (I'm optimistic.)

Ruby is available for most major operating systems from its web site: `www.ruby-lang.org/`. There are detailed instructions for each platform, but the basic idea is that Windows users should use the installer bundle, Linux users should use their distribution's package manager, and Mac users can choose between an installer or a package manager like MacPorts.

You'll also want RubyGems installed. RubyGems is the convenient system for managing Ruby libraries. Depending on how you installed Ruby, you may or may not have RubyGems already installed. You can easily check by typing the following into irb:

```
require 'rubygems'
```

■ **Tip** *irb* is the interactive Ruby environment. Type some Ruby in and see the result. You'll probably need to open a terminal or command prompt, and then type **irb** to launch it.

If you get a LoadError, you'll need to install RubyGems. There are excellent directions available at `www.rubygems.org/read/chapter/3`.

Once you've installed RubyGems, installing any of the gems mentioned in the following chapters is as simple as typing (at the terminal or command prompt):

```
gem install GEMNAME
```

If you'd like to install all the gems before you start, you can install the midilib, sexp, rparsec, and extensions gems.

Source Code in This Book

Most computer books have some source code kicking around inside their covers. This book has a lot of it. Source code is presented throughout the book in monospaced font. Method definitions are usually written in open class style, so they can be cumulatively executed in an irb session or sequentially added to a file (most of the projects only require a single file for code).

The bundled versions of the source code for each chapter are available online. Each chapter is provided in a separate directory containing multiple versions of the source files moving through time. Each new section of code is successively integrated into each new source file. So while the chapters provide a walk through the code, you can also look at how it all fits together at each step in the process.

While following along in the text, there are a few helpful conventions to be aware of. In most cases, each line of Ruby code fits on a printable line. However, in a few cases, I have been forced to break lines on the page. You'll recognize this by the ➥ symbol.

This is not to be confused with the transformation symbol ➤. This symbol is used in various sections of imperative code (for demonstration purposes) to show the return value of an evaluated expression. For example:

```
1 + 1 ➤ 2
```

Your Projects

While the ideas in this book are useful and exciting concepts, the best projects always come from your own interests. I hope these projects are engaging and fun, but I also hope they're a place from which to explore.

I'm looking forward to seeing your projects in the Ruby community. Blog, publish, speak—whatever works best. I can't wait to see what you're working on!