

**Learn**

the basics of object-oriented programming using Ruby

**Apply**

your knowledge in the real world

**SAMS**  
**Teach Yourself**

# Ruby

in **21** Days

**SAMS**

---

Mark Slagell

**SAMS**  
**Teach Yourself**

# **Ruby**

in **21** Days

**SAMS**

*800 East 96th St., Indianapolis, Indiana, 46240*

---

# Sams Teach Yourself Ruby in 21 Days

## Copyright ©2002 by Sams Publishing

All rights reserved. No part of this book shall be reproduced, stored in a retrieval system, or transmitted by any means, electronic, mechanical, photocopying, recording, or otherwise, without written permission from the publisher. No patent liability is assumed with respect to the use of the information contained herein. Although every precaution has been taken in the preparation of this book, the publisher and author assume no responsibility for errors or omissions. Nor is any liability assumed for damages resulting from the use of the information contained herein.

International Standard Book Number: 0-672-32252-8

Library of Congress Catalog Number: 2001094220

Printed in the United States of America

First Printing: March 2002

04 03 02 01                    4 3 2 1

### Trademarks

All terms mentioned in this book that are known to be trademarks or service marks have been appropriately capitalized. Sams Publishing cannot attest to the accuracy of this information. Use of a term in this book should not be regarded as affecting the validity of any trademark or service mark.

### Warning and Disclaimer

Every effort has been made to make this book as complete and as accurate as possible, but no warranty or fitness is implied. The information provided is on an "as is" basis. The author and the publisher shall have neither liability nor responsibility to any person or entity with respect to any loss or damages arising from the information contained in this book.

#### ASSOCIATE PUBLISHER

Mark Taber

#### ACQUISITIONS EDITOR

Katie Purdum

#### DEVELOPMENT EDITOR

Maryann Steinhart

#### MANAGING EDITOR

Charlotte Clapp

#### PROJECT EDITOR

Matt Purcell

#### COPY EDITOR

Jerome Colburn  
(Publication Services, Inc.)

#### INDEXER

Jason Mortenson  
(Publication Services, Inc.)

#### PRODUCTION EDITOR

Theodore Young, Jr.  
(Publication Services, Inc.)

#### PROOFREADER

Phil Hamer  
(Publication Services, Inc.)

#### TECHNICAL EDITORS

Gene Winston  
Dave Thomas  
Daniel Solin

#### TEAM COORDINATOR

Amy Patton

#### INTERIOR DESIGNER

Gary Adair

#### COVER DESIGNER

Aren Howell

#### PAGE LAYOUT

Nina Betterly  
Jennifer Faaborg  
Jim Torbit  
Michael Tarleton  
Jessica Vonasch  
(Publication Services, Inc.)

---

# Contents at a Glance

Introduction	1
<b>Week 1 The Fundamentals</b>	<b>9</b>
Day 1 Getting Started with Ruby	11
2 Hello, Objects!	27
3 Containers	51
4 Iteration and Flow Control	69
5 The Characteristics of Objects	93
6 Ins and Outs	113
7 Catching Up on Details	137
<b>Week 2 Power Scripting</b>	<b>161</b>
Day 8 Pattern Matching	163
9 Inheritance and Modules	185
10 Program File Layout, Program Design, and the General Case	203
11 Modules and Classes in Depth	223
12 An Introduction to Recursion	247
13 Mastering the Operating System	265
14 Arguments, Blocks, and Procs	289
<b>Week 3 Making It Work for You</b>	<b>311</b>
Day 15 Toward Habitable Interfaces	313
16 Putting It Together (Part I)	337
17 Ruby/Tk	355
18 Ruby/Gtk	389
19 Some Advanced Topics (That Aren't So Hard)	427
20 Working with the Web	449
21 Putting It together (Part II)	475

---

<b>Appendices</b>	<b>499</b>
A irb Results	501
B Installation Help	503
C Debugging, with and without a Debugger	509
D Essential Vocabulary	517
Index	521

---

# Contents

<b>Introduction</b>	<b>1</b>
<b>Week 1 The Fundamentals</b>	<b>9</b>
<b>Day 1 Getting Started with Ruby</b>	<b>11</b>
Why Ruby? .....	11
Ruby Is Small and Intuitive.....	12
Ruby Gives You Immediate Feedback .....	12
Ruby Is Free .....	12
Ruby Is Portable .....	12
Ruby Is Object-Oriented .....	13
Ruby Is a Scripting Language.....	13
Ruby Supports Regular Expressions .....	14
Make Sure You Have Ruby.....	14
First Steps .....	14
Life in the Command Line .....	15
Choosing a Text Editor .....	15
Our First Script .....	16
Some Experiments.....	16
Play-by-Play .....	17
Remarks About the Experimental Results.....	18
The <i>Shebang</i> Line .....	20
Using Ruby Interactively .....	21
An Interactive Tool: <code>irb</code> .....	22
The <code>eval.rb</code> Script .....	23
Summary .....	24
Is There Homework in This Class? .....	24
Exercises .....	25
Answers .....	25
<b>Day 2 Hello, Objects!</b>	<b>27</b>
What Is an Object? .....	27
The First Object: <code>self</code> .....	28
Teach Yourself Tricks .....	29
The Second Object: <code>other</code> .....	30
Classes.....	32
Making Instances of a Class .....	33
Appending Methods to a Class .....	34
Altering and Removing Methods .....	35

Everybody's Methods .....	36
Who Is This "self" Anyway? .....	37
Method Arguments .....	38
Local Scope for Variables .....	40
Communication Within an Object.....	42
Communication Between Different Objects .....	44
Identifiers and Variables .....	45
Name Tags, Not Suitcases .....	45
Summary .....	47
Exercises .....	48
Answers .....	48
<b>Day 3 Containers</b> .....	<b>51</b>
A Change in Convention.....	51
Some Words About Ambiguity .....	52
Back to Business.....	54
Strings .....	54
Specifying Substrings by Position.....	54
Individual Characters.....	56
Specifying Substrings by Matching .....	57
A Few Useful String Instance Methods.....	58
Arrays .....	59
Arrays Containing Arrays .....	61
FIFOs and Stacks.....	62
Stacks.....	63
A Few Useful Array Instance Methods.....	63
Hashes .....	64
A Few Useful Hash Instance Methods.....	65
Ranges.....	66
Summary .....	67
Exercises .....	67
Answers .....	67
<b>Day 4 Iteration and Flow Control</b> .....	<b>69</b>
Using Iterators.....	69
each .....	70
Variations on Iterator Calls .....	71
times, downto, upto, step .....	72
each_byte .....	73
each_index, each_with_index .....	74
each_pair, each_key, each_value for Hashes .....	74
select, map .....	75
Flow Control.....	76
Conditional Code.....	76

Loops .....	78
A Text Filter .....	78
Interrupting the Flow .....	80
Odd-Position Elements .....	81
Testing Multiple Conditions .....	83
Getting a Value from <code>if</code> or <code>case</code> .....	84
Grouping Several Expressions into One .....	86
Why Loop When You Can Iterate? .....	86
Summary .....	86
Exercises .....	87
Answers .....	89
<b>Day 5 The Characteristics of Objects</b> .....	<b>93</b>
Instance Variables .....	94
Mars and the Metric System: A Cautionary Tale .....	95
Writing a Temperature Class .....	96
Integers and Floats.....	98
A Convention for Naming Methods .....	98
Automatic Accessors .....	99
Define Your Own Operators .....	100
Class Constants .....	103
Access Control (or, A Cure for the Common Code).....	105
The <code>initialize</code> Method .....	105
Global Variables .....	107
What to Name Things.....	108
Exercises .....	109
Answers .....	110
<b>Day 6 Ins and Outs</b> .....	<b>113</b>
Streams .....	113
The Standard Streams .....	114
Files .....	116
Some Useful IO Methods .....	117
What About Memory Space? .....	120
Formatted Output .....	121
Class Methods.....	123
Some Useful <code>File</code> Class Methods.....	125
The Command Line .....	127
Class Variables .....	128
Errors and Exceptions.....	130
Summary .....	132
Exercises .....	132
Answers .....	133



---

<b>Day 7 Catching Up on Details</b>	<b>137</b>
Numbers in Ruby .....	138
Integer Literals.....	138
Floating-Point Literals.....	139
How Numbers Are Stored .....	140
How Numbers Are Presented .....	142
Binary Arithmetic .....	143
Boolean Logic.....	146
Short-Circuit Evaluation .....	147
Strings and String Literals .....	148
Block Scope for Local Variables .....	151
Shortcuts and Tricks .....	151
Variable Modification .....	151
Chained Assignment .....	152
Multiple Assignment .....	153
Functional and Imperative Styles .....	155
Garbage Collection .....	155
Gotchas .....	156
Summary .....	157
Exercises .....	157
Answers .....	159
<b>Week 2 Power Scripting</b>	<b>161</b>
<b>Day 8 Pattern Matching</b>	<b>163</b>
Simple Pattern Matching .....	164
Wildcards and Character Classes .....	165
Character Ranges.....	166
Negation.....	166
Abbreviations.....	167
Position Anchors.....	168
Repetition .....	169
Greed .....	171
Grouping .....	172
Grouping and Repetition .....	172
Grouping and Alternation .....	176
Grouping and Memory .....	177
Switches .....	178
Case Insensitivity: /i.....	178
Extended Legibility: /x .....	178
Multiline Matching: /m .....	178
Some Container Methods That Use Regexes .....	178

Regexes and Matches As Objects .....	180
Summary .....	181
Exercises .....	181
Answers .....	182
<b>Day 9 Inheritance and Modules</b> .....	<b>185</b>
Organizing Organization .....	185
The Make-Up of Ruby Classes: Some “What” and a Little “How” .....	186
The Basics .....	186
Inheritance .....	187
Fiddling with Inheritance .....	189
Play It Again, Ruby: Another Method Call Example .....	193
Modules .....	195
Resolving Method Ambiguities .....	197
Summary .....	198
Exercises .....	198
Answers .....	200
<b>Day 10 Program File Layout, Program Design, and the General Case</b> .....	<b>203</b>
Program and File Interaction .....	204
Runtime Extension of Ruby’s World: <code>require</code> .....	205
The Argument to <code>require</code> .....	208
Compiled Extensions .....	209
<code>require</code> and Variables .....	210
<code>require</code> Versus <code>include</code> .....	211
Examining the Ruby Installation .....	211
Abstraction and the General Case .....	213
Under- and Over-Abstraction .....	214
Getting Abstraction Right .....	215
Summary .....	218
Exercises .....	218
Answers .....	220
<b>Day 11 Modules and Classes in Depth</b> .....	<b>223</b>
Designing Modules and Classes for Clarity and Reuse .....	223
Some Class/Module Distinctions .....	224
An Exercise in Adjectival Thinking .....	226
Code Reusability .....	229
Embedded Modules and Namespace Management .....	229
Modules Mixing in Modules .....	231
Classes Defined Inside Modules .....	232
Classes Defined Inside Classes .....	233
Class-Module Distribution Across Program Files .....	234

Overriding Methods .....	235
Overriding and Aliasing .....	236
Overriding an Inherited Method .....	237
Handling Arguments in Overridden Methods .....	239
Summary .....	239
Exercises .....	240
Answers .....	244
<b>Day 12 An Introduction to Recursion</b> .....	<b>247</b>
The Canonical Starting Point: Factorials .....	247
A Little Too Much Like Magic? .....	249
Recursive Functions .....	250
Efficiency Concerns .....	250
Memoization .....	252
The Towers of Hanoi .....	254
Summary .....	261
Exercises .....	261
Answers .....	263
<b>Day 13 Mastering the Operating System</b> .....	<b>265</b>
Motivation .....	265
Portability Notes .....	266
Gathering Information .....	267
Treating Programs As Functions .....	269
Extended Conversations .....	272
The Art of Instant Reproduction .....	275
Waiting for Children .....	277
Pipes.....	277
How to Control Your Children .....	278
Example Spinner #1, Using <code>fork</code> .....	279
Example Spinner #2, Using <code>IO.popen</code> .....	281
Example Spinner #3, Using <code>Kernel.open</code> .....	283
Summary .....	283
Exercises .....	284
Answers .....	285
<b>Day 14 Arguments, Blocks, and Procs</b> .....	<b>289</b>
Life Without Iterators .....	290
Stealthy Approach .....	291
A Hands-on Approach .....	292
A Hands-off Approach .....	293
Final Approach: Very Hands-off .....	295

Writing Iterator Methods .....297  
     Hybrid Iterators .....298  
     New Iterators for Old Classes .....299  
     Was This Trip Really Necessary? .....301  
     select\_by\_index .....302  
 Other Uses for Blocks.....303  
 Summary .....304  
     Exercises .....306  
     Answers .....306

**Week 3 Making It Work for You 311**

**Day 15 Toward Habitable Interfaces 313**

Interface Size and Intuitiveness .....314  
 An IntegerMatrix Class.....315  
     Initialization.....315  
     Storing and Retrieving Elements.....316  
     Accessing Dimensions.....318  
     Viewing a Matrix As a Whole .....319  
     Providing Iterators .....321  
     Doing the Math .....322  
     Multipurpose Methods.....324  
     A Versatile Constructor Using Default Values .....326  
     A More Versatile Constructor .....327  
 Summary .....331  
     Exercises .....332  
     Answers .....333

**Day 16 Putting It Together (Part I) 337**

The Unjumbler .....337  
     Getting the Permutations .....338  
     Finding the Needles in the Haystack.....341  
     Finished Product .....343  
     A Portable is\_word? Function.....344  
     Bailing Out Early.....345  
     Resorting to Wizardry .....346  
 Notes on Language Enhancement .....347  
 An Interactive Process Killer .....348  
     A Little About the Unix Tools.....348  
     Designing from the Top Down .....350  
 Summary .....331

<b>Day 17 Ruby/Tk</b>	<b>355</b>
What Is Tk?.....	357
Our First Tk Application.....	357
Geometry Managers .....	360
Entry Widgets and Buttons .....	362
Some Other Widgets .....	366
More Complex Coding .....	375
Summary .....	376
Exercises .....	377
Answers .....	378
<b>Day 18 Ruby/Gtk</b>	<b>389</b>
Installation under UNIX .....	390
Installation under Windows .....	391
First Ruby/Gtk Scripts .....	392
Simple Widget Layout .....	394
Bin Containers .....	394
Box Containers .....	395
Table Containers .....	399
Modular Design for Multiple Windows .....	400
Window Subclasses and Test Code .....	400
A Table Window .....	400
An Application Split into Two Files .....	402
Modal Dialogs .....	403
The Messagebox Class .....	403
More Widget Types.....	406
Sample 1: Checkbutton and Radiobutton.....	406
Sample 2: AccelGroup, ToggleButton, and HSeparator .....	408
Sample 3: Text .....	409
Sample 4: ScrollableWindow .....	411
A Full Ruby/Gtk Application .....	411
Summary .....	419
Exercises .....	420
Answers .....	421
<b>Day 19 Some Advanced Topics (That Aren't So Hard)</b>	<b>427</b>
Sockets .....	428
"Hello, World" Using TCP .....	429
"Hello, World" Using UDP .....	430
A TCP Chat Session .....	432
A Simple Web Server .....	433
Supporting Concurrent Sessions .....	438

Threads .....	440
Passing Control Around.....	442
Establishing a Pecking Order .....	444
The Bathroom Pass .....	445
When to Thread and When to Fork.....	446
Summary .....	446
Exercises .....	447
Answers .....	447
<b>Day 20 Working with the Web</b> .....	<b>449</b>
Static Content Versus Dynamic Content.....	450
Server-Side Versus Client-Side .....	450
Privileges at the Server .....	451
Configuring Apache: httpd.conf .....	452
Choose a Root Directory for Web Documents .....	452
Enable SHTML .....	453
Enable CGI .....	453
Set a User ID .....	454
Activate the New Configuration .....	454
Test Drives .....	454
Using Ruby to Generate an HTML File Directly .....	455
A First CGI Script .....	456
A First SHTML Script.....	457
Know Thy Client .....	458
Remember the Past .....	460
Use DBM to Remember Old Session Information .....	461
Object-Oriented CGI Support.....	463
Session Information Using the CGI Class .....	465
Persistent Session Information .....	466
Embedded Ruby .....	468
Summary .....	470
Exercises .....	470
Answers .....	471
<b>Day 21 Putting It Together (Part II)</b> .....	<b>475</b>
Binary Decision Trees.....	476
Script Overview .....	478
The BDT Class and Its Relatives.....	478
The BDT-sample Class .....	480
A Mild Dose of Information Theory.....	481
The BDT-set Class .....	482
The Full Script.....	485
A Test of Intelligence .....	489

Ideas for Improvements and Enhancements .....	490
Tk-based Peer Chat.....	491
What Do We Mean by “Peer”? .....	492
Top-Level Scripts.....	493
The TkChat class .....	494
Testing the Scripts .....	496
Ideas for Improvements and Enhancements .....	497
Summary .....	498
<b>Appendices</b>	<b>499</b>
<b>Appendix A irb Results</b>	<b>501</b>
<b>Appendix B Installation Help</b>	<b>503</b>
Unix.....	503
Step 1. Download the Source Code .....	504
Step 2. Unpack the Archive.....	504
Step 3. Prepare for Compiling.....	504
Step 4. make the Interpreter .....	505
Step 5. Test Before Installing .....	505
Step 6a. Install (If You’re the Administrator) .....	505
Step 6b. Update PATH (If You’re Not the Administrator) .....	506
Step 7. Test Accessibility.....	506
Step 8. Set Up the emacs Ruby Mode (Optional).....	506
Microsoft Windows.....	507
The “One-Click” Installer .....	507
Unix Wannabe Installation for Windows .....	507
<b>Appendix C Debugging, With and Without a Debugger</b>	<b>509</b>
Stack Traces .....	509
Inline Diagnostics .....	511
The Built-In Debugger .....	512
<b>Appendix D Essential Vocabulary</b>	<b>517</b>
<b>Index</b>	<b>521</b>

---

# About the Lead Author

## Mark Slagell. . . .

- . . . works as a system administrator and developer, using Ruby as a common language tool to solve problems in a heterogeneous Unix/Windows environment;
- . . . has taught C++ programming and developed original course materials at the university level;
- . . . holds a Master's degree in computer science and has participated in information security research involving the development of mobile agent tools in Java for distributed intrusion detection;
- . . . got his start hand-assembling machine code for the 6502 and Z80 processors back in the dark, misty days of microcomputing (and predictably, has been fascinated ever since with the whole idea of computer *languages*);
- . . . enjoys small-town life in Iowa and feels that in a world characterized by high-volume entertainment and fast-moving technology, nothing is more enjoyable than—or should ever take precedence over—a good game of softball.

# About the Contributing Authors

## David A. Black, Ph.D.. . .

- . . . is an Associate Professor of Communication at Seton Hall University;
- . . . has written for *Linux Journal*, as well as for numerous critical journals in film and media studies;
- . . . is the author of the book *Law in Film*;
- . . . was one of the organizers of Ruby Conference 2001;
- . . . is active on the ruby-talk mailing list and #ruby-lang IRC channel.

## Hal Fulton. . .

- . . . has two degrees in computer science;
- . . . has taught at the postsecondary level and has more than a decade of industry experience as a programmer;
- . . . is a member of the ACM and the IEEE Computer Society;
- . . . is also the author of *The Ruby Way* (Sams Publishing, 2001).



---

# Dedication

*To Brian Kernighan and Dennis Ritchie, whose little 1978 book  
The C Programming Language stands today as a model of computer language instruction.*

# Acknowledgments

Where can I begin?

Where else but with Yukihiro “Matz” Matsumoto, creator of the breakthrough language you’re about to get hooked on.

Warm thanks go to David Alan Black and Hal Fulton for their substantial contributions to this volume, and to Dave Thomas for his careful reading, insights, and just-tactful-enough suggestions. I’d also like to thank Neil Conway for some ideas that helped shape the Ruby/Gtk chapter.

I have Akinori Musha to thank for bringing my English retranslation of Matz’s original Ruby tutorial into the mainstream, which seems to be what got me involved in the Ruby community in the first place, and Curt Clifton and Gary Leavens for their early encouragement and interest, which helped me decide to forge ahead with this project.

My friends Susan Yager and David Heddendorf helped me think about the book in a literary sense and suggested one of the better quotations that was used to open a chapter.

To my wife Amy for the patience I have not yet managed to exhaust, to older son Carter for taking on some extra tasks to help free up my time, and to younger son Kenny for asking, as only a five-year-old could, how my “story” was coming: Thank you all.

---

# Tell Us What You Think!

As the reader of this book, you are our most important critic and commentator. We value your opinion and want to know what we're doing right, what we could do better, what areas you'd like to see us publish in, and any other words of wisdom you're willing to pass our way.

As an associate publisher for Sams Publishing, I welcome your comments. You can fax, e-mail, or write me directly to let me know what you did or didn't like about this book—as well as what we can do to make our books stronger.



Please note that I cannot help you with technical problems related to the topic of this book, and that due to the high volume of mail I receive, I might not be able to reply to every message.

When you write, please be sure to include this book's title and author name as well as your name and phone or fax number. I will carefully review your comments and share them with the author and editors who worked on the book.

Fax: 317-581-4770

E-mail: [consumer@sampublishing.com](mailto:consumer@sampublishing.com)

Mail: Mark Taber

Sams Publishing  
201 West 103rd Street  
Indianapolis, IN 46290 USA



---

# Introduction

The book you hold in your hands is unusual.

You know that it is a book about computer programming, and it is about a language that you may not have heard a lot about; but that much is not, perhaps, unusual. There are many computer languages in existence, you are probably unfamiliar with most of them (isn't everybody?), and you've seen bookstore shelves overflowing with programming titles.

What is unusual is that Ruby is a very advanced programming language, yet this book claims to teach it to you while assuming almost nothing about your expertise or experience. If those two statements sound unrealistic and incompatible, don't immediately put the book down and give up. Instead, consider what it means for a programming language to *be* advanced. You might be inclined to say that an advanced language would be powerful, cryptic, and hard to learn; and would be the province of the highly experienced professional who has been doing it for years, already knows half a dozen languages inside and out, and has found them all wanting in some way.

And you'd be partly right. An advanced language would be powerful. It would also attract highly experienced professionals who found that other languages didn't do what they wanted in quite the way they wanted. But think again about the rest: A truly advanced language might not necessarily be cryptic, because being cryptic cannot be considered a virtue. Nor, by the same token, would it necessarily be hard to learn. And so its usefulness might not be limited to the most experienced programmers.

The world of programming languages has been undergoing sweeping changes in recent years, driven indirectly by advances in hardware. In the good old days, memory was scarce and expensive, and a computer didn't compute all that fast, although it always *felt* fast at the time, as I recall. The lack of cheap speed and cheap storage meant that to get a computer to do anything useful, a programmer had to work at a fairly low level of abstraction. We made our living close to the operating system and, in some cases, the hardware. We had to optimize and tweak on things to get them to run at an acceptable speed (though good compilers and other tools helped with this). Whenever we had collections of information, we thought carefully about how the items were organized in memory. Storage had to be explicitly allocated and freed, and it had to be done just right or things would go to pieces in confusing and seemingly random ways. It was tricky, and it was hard work.

Of course there's nothing wrong with hard work. But there's a good reason to get away from that particular sort of hard work as often as possible. When you write a computer

program, you are doing it to solve some particular problem, and that problem usually has nothing to do with the computer, the hardware in front of you, the operating system you're using, the language and compiler you use, and so on. The problem that you want to work on is, in and of itself, interesting. That problem is something you can probably clearly state in words (if you can't, you're in trouble from the start). And therein lies the need for advanced computer languages: When you are thinking about the *problem you wanted to solve in the first place*, you are working mentally at a different and higher level of abstraction than when you've got your hands all over the operating system. It's unnaturally hard to work at more than one level of abstraction at the same time. Not many of us are good at it. So we need high-level languages (what some people are calling "very high-level languages" or *VHLLs*) to help us concentrate on the problem at hand.

## What Is Assumed About You

The idea of VHLLs is not new; the need has been present for a long time, and the term has been applied to a number of languages in the recent past. But Ruby is the first language, in my view, to achieve this combination of power, consistency, and readability. Interested people who may not have a programming background can reasonably hope to grasp it. Ruby has made a decisive crossing of the Geek Event Horizon and is the first serious language (as opposed to BASIC and similar toys) to have done so.

Accordingly, this book does not assume that you have any particular experience in programming or computer science. It assumes only that you have intelligence, curiosity, and some time; given those (and perhaps a healthy sense of humor), you can expect to pick up the new skills you are looking for.

On the other hand, neither is it assumed that you are a beginner. If you are a seasoned C, C++, Java, or Perl programmer, Ruby represents a paradigm shift, an opportunity to take a step back and reexamine how you have approached problems. You may find that there are some things you need to unlearn, but your past experience will also certainly prove useful at many points.

## What's This About Levels of Abstraction?

Consider a task described as follows: Read a file named `raw`, extract all the lines containing the word `Brighton`, sort those lines in increasing numerical order by their third whitespace-separated field, and write the results to a new file named `processed`. This example will help us understand the motivation for VHLLs in more concrete terms.

Someone trying to perform that task in C would typically think at this level (at least, if trying to solve the problem in haste):

*Declare two file handles. Reserve a small number of bytes as a character buffer. Is 80 enough? Yeah, that's probably reasonable. I can always increase it later if I find out differently. And I'll probably want a really big buffer to hold the whole file. But I don't know ahead of time how big the file will be . . . How big is big enough? It's safest to do that after opening the file. Take a deep breath and do it right. I'll ask the operating system for a buffer. So let's see, open the "raw" file in read mode. Do a seek to move the file pointer to the end of the file and a tell to find out what the absolute position is, meaning that's the total length of the file. Remember that number. Rewind the file pointer back to the beginning. Ask the operating system for enough memory to hold the file. Check the return value in case there's a problem. Now I've got a pointer at the beginning of the big buffer, and I'll make a copy of that pointer. The original will stay put and the copy will move. While not yet at the input end-of-file, read a line into the smaller buffer and use a library function to search for a "Brighton" in the buffer. If the function returns something nonzero (or was it zero? Doesn't zero mean success? Wait, better look that up, I can never remember), it means the string was found in the buffer, so write the buffer to where the roving pointer points, then move the pointer forward the size of the input line. When at to the end of the input file, close it. Now I've got a buffer containing just the "Brighton" lines. So far, so good. But how to sort it? It will be hard to sort in place because the lines aren't all the same length. Maybe I should have made an array of pointers at the beginning of each line, then sorted the pointers . . . wait, this will work, I'll keep going through the buffer, picking out the smallest line I find each time, writing that line to the output file and then kind of blanking it out. Ready to proceed, so open the output file with name "processed". Make a flag variable, set it to false, I mean 0. Set up a do-while loop using the flag. Copy the stationary buffer pointer to the roving one again. In the loop, move the roving pointer forward through the line until it sees whitespace twice. I should make an inner loop for that. Now use a library function to convert the text where the pointer is pointing into an integer. Wait, no, a float. Compare that to the lowest number we've seen so far. Hmm, I don't know what the lowest number seen so far is. Better declare a lowest variable, then initialize it to something impossibly high. Do that before the outer loop starts. If it's lower than the lowest, copy that value into lowest, remember where the start of the line was (Wait! I moved the pointer and don't know where the line started anymore. No problem, make a third pointer, copy the second pointer to it every time before you go looking for whitespace), and set the flag to true, I mean 1. At the end of the buffer (how will I know I'm at the end? Better back up and set up a fourth pointer after reading the input file, so we can compare position to that pointer), if the flag is 1, write the line that had lowest value to the output file, then mark the line so it won't be considered again. If the flag is 0, then we must have written all the lines, so we close the output file and we're done.*

Phew. Programming is hard, eh? And that's leaving out quite a few details and resorting to an inefficient and simple-minded sorting method. We also haven't accounted for the debugging phase, which can be counted on to be necessary because something is almost sure to go wrong somewhere. But eventually we'll get it to work—at least until we come across an input file that violates our assumptions, or until someone changes the requirements a little and we find that what we wrote won't easily adapt to the altered task and so we have to go back, Jack, and do it again. We spend our time putzing around with low-level programming details, struggling to come up with some rickety contraption to do the required job, instead of crafting a sensible, adaptable design.

I don't bring this up to disparage the C language. C programs tend to perform with good speed; there are tasks for which it is very well suited; it's possible to write in C and think much more clearly than the above example indicates. There are people who are very good at it. Either they have become so adept at the mental dance involved that they can do it efficiently—even unconsciously—or they have learned to avoid as much low-level thinking as possible and to make prudent use of their own and other people's code libraries, which is a first step toward true high-level programming.

Now let's head for more hospitable latitudes. Suppose you want to perform the same task in Ruby. Here's the likely thought process involved:

*Make an array of the lines from the input file "raw". Get rid of the lines that don't contain "Brighton". Sort the array in increasing numerical order by the third whitespace-separated field. Open an output file, "processed". Print each line from the array to the file. Close the file.*

Well, that sounds suspiciously like the original problem description, doesn't it? It also suggests a short program. Observe:

```
the_lines = File.readlines("raw")
the_lines.delete_if { |l| l !~ "Brighton" }
the_lines.sort! { |x,y| (x.split)[2].to_f <=> (y.split)[2].to_f }
out_file = File.new("processed","w")
the_lines.each { |l| out_file.print l }
out_file.close
```

What just happened? We concentrated on what the job was about, and the high-level task description translated cleanly into a working program. No pointers, no memory allocation, no loops.<sup>1</sup> Thinking about the problem at hand, we shall see, is the Ruby Way.

Does it all sound too easy? Don't worry. There will be plenty of challenges ahead.

---

<sup>1</sup>Not that we couldn't write "loopy" code in Ruby too. We can. But it isn't usually necessary.

---

But when learning Ruby you'll find life to be pretty pleasant. While it doesn't solve all the big problems for you, it gives you a nice logical framework and a set of tools to handle the most tedious stuff so that you can keep working at a high level of abstraction, completing your jobs quickly, reliably, and maybe even happily.

## "The Ruby Way"

We'll keep coming back to discussion about the Ruby Way, which is a phrase often thrown around in the online Ruby developer and user community. As presumptuous as it may be to suggest that there is only one Ruby Way—after all, my view of it, or way of expressing it, may be much different from another's—there is at least some shared intuition about what it is.<sup>2</sup>

The Ruby Way is to avoid sweating the details when it's unnecessary.

The Ruby Way is to solve most problems exactly once.

The Ruby Way is to use your task specifications, whatever they are, to guide the writing of your programs.

The Ruby Way is to test as you go.

Above all:

The Ruby Way is to *use the language as a language* in the conventional, or literary, sense; something that exists to help you express yourself, not an obscure technical maze that keeps getting in your way, begging to be worked around and outsmarted.

Enjoy yourself. It's what we are all here for.

## Organization of the Book

This book is organized into 21 lessons, and in keeping with the title, they are numbered by days. While it is undeniably true that everyone learns in different ways and at different speeds, it is suggested that something close to the chapter-per-day pace be observed to ensure that each lesson has a chance to sink in before the next is devoured.

At the end of most lessons there are a few exercises. They are intended to give you opportunities to test and hone your new skills, and also to goad you into contemplating some questions that will motivate you to the lessons yet to come.

---

<sup>2</sup>The Ruby Way is also the title of a pretty good Ruby book you may want to pick up when you're done with this one.



---

sample content of Sams Teach Yourself Ruby in 21 Days

- [read online Naamah's Blessing \(Kushiel's Universe, Book 9; Kushiel's Legacy, Book 9; Moirin's Trilogy, Book 3\) pdf, azw \(kindle\), epub](#)
- [read The Compassionate Mind: A New Approach to Life's Challenges](#)
- [read online Take Control of LaunchBar book](#)
- [Kings of Ruin pdf, azw \(kindle\), epub, doc, mobi](#)
- [The Origin of Concurrent Programming: From Semaphores to Remote Procedure Calls pdf, azw \(kindle\), epub, doc, mobi](#)
- [\*In Transit here\*](#)
  
- <http://pittiger.com/lib/Naamah-s-Blessing--Kushiel-s-Universe--Book-9--Kushiel-s-Legacy--Book-9--Moirin-s-Trilogy--Book-3-.pdf>
- <http://nexson.arzamaszev.com/library/The-Compassionate-Mind--A-New-Approach-to-Life-s-Challenges.pdf>
- <http://aseasonedman.com/ebooks/Take-Control-of-LaunchBar.pdf>
- <http://korplast.gr/lib/Are-You-There-God--It-s-Me--Margaret-.pdf>
- <http://xn--d1aboelcb1f.xn--p1ai/lib/The-Origin-of-Concurrent-Programming--From-Semaphores-to-Remote-Procedure-Calls.pdf>
- <http://musor.ruspb.info/?library/The-Norman-Maclean-Reader.pdf>