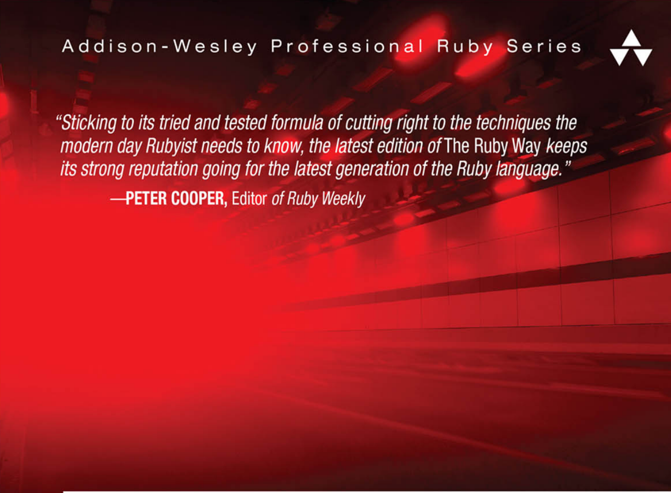
Addison-Wesley Professional Ruby Series



"Sticking to its tried and tested formula of cutting right to the techniques the modern day Rubyist needs to know, the latest edition of The Ruby Way *keeps its strong reputation going for the latest generation of the Ruby language."*

—**PETER COOPER,** Editor *of Ruby Weekly*

THE RUBY WAY

THIRD EDITION

HAL FULTON

with **ANDRÉ ARKO**

FREE SAMPLE CHAPTER

SHARE WITH OTHERS

Praise for *The Ruby Way,* Third Edition

"Sticking to its tried and tested formula of cutting right to the techniques the modern day Rubyist needs to know, the latest edition of *The Ruby Way* keeps its strong reputation going for the latest generation of the Ruby language."

Peter Cooper
Editor of *Ruby Weekly*

"The authors' excellent work and meticulous attention to detail continues in this latest update; this book remains an outstanding reference for the beginning Ruby programmer—as well as the seasoned developer who needs a quick refresh on Ruby. Highly recommended for anyone interested in Ruby programming."

Kelvin Meeks
Enterprise Architect

Praise for Previous Editions of *The Ruby Way*

"Among other things, this book excels at explaining metaprogramming, one of the most interesting aspects of Ruby. Many of the early ideas for Rails were inspired by the first edition, especially what is now Chapter 11. It puts you on a rollercoaster ride between 'How could I use this?' and 'This is so cool!' Once you get on that rollercoaster, there's no turning back."

David Heinemeier Hansson
Creator of Ruby on Rails,
Founder at Basecamp

"The appearance of the second edition of this classic book is an exciting event for Rubyists—and for lovers of superb technical writing in general. Hal Fulton brings a lively erudition and an engaging, lucid style to bear on a thorough and meticulously exact exposition of Ruby. You palpably feel the presence of a teacher who knows a tremendous amount and really wants to help you know it too."

David Alan Black
Author of *The Well-Grounded Rubyist*

“This is an excellent resource for gaining insight into how and why Ruby works. As someone who has worked with Ruby for several years, I still found it full of new tricks and techniques. It’s accessible both as a straight read and as a reference that one can dip into and learn something new.”

Chet Hendrickson
Agile software pioneer

“Ruby’s a wonderful language—but sometimes you just want to get something done. Hal’s book gives you the solution and teaches a good bit about why that solution is good Ruby.”

Martin Fowler
Chief Scientist, ThoughtWorks
Author of *Patterns of Enterprise*
Application Architecture

THE RUBY WAY

Third Edition

Hal Fulton

with André Arko

Addison-Wesley

Upper Saddle River, NJ • Boston • Indianapolis • San Francisco
New York • Toronto • Montreal • London • Munich • Paris • Madrid
Cape Town • 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 authors 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.

For information about buying this title in bulk quantities, or for special sales opportunities (which may include electronic versions; custom cover designs; and content particular to your business, training goals, marketing focus, or branding interests), please contact our corporate sales department at corpsales@pearsoned.com or (800) 382-3419.

For government sales inquiries, please contact governmentsales@pearsoned.com.

For questions about sales outside the U.S., please contact international@pearsoned.com.

Visit us on the Web: informit.com/aw

Library of Congress Control Number: 2014945504

Copyright © 2015 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. To obtain permission to use material from this work, please submit a written request to Pearson Education, Inc., Permissions Department, One Lake Street, Upper Saddle River, New Jersey 07458, or you may fax your request to (201) 236-3290.

ISBN-13: 978-0-321-71463-3
ISBN-10: 0-321-71463-6

Text printed in the United States on recycled paper at RR Donnelley in Crawfordsville, Indiana
First printing: March 2015

Editor-in-Chief
Mark Taub

Executive Editor
Debra Williams-Cauley

Development Editor
Songlin Qiu

Managing Editor
Kristy Hart

Project Editor
Andy Beaster

Copy Editor
Bart Reed

Indexer
Ken Johnson

Proofreader
Sarah Kearns

Cover Designer
Chuti Prasertsith

Senior Compositor
Gloria Schurick

To my parents, without whom I would not be possible
—Hal

This page intentionally left blank

Contents

Foreword xxiv

Acknowledgments xxviii

About the Authors xxxii

Introduction xxxiii

1 Ruby in Review 1

1.1 An Introduction to Object Orientation 2

1.1.1 What Is an Object? 2

1.1.2 Inheritance 4

1.1.3 Polymorphism 6

1.1.4 A Few More Terms 7

1.2 Basic Ruby Syntax and Semantics 8

1.2.1 Keywords and Identifiers 9

1.2.2 Comments and Embedded Documentation 10

1.2.3 Constants, Variables, and Types 11

1.2.4 Operators and Precedence 13

1.2.5 A Sample Program 14

1.2.6 Looping and Branching 17

1.2.7 Exceptions 22

1.3 OOP in Ruby 25

1.3.1 Objects 26

1.3.2 Built-in Classes 26

1.3.3 Modules and Mixins 28

1.3.4 Creating Classes 29

1.3.5 Methods and Attributes 34

1.4 Dynamic Aspects of Ruby 36
1.4.1 Coding at Runtime 36
1.4.2 Reflection 38
1.4.3 Missing Methods 40
1.4.4 Garbage Collection 40
1.5 Training Your Intuition: Things to Remember 41
1.5.1 Syntax Issues 41
1.5.2 Perspectives in Programming 44
1.5.3 Ruby's `case` Statement 47
1.5.4 Rubyisms and Idioms 50
1.5.5 Expression Orientation and Other Miscellaneous Issues 57
1.6 Ruby Jargon and Slang 59
1.7 Conclusion 62

2 Working with Strings 63
2.1 Representing Ordinary Strings 64
2.2 Representing Strings with Alternate Notations 65
2.3 Using Here-Documents 65
2.4 Finding the Length of a String 67
2.5 Processing a Line at a Time 68
2.6 Processing a Character or Byte at a Time 68
2.7 Performing Specialized String Comparisons 69
2.8 Tokenizing a String 71
2.9 Formatting a String 73
2.10 Using Strings as IO Objects 74
2.11 Controlling Uppercase and Lowercase 74
2.12 Accessing and Assigning Substrings 75
2.13 Substituting in Strings 78
2.14 Searching a String 79
2.15 Converting Between Characters and ASCII Codes 80
2.16 Implicit and Explicit Conversion 80
2.17 Appending an Item onto a String 83
2.18 Removing Trailing Newlines and Other Characters 83
2.19 Trimming Whitespace from a String 84
2.20 Repeating Strings 85
2.21 Embedding Expressions within Strings 85

2.22 Delayed Interpolation of Strings 86
2.23 Parsing Comma-Separated Data 86
2.24 Converting Strings to Numbers (Decimal and Otherwise) 87
2.25 Encoding and Decoding `rot13` Text 89
2.26 Encrypting Strings 90
2.27 Compressing Strings 91
2.28 Counting Characters in Strings 92
2.29 Reversing a String 92
2.30 Removing Duplicate Characters 93
2.31 Removing Specific Characters 93
2.32 Printing Special Characters 93
2.33 Generating Successive Strings 94
2.34 Calculating a 32-Bit CRC 94
2.35 Calculating the SHA-256 Hash of a String 95
2.36 Calculating the Levenshtein Distance Between Two Strings 96
2.37 Encoding and Decoding Base64 Strings 98
2.38 Expanding and Compressing Tab Characters 98
2.39 Wrapping Lines of Text 99
2.40 Conclusion 100

3 Working with Regular Expressions 101

3.1 Regular Expression Syntax 102
3.2 Compiling Regular Expressions 104
3.3 Escaping Special Characters 105
3.4 Using Anchors 105
3.5 Using Quantifiers 106
3.6 Positive and Negative Lookahead 109
3.7 Positive and Negative Lookbehind 110
3.8 Accessing Backreferences 111
3.9 Named Matches 114
3.10 Using Character Classes 116
3.11 Extended Regular Expressions 118
3.12 Matching a Newline with a Dot 119
3.13 Using Embedded Options 119
3.14 Using Embedded Subexpressions 120
3.14.1 Recursion in Regular Expressions 121

3.15 A Few Sample Regular Expressions 122
3.15.1 Matching an IP Address 122
3.15.2 Matching a Keyword-Value Pair 123
3.15.3 Matching Roman Numerals 124
3.15.4 Matching Numeric Constants 125
3.15.5 Matching a Date/Time String 125
3.15.6 Detecting Doubled Words in Text 126
3.15.7 Matching All-Caps Words 127
3.15.8 Matching Version Numbers 127
3.15.9 A Few Other Patterns 127
3.16 Conclusion 128

4 Internationalization in Ruby 129
4.1 Background and Terminology 131
4.2 Working with Character Encodings 135
4.2.1 Normalization 136
4.2.2 Encoding Conversions 139
4.2.3 Transliteration 141
4.2.4 Collation 141
4.3 Translations 144
4.3.1 Defaults 146
4.3.2 Namespaces 147
4.3.3 Interpolation 148
4.3.4 Pluralization 149
4.4 Localized Formatting 151
4.4.1 Dates and Times 151
4.4.2 Numbers 152
4.4.3 Currencies 153
4.5 Conclusion 153

5 Performing Numerical Calculations 155
5.1 Representing Numbers in Ruby 156
5.2 Basic Operations on Numbers 157
5.3 Rounding Floating Point Values 158
5.4 Comparing Floating Point Numbers 160
5.5 Formatting Numbers for Output 162

5.6 Formatting Numbers with Commas 162
5.7 Working with Very Large Integers 163
5.8 Using `BigDecimal` 163
5.9 Working with Rational Values 166
5.10 Matrix Manipulation 167
5.11 Working with Complex Numbers 171
5.12 Using `mathn` 172
5.13 Finding Prime Factorization, GCD, and LCM 173
5.14 Working with Prime Numbers 174
5.15 Implicit and Explicit Numeric Conversion 175
5.16 Coercing Numeric Values 176
5.17 Performing Bit-Level Operations on Numbers 177
5.18 Performing Base Conversions 179
5.19 Finding Cube Roots, Fourth Roots, and So On 180
5.20 Determining the Architecture's Byte Order 181
5.21 Numerical Computation of a Definite Integral 182
5.22 Trigonometry in Degrees, Radians, and Grads 183
5.23 Finding Logarithms with Arbitrary Bases 184
5.24 Finding the Mean, Median, and Mode of a Data Set 185
5.25 Variance and Standard Deviation 187
5.26 Finding a Correlation Coefficient 187
5.27 Generating Random Numbers 189
5.28 Caching Functions with Memoization 190
5.29 Conclusion 191

6 Symbols and Ranges 193

6.1 Symbols 193
6.1.1 Symbols as Enumerations 195
6.1.2 Symbols as Metavalues 196
6.1.3 Symbols, Variables, and Methods 197
6.1.4 Converting to/from Symbols 197
6.2 Ranges 199
6.2.1 Open and Closed Ranges 199
6.2.2 Finding Endpoints 200
6.2.3 Iterating Over Ranges 200
6.2.4 Testing Range Membership 201
6.2.5 Converting to Arrays 202

6.2.6 Backward Ranges 202
6.2.7 The Flip-Flop Operator 203
6.2.8 Custom Ranges 206
6.3 Conclusion 209

7 Working with Times and Dates 211
7.1 Determining the Current Time 212
7.2 Working with Specific Times (Post-Epoch) 212
7.3 Determining the Day of the Week 214
7.4 Determining the Date of Easter 215
7.5 Finding the *N*th Weekday in a Month 215
7.6 Converting Between Seconds and Larger Units 217
7.7 Converting to and from the Epoch 217
7.8 Working with Leap Seconds: Don't! 218
7.9 Finding the Day of the Year 219
7.10 Validating a Date or Time 219
7.11 Finding the Week of the Year 220
7.12 Detecting Leap Years 221
7.13 Obtaining the Time Zone 222
7.14 Working with Hours and Minutes Only 222
7.15 Comparing Time Values 223
7.16 Adding Intervals to Time Values 223
7.17 Computing the Difference in Two Time Values 224
7.18 Working with Specific Dates (Pre-Epoch) 224
7.19 Time, Date, and DateTime 225
7.20 Parsing a Date or Time String 225
7.21 Formatting and Printing Time Values 226
7.22 Time Zone Conversions 227
7.23 Determining the Number of Days in a Month 228
7.24 Dividing a Month into Weeks 229
7.25 Conclusion 230

8 Arrays, Hashes, and Other Enumerables 231
8.1 Working with Arrays 232
8.1.1 Creating and Initializing an Array 232
8.1.2 Accessing and Assigning Array Elements 233
8.1.3 Finding an Array's Size 235

8.1.4 Comparing Arrays 235
8.1.5 Sorting an Array 237
8.1.6 Selecting from an Array by Criteria 240
8.1.7 Using Specialized Indexing Functions 242
8.1.8 Implementing a Sparse Matrix 244
8.1.9 Using Arrays as Mathematical Sets 244
8.1.10 Randomizing an Array 248
8.1.11 Using Multidimensional Arrays 249
8.1.12 Finding Elements in One Array But Not Another 250
8.1.13 Transforming or Mapping Arrays 250
8.1.14 Removing `nil` Values from an Array 251
8.1.15 Removing Specific Array Elements 251
8.1.16 Concatenating and Appending onto Arrays 253
8.1.17 Using an Array as a Stack or Queue 254
8.1.18 Iterating over an Array 254
8.1.19 Interposing Delimiters to Form a String 255
8.1.20 Reversing an Array 256
8.1.21 Removing Duplicate Elements from an Array 256
8.1.22 Interleaving Arrays 256
8.1.23 Counting Frequency of Values in an Array 257
8.1.24 Inverting an Array to Form a Hash 257
8.1.25 Synchronized Sorting of Multiple Arrays 258
8.1.26 Establishing a Default Value for New Array Elements 259
8.2 Working with Hashes 260
8.2.1 Creating a New Hash 260
8.2.2 Specifying a Default Value for a Hash 261
8.2.3 Accessing and Adding Key-Value Pairs 262
8.2.4 Deleting Key-Value Pairs 264
8.2.5 Iterating Over a Hash 264
8.2.6 Inverting a Hash 265
8.2.7 Detecting Keys and Values in a Hash 265
8.2.8 Extracting Hashes into Arrays 266
8.2.9 Selecting Key-Value Pairs by Criteria 266
8.2.10 Sorting a Hash 267
8.2.11 Merging Two Hashes 268
8.2.12 Creating a Hash from an Array 268

8.2.13 Finding Difference or Intersection of Hash Keys 268
8.2.14 Using a Hash as a Sparse Matrix 269
8.2.15 Implementing a Hash with Duplicate Keys 270
8.2.16 Other Hash Operations 273
8.3 Enumerables in General 273
8.3.1 The `inject` Method 274
8.3.2 Using Quantifiers 275
8.3.3 The `partition` Method 276
8.3.4 Iterating by Groups 277
8.3.5 Converting to Arrays or Sets 278
8.3.6 Using Enumerator Objects 278
8.4 More on Enumerables 280
8.4.1 Searching and Selecting 280
8.4.2 Counting and Comparing 281
8.4.3 Iterating 282
8.4.4 Extracting and Converting 283
8.4.5 Lazy Enumerators 284
8.5 Conclusion 285

9 More Advanced Data Structures 287
9.1 Working with Sets 288
9.1.1 Simple Set Operations 288
9.1.2 More Advanced Set Operations 290
9.2 Working with Stacks and Queues 291
9.2.1 Implementing a Stricter Stack 293
9.2.2 Detecting Unbalanced Punctuation in Expressions 294
9.2.3 Understanding Stacks and Recursion 295
9.2.4 Implementing a Stricter Queue 297
9.3 Working with Trees 298
9.3.1 Implementing a Binary Tree 298
9.3.2 Sorting Using a Binary Tree 300
9.3.3 Using a Binary Tree as a Lookup Table 302
9.3.4 Converting a Tree to a String or Array 303
9.4 Working with Graphs 304
9.4.1 Implementing a Graph as an Adjacency Matrix 304
9.4.2 Determining Whether a Graph Is Fully Connected 307

9.4.3 Determining Whether a Graph Has an Euler Circuit 308
9.4.4 Determining Whether a Graph Has an Euler Path 309
9.4.5 Graph Tools in Ruby 310
9.5 Conclusion 310

10 I/O and Data Storage 311
10.1 Working with Files and Directories 313
10.1.1 Opening and Closing Files 313
10.1.2 Updating a File 314
10.1.3 Appending to a File 315
10.1.4 Random Access to Files 315
10.1.5 Working with Binary Files 316
10.1.6 Locking Files 318
10.1.7 Performing Simple I/O 318
10.1.8 Performing Buffered and Unbuffered I/O 320
10.1.9 Manipulating File Ownership and Permissions 321
10.1.10 Retrieving and Setting Timestamp Information 323
10.1.11 Checking File Existence and Size 325
10.1.12 Checking Special File Characteristics 326
10.1.13 Working with Pipes 328
10.1.14 Performing Special I/O Operations 329
10.1.15 Using Nonblocking I/O 330
10.1.16 Using `readpartial` 331
10.1.17 Manipulating Pathnames 331
10.1.18 Using the `Pathname` Class 333
10.1.19 Command-Level File Manipulation 334
10.1.20 Grabbing Characters from the Keyboard 336
10.1.21 Reading an Entire File into Memory 336
10.1.22 Iterating Over a File by Lines 337
10.1.23 Iterating Over a File by Byte or Character 337
10.1.24 Treating a String As a File 338
10.1.25 Copying a Stream 339
10.1.26 Working with Character Encodings 339
10.1.27 Reading Data Embedded in a Program 339
10.1.28 Reading Program Source 340
10.1.29 Working with Temporary Files 340

10.1.30 Changing and Setting the Current Directory 341

10.1.31 Changing the Current Root 342

10.1.32 Iterating Over Directory Entries 342

10.1.33 Getting a List of Directory Entries 342

10.1.34 Creating a Chain of Directories 342

10.1.35 Deleting a Directory Recursively 343

10.1.36 Finding Files and Directories 343

10.2 Higher-Level Data Access 344

10.2.1 Simple Marshaling 345

10.2.2 "Deep Copying" with `Marshal` 346

10.2.3 More Complex Marshaling 346

10.2.4 Marshaling with `YAML` 347

10.2.5 Persisting Data with `JSON` 349

10.2.6 Working with CSV Data 350

10.2.7 SQLite3 for SQL Data Storage 352

10.3 Connecting to External Data Stores 353

10.3.1 Connecting to MySQL Databases 354

10.3.2 Connecting to PostgreSQL Databases 356

10.3.3 Object-Relational Mappers (ORMs) 358

10.3.4 Connecting to Redis Data Stores 359

10.4 Conclusion 360

11 OOP and Dynamic Features in Ruby 361

11.1 Everyday OOP Tasks 362

11.1.1 Using Multiple Constructors 362

11.1.2 Creating Instance Attributes 364

11.1.3 Using More Elaborate Constructors 366

11.1.4 Creating Class-Level Attributes and Methods 368

11.1.5 Inheriting from a Superclass 372

11.1.6 Testing Classes of Objects 374

11.1.7 Testing Equality of Objects 377

11.1.8 Controlling Access to Methods 378

11.1.9 Copying an Object 381

11.1.10 Using `initialize_copy` 383

11.1.11 Understanding `allocate` 384

11.1.12 Working with Modules 384
11.1.13 Transforming or Converting Objects 388
11.1.14 Creating Data-Only Classes (Structs) 390
11.1.15 Freezing Objects 391
11.1.16 Using `tap` in Method Chaining 393
11.2 More Advanced Techniques 394
11.2.1 Sending an Explicit Message to an Object 394
11.2.2 Specializing an Individual Object 396
11.2.3 Nesting Classes and Modules 399
11.2.4 Creating Parametric Classes 400
11.2.5 Storing Code as `Proc` Objects 403
11.2.6 Storing Code as `Method` Objects 405
11.2.7 Using Symbols as Blocks 406
11.2.8 How Module Inclusion Works 406
11.2.9 Detecting Default Parameters 409
11.2.10 Delegating or Forwarding 409
11.2.11 Defining Class-Level Readers and Writers 412
11.2.12 Working in Advanced Programming Disciplines 414
11.3 Working with Dynamic Features 416
11.3.1 Evaluating Code Dynamically 416
11.3.2 Retrieving a Constant by Name 418
11.3.3 Retrieving a Class by Name 418
11.3.4 Using `define_method` 419
11.3.5 Obtaining Lists of Defined Entities 423
11.3.6 Removing Definitions 425
11.3.7 Handling References to Nonexistent Constants 427
11.3.8 Handling Calls to Nonexistent Methods 429
11.3.9 Improved Security with `taint` 430
11.3.10 Defining Finalizers for Objects 432
11.4 Program Introspection 433
11.4.1 Traversing the Object Space 434
11.4.2 Examining the Call Stack 435
11.4.3 Tracking Changes to a Class or Object Definition 435
11.4.4 Monitoring Program Execution 439
11.5 Conclusion 441

12 Graphical Interfaces for Ruby 443

12.1 Shoes 4 444

12.1.1 Starting Out with Shoes 444

12.1.2 An Interactive Button 445

12.1.3 Text and Input 446

12.1.4 Layout 448

12.1.5 Images and Shapes 450

12.1.6 Events 450

12.1.7 Other Notes 451

12.2 Ruby/Tk 452

12.2.1 Overview 452

12.2.2 A Simple Windowed Application 453

12.2.3 Working with Buttons 455

12.2.4 Working with Text Fields 459

12.2.5 Working with Other Widgets 463

12.2.6 Other Notes 467

12.3 Ruby/GTK3 467

12.3.1 Overview 467

12.3.2 A Simple Windowed Application 468

12.3.3 Working with Buttons 469

12.3.4 Working with Text Fields 471

12.3.5 Working with Other Widgets 474

12.3.6 Other Notes 479

12.4 QtRuby 480

12.4.1 Overview 480

12.4.2 A Simple Windowed Application 480

12.4.3 Working with Buttons 481

12.4.4 Working with Text Fields 483

12.4.5 Working with Other Widgets 485

12.4.6 Other Notes 490

12.5 Swing 491

12.6 Other GUI Toolkits 493

12.6.1 UNIX and X11 493

12.6.2 FXRuby (FOX) 493

12.6.3 RubyMotion for iOS and Mac OS X 494
12.6.4 The Windows Win32API 494
12.7 Conclusion 494

13 Threads and Concurrency 495
13.1 Creating and Manipulating Threads 497
13.1.1 Creating Threads 497
13.1.2 Accessing Thread-Local Variables 498
13.1.3 Querying and Changing Thread Status 500
13.1.4 Achieving a Rendezvous (and Capturing a Return Value) 505
13.1.5 Dealing with Exceptions 506
13.1.6 Using a Thread Group 508
13.2 Synchronizing Threads 509
13.2.1 Performing Simple Synchronization 511
13.2.2 Synchronizing Access with a Mutex 512
13.2.3 Using the Built-in Queue Classes 515
13.2.4 Using Condition Variables 517
13.2.5 Other Synchronization Techniques 518
13.2.6 Setting a Timeout for an Operation 522
13.2.7 Waiting for an Event 524
13.2.8 Collection Searching in Parallel 525
13.2.9 Recursive Deletion in Parallel 526
13.3 Fibers and Cooperative Multitasking 527
13.4 Conclusion 530

14 Scripting and System Administration 531
14.1 Running External Programs 532
14.1.1 Using `system` and `exec` 532
14.1.2 Capturing Command Output 533
14.1.3 Manipulating Processes 534
14.1.4 Manipulating Standard Input and Output 537
14.2 Command-Line Options and Arguments 538
14.2.1 Working with `ARGV` 538
14.2.2 Working with `ARGF` 539
14.2.3 Parsing Command-Line Options 540

14.3 The Shell Library 542
14.3.1 Using Shell for I/O Redirection 542
14.3.2 Other Notes on Shell 544
14.4 Accessing Environment Variables 545
14.4.1 Getting and Setting Environment Variables 545
14.4.2 Storing Environment Variables as an Array or Hash 546
14.5 Working with Files, Directories, and Trees 547
14.5.1 A Few Words on Text Filters 547
14.5.2 Copying a Directory Tree 548
14.5.3 Deleting Files by Age or Other Criteria 549
14.5.4 Determining Free Space on a Disk 550
14.6 Other Scripting Tasks 551
14.6.1 Distributing Ruby Programs 551
14.6.2 Piping into the Ruby Interpreter 552
14.6.3 Testing Whether a Program Is Running Interactively 553
14.6.4 Determining the Current Platform or Operating System 554
14.6.5 Using the Etc Module 554
14.7 Conclusion 555

15 Ruby and Data Formats 557
15.1 Parsing JSON 558
15.1.1 Navigating JSON Data 559
15.1.2 Handling Non-JSON Data Types 560
15.1.3 Other JSON Libraries 560
15.2 Parsing XML (and HTML) 561
15.2.1 Document Parsing 561
15.2.2 Stream Parsing 564
15.3 Working with RSS and Atom 566
15.3.1 Parsing Feeds 567
15.3.2 Generating Feeds 568
15.4 Manipulating Image Data with RMagick 569
15.4.1 Common Graphics Tasks 570
15.4.2 Special Effects and Transformations 573
15.4.3 The Drawing API 576

15.5 Creating PDF Documents with Prawn 579
15.5.1 Basic Concepts and Techniques 579
15.5.2 An Example Document 580
15.6 Conclusion 584

16 Testing and Debugging 585
16.1 Testing with RSpec 586
16.2 Testing with Minitest 589
16.3 Testing with Cucumber 594
16.4 Using the `byebug` Debugger 596
16.5 Using `pry` for Debugging 600
16.6 Measuring Performance 601
16.7 Pretty-Printing Objects 606
16.8 Not Covered Here 608
16.9 Conclusion 609

17 Packaging and Distributing Code 611
17.1 Libraries and Rubygems 612
17.1.1 Using Rubygems 612
17.1.2 Creating Gems 613
17.2 Managing Dependencies with Bundler 614
17.2.1 Semantic Versioning 615
17.2.2 Dependencies from Git 616
17.2.3 Creating Gems with Bundler 617
17.2.4 Private Gems 617
17.3 Using RDoc 618
17.3.1 Simple Markup 620
17.3.2 Advanced Documentation with Yard 622
17.4 Conclusion 623

18 Network Programming 625
18.1 Network Servers 627
18.1.1 A Simple Server: Time of Day 627
18.1.2 Implementing a Threaded Server 629
18.1.3 Case Study: A Peer-to-Peer Chess Server 630

18.2 Network Clients 638
18.2.1 Retrieving Truly Random Numbers from the Web 638
18.2.2 Contacting an Official Timeserver 641
18.2.3 Interacting with a POP Server 642
18.2.4 Sending Mail with SMTP 644
18.2.5 Interacting with an IMAP Server 647
18.2.6 Encoding/Decoding Attachments 649
18.2.7 Case Study: A Mail-News Gateway 651
18.2.8 Retrieving a Web Page from a URL 657
18.2.9 Using the Open-URI Library 658
18.3 Conclusion 658

19 Ruby and Web Applications 661
19.1 HTTP Servers 662
19.1.1 A Simple HTTP Server 662
19.1.2 Rack and Web Servers 664
19.2 Application Frameworks 667
19.2.1 Routing in Sinatra 668
19.2.2 Routing in Rails 669
19.2.3 Parameters in Sinatra 671
19.2.4 Parameters in Rails 672
19.3 Storing Data 673
19.3.1 Databases 674
19.3.2 Data Stores 676
19.4 Generating HTML 677
19.4.1 ERB 678
19.4.2 Haml 680
19.4.3 Other Templating Systems 681
19.5 The Asset Pipeline 681
19.5.1 CSS and Sass 682
19.5.2 JavaScript and CoffeeScript 683
19.6 Web Services via HTTP 686
19.6.1 JSON for APIs 686
19.6.2 REST (and REST-ish) APIs 687

19.7 Generating Static Sites 688
19.7.1 Middleman 688
19.7.2 Other Static Site Generators 690
19.8 Conclusion 690

20 Distributed Ruby 691
20.1 An Overview: Using `drb` 692
20.2 Case Study: A Stock Ticker Simulation 695
20.3 Rinda: A Ruby Tuplespace 698
20.4 Service Discovery with Distributed Ruby 703
20.5 Conclusion 704

21 Ruby Development Tools 705
21.1 Using Rake 706
21.2 Using `irb` 710
21.3 The Basics of `pry` 715
21.4 The `ri` Utility 716
21.5 Editor Support 717
21.5.1 Vim 717
21.5.2 Emacs 718
21.6 Ruby Version Managers 719
21.6.1 Using `rvm` 719
21.6.2 Using `rbenv` 720
21.6.3 Using `chruby` 721
21.7 Conclusion 722

22 The Ruby Community 723
22.1 Web Resources 723
22.2 Mailing Lists, Podcasts, and Forums 724
22.3 Ruby Bug Reports and Feature Requests 724
22.4 IRC Channels 725
22.5 Ruby Conferences 725
22.6 Local Ruby Groups 726
22.7 Conclusion 726

Index 727

Foreword

Foreword to the Third Edition

Yesterday I was reading an article about geek fashion in `Wired.com`. According to it, wearing a Rubyconf 2012 t-shirt these days signals to people: "I work for Oracle."

Wow. How far we've come in the last 10 years!

For quite some time, using Ruby set you apart from the mainstream. Now it seems we are the mainstream. And what a long, strange journey it has been to get there.

Ruby adoption took a long time by today's standards. I read this book in 2005, and at that point, the first edition was over four years old. Ruby had just begun its second wave of adoption thanks to DHH and the start of Rails mania. It seemed like there might be a couple hundred people in the entire (English-speaking) world that used Ruby. Amazingly, at that point, the first edition of this book was already four years old. That's how ahead of its time it was.

This new edition keeps the writing style that has made the book such a hit with experienced programmers over the years. The long first chapter covers fundamental basics of object-orientation and the Ruby language. It's a must read for anyone new to the language. But it does so in concise, fast-moving narrative that assumes you already know how to create software.

From there, the chapters follow a distinctive pattern. A bit of backstory narrative, followed by rapid-fire bits of knowledge about the Ruby language. Snippets of example code are abundant and help to illuminate the concept under discussion. You can lift code samples verbatim into your programs. Especially once you get into the more practical applications chapters later in the book.

A brief bit of personal backstory seems appropriate. I owe a huge debt of gratitude to Hal for this book and the way that he wrote it. In 2005, I started work on a manuscript for Addison Wesley about the use of Ruby on Rails in the enterprise. It was my first attempt at authoring a book, and after penning about two chapters, I got stuck. Few people were using Ruby or Rails in the enterprise at that time and I had to remind myself that I was attempting to write non-fiction.

After discussing options with my editor, we determined that the best course of action might be to ditch the idea and start on a new one. *The Rails Way* was to cover the nascent Ruby on Rails framework in the style of this book. I employed terse narrative accompanying plentiful code examples. Instead of long listings, I interspersed commentary between sprinkles of code that provided just enough samples of the framework to make sense.

Like *The Ruby Way*, I aimed for breadth of coverage rather than depth. I wanted *The Rails Way* to claim permanent real estate on the desk of the serious Rails programmer. Like *The Ruby Way*, I wanted my book to be a default go-to reference. In contrast to other Rails books, I skipped tutorial material and ignored complete beginners.

And it was a huge success! Safe to say that without Hal's book, my own book would not exist and my career would have taken a less successful trajectory.

But enough congratulatory retrospective! Let's get back to the present day and the newest edition of *The Ruby Way* that you're currently reading. The immensely talented André Arko joins Hal this time around. What a great team! They deliver a painstaking revision that brings the book up to date with the latest edition of our beloved Ruby language.

My personal highlights of this edition include the following:

- A whole chapter of in-depth coverage of the new Onigmo regular expression engine. I love its beautiful and concise explanations of concepts such as positive and negative lookahead and lookbehind.
- The Internationalization chapter tackles thorny issues around String encoding and Unicode normalization. Bloggers have covered the subject in spotty fashion over the years, but having it all presented in one place is invaluable.
- The Ruby and Web Applications chapter manages to squeeze a crash-course in Rack, Sinatra, and Rails into less than 30 pages.

* Want proof of André's ingenuity? See how he cuts the load time for a real Rails app down to 500ms or less at http://andre.arko.net/2014/06/27/rails-in-05-seconds/.

I predict that this edition of *The Ruby Way* will be as successful as its predecessors. It gives me great joy to make it the latest addition to our Professional Ruby Series.

Obie Fernandez
September 15, 2014

Foreword to the Second Edition

In ancient China, people, especially philosophers, thought that something was hidden behind the world and every existence. It can never be told, nor explained, nor described in concrete words. They called it *Tao* in Chinese and *Do* in Japanese. If you translate it into English, it is the word for *Way.* It is the *Do* in Judo, Kendo, Karatedo, and Aikido. They are not only martial arts, but they also include a philosophy and a way of life.

Likewise, Ruby the programming language has its philosophy and way of thinking. It enlightens people to think differently. It helps programmers have more fun in their work. It is not because Ruby is from Japan but because programming is an important part of the human being (well, at least *some* human beings), and Ruby is designed to help people have a better life.

As always, "Tao" is difficult to describe. I feel it but have never tried to explain it in words. It's just too difficult for me, even in Japanese, my native tongue. But a guy named Hal Fulton tried, and his first try (the first edition of this book) was pretty good. This second version of his trial to describe the Tao of Ruby becomes even better with help from many people in the Ruby community. As Ruby becomes more popular (partly due to Ruby on Rails), it becomes more important to understand the secret of programmers' productivity. I hope this book helps you to become an efficient programmer.

Happy Hacking.

Yukihiro "Matz" Matsumoto
August 2006, Japan
まつもと ゆきひろ

Foreword to the First Edition

Shortly after I first met with computers in the early 80s, I became interested in programming languages. Since then I have been a "language geek." I think the reason for this interest is that programming languages are ways to express human thought. They are fundamentally human-oriented.

Despite this fact, programming languages have tended to be machine-oriented. Many languages were designed for the convenience of the computer.

But as computers became more powerful and less expensive, this situation gradually changed. For example, look at structured programming. Machines do not care whether programs are structured well; they just execute them bit by bit. Structured programming is not for machines, but for humans. This is true of object-oriented programming as well.

The time for language design focusing on humans has been coming.

In 1993, I was talking with a colleague about scripting languages, about their power and future. I felt scripting to be the way future programming should be—human-oriented.

But I was not satisfied with existing languages such as Perl and Python. I wanted a language that was more powerful than Perl and more object-oriented than Python. I couldn't find the ideal language, so I decided to make my own.

Ruby is not the simplest language, but the human soul is not simple in its natural state. It loves simplicity and complexity at the same time. It can't handle too many complex things, nor too many simple things. It's a matter of balance.

So to design a human-oriented language, Ruby, I followed the Principle of Least Surprise. I consider that everything that surprises me less is good. As a result, I feel a natural feeling, even a kind of joy, when programming in Ruby. And since the first release of Ruby in 1995, many programmers worldwide have agreed with me about the joy of Ruby programming.

As always I'd like to express my greatest appreciation to the people in the Ruby community. They are the heart of Ruby's success.

I am also thankful to the author of this book, Hal E. Fulton, for declaring the Ruby Way to help people.

This book explains the philosophy behind Ruby, distilled from my brain and the Ruby community. I wonder how it can be possible for Hal to read my mind to know and reveal this secret of the Ruby Way. I have never met him face to face; I hope to meet him soon.

I hope this book and Ruby both serve to make your programming fun and happy.

Yukihiro "Matz" Matsumoto
September 2001, Japan
まつもと ゆきひろ

Acknowledgments

Acknowledgments for the Third Edition

As can be expected by now, the process of updating this book for the third edition turned out to be somewhat monumental. Ruby has changed dramatically since the days of 1.8, and being a Ruby programmer is far more popular now than it has ever been before.

Verifying, updating, and rewriting this book took quite some time longer than expected. Ruby has progressed from 1.9 through 2.0 and 2.1, and this book has progressed through at least as many edits and rewrites along the way.

Many people contributed to making this book possible. At Addison-Wesley, Debra Williams Cauley, Songlin Qiu, Andy Beaster, and Bart Reed provided the encouragement, coordination, and editing needed to complete this edition. The contributions of Russ Olsen and André Arko were *absolutely invaluable.*

This edition was technically edited by Russ Olsen and Steve Klabnik, providing feedback and suggestions that made the book more accurate and understandable. Russ also provided the Ruby libraries and scripts that compiled the latest version of the book itself. As always, any errors are mine, not theirs.

Suggestions, code samples, or simply helpful explanations were provided by Dave Thomas, David Alan Black, Eric Hodel, Chad Fowler, Brad Ediger, Sven Fuchs, Jesse Storimer, Luke Francl, and others over the years.

Special thanks go to Paul Harrison and the rest of my colleagues at Simpli.fi for their encouragement and support.

I also wish to honor the memory of Guy Decoux and more recently Jim Weirich. Jim in particular made significant contributions to this book and to our community.

Final thanks are owed, as always, to Matz himself for creating Ruby, and to you, the reader of this book. I hope it is able to teach, inform, and maybe even amuse you.

Acknowledgments for the Second Edition

Common sense says that a second edition will only require half as much work as the first edition required. Common sense is wrong.

Even though a large part of this book came directly from the first edition, even that part had to be tweaked and tuned. Every single sentence in this book had to pass through (at the very least) a filter that asked: Is what was true in 2001 still true in 2006? And that, of course, was only the beginning.

In short, I put in many hundreds of hours of work on this second edition—nearly as much time as on the first. And yet I am "only the author."

A book is possible only through the teamwork of many people. On the publisher's side, I owe thanks to Debra Williams-Cauley, Songlin Qiu, and Mandie Frank for their hard work and infinite patience. Thanks go to Geneil Breeze for her tireless copy editing and picking bits of lint from my English. There are also others I can't name because their work was completely behind the scenes, and I never talked with them.

Technical editing was done primarily by Shashank Date and Francis Hwang. They did a great job, and I appreciate it. Errors that slipped through are my responsibility, of course.

Thanks go to the people who supplied explanations, wrote sample code, and answered numerous questions for me. These include Matz himself (Yukihiro Matsumoto), Dave Thomas, Christian Neukirchen, Chad Fowler, Curt Hibbs, Daniel Berger, Armin Roehrl, Stefan Schmiedl, Jim Weirich, Ryan Davis, Jenny W., Jim Freeze, Lyle Johnson, Martin DeMello, Matt Lawrence, the infamous *why the lucky stiff,* Ron Jeffries, Tim Hunter, Chet Hendrickson, Nathaniel Talbott, and Bil Kleb.

Special thanks goes to the heavier contributors. Andrew Johnson greatly enhanced my regular expression knowledge. Paul Battley made great contributions to the internationalization chapter. Masao Mutoh added to that same chapter and also contributed material on GTK. Austin Ziegler taught me the secrets of writing PDF files. Caleb Tennis added to the Qt material. Eric Hodel added to the Rinda and Ring material, and James Britt contributed heavily to the web development chapter.

Thanks and appreciation again must go to Matz, not only for his assistance but for creating Ruby in the first place. *Domo arigato gozaimasu*!

Again I have to thank my parents. They have encouraged me without ceasing and are looking forward to seeing this book. I will make programmers of them both yet.

And once again, I have to thank all of the Ruby community for their tireless energy, productivity, and community spirit. I particularly thank the readers of this book (in both editions). I hope you find it informative, useful, and perhaps even entertaining.

Acknowledgments for the First Edition

Writing a book is truly a team effort; this is a fact I could not fully appreciate until I wrote one myself. I recommend the experience, although it is a humbling one. It is a simple truth that without the assistance of many other people, this book would not have existed.

Thanks and appreciation must first go to Matz (Yukihiro Matsumoto), who created the Ruby language in the first place. Domo arigato gozaimasu!

Thanks goes to Conrad Schneiker for conceiving the overall idea for the book and helping to create its overall structure. He also did me the service of introducing me to the Ruby language in 1999.

Several individuals have contributed material to the body of the book. The foremost of these was Guy Hurst, who wrote substantial parts of the earlier chapters as well as two of the appendices. His assistance was absolutely invaluable.

Thanks also goes to the other contributors, whom I'll name in no particular order. Kevin Smith did a great job on the GTK section of Chapter 6, saving me from a potentially steep learning curve on a tight schedule. Patrick Logan, in the same chapter, shed light on the mysteries of the FOX GUI. Chad Fowler, in Chapter 9, plumbed the depths of XML and also contributed to the CGI section.

Thanks to those who assisted in proofreading or reviewing or in other miscellaneous ways: Don Muchow, Mike Stok, Miho Ogishima, and others already mentioned. Thanks to David Eppstein, the mathematics professor, for answering questions about graph theory.

One of the great things about Ruby is the support of the community. There were many on the mailing list and the newsgroup who answered questions and gave me ideas and assistance. Again in no particular order, these are Dave Thomas, Andy Hunt, Hee-Sob Park, Mike Wilson, Avi Bryant, Yasushi Shoji ("Yashi"), Shugo Maeda, Jim Weirich, "arton," and Masaki Suketa. I'm sorry to say I have probably overlooked someone.

To state the obvious, a book would never be published without a publisher. Many people behind the scenes worked hard to produce this book; primarily I have to thank William Brown, who worked closely with me and was a constant source of encouragement; and Scott Meyer, who delved deeply into the details of putting the material together. Others I cannot even name because I have never heard of them. You know who you are.

I have to thank my parents, who watched this project from a distance, encouraged me along the way, and even bothered to learn a little bit of computer science for my sake.

A writer friend of mine once told me, “If you write a book and nobody reads it, you haven’t really written a book.” So, finally, I want to thank the reader. This book is for you. I hope it is of some value.

About the Authors

Hal Fulton first began using Ruby in 1999. In 2001, he started work on *The Ruby Way*, which was the second Ruby book published in English. Fulton was an attendee at the very first Ruby conference in 2001 and has presented at numerous other Ruby conferences on three continents, including the first European Ruby Conference in 2003. He holds two degrees in computer science from the University of Mississippi and taught computer science for four years. He has worked for more than 25 years with various forms of UNIX and Linux. He is now at Simpli.fi in Fort Worth, Texas, where he works primarily in Ruby.

André Arko first encountered Ruby as a student in 2004, and reading the first edition of this book helped him decide to pursue a career as a Ruby programmer. He is team lead of Bundler, the Ruby dependency manager, and has created or contributes to dozens of other open source projects. He works at Cloud City Development as a consultant providing team training and expertise on Ruby and Rails as well as developing web applications.

André enjoys sharing hard-won knowledge and experience with other developers, and has spoken at over a dozen Ruby conferences on four continents. He is a regular volunteer at RailsBridge and RailsGirls programming outreach events, and works to increase diversity and inclusiveness in both the Ruby community and technology as a field. He lives in San Francisco, California.

Introduction

The way that can be named is not the true Way.

—*Lao Tse,* Tao Te Ching

The title of this book is *The Ruby Way.* This is a title that begs for a disclaimer.

It has been my aim to align this book with the philosophy of Ruby as well as I could. That has also been the aim of the other contributors. Credit for success must be shared with these others, but the blame for any mistakes must rest solely with me.

Of course, I can't presume to tell you with exactness what the spirit of Ruby is all about. That is primarily for Matz to say, and I think even he would have difficulty communicating all of it in words.

In short, *The Ruby Way* is only a book, but the Ruby Way is the province of the language creator and the community as a whole. This is something difficult to capture in a book.

Still, I have tried in this introduction to pin down a little of the ineffable spirit of Ruby. The wise student of Ruby will not take it as totally authoritative.

About the Third Edition

Everything changes, and Ruby is no exception. There are many changes and much new material in this edition. In a larger sense, every single chapter in this book is "new." I have revised and updated every one of them, making hundreds of minor changes and dozens of major changes. I deleted items that were obsolete or of lesser importance; I changed material to fit changes in Ruby itself; I added examples and commentary to every chapter.

As the second Ruby book in the English language (after *Programming Ruby*, by Dave Thomas and Andy Hunt), *The Ruby Way was* designed to be complementary to that book rather than overlap with it; that is still true today.

There have been numerous changes between Ruby 1.8, covered in the second edition, and Ruby 2.1, covered here. It's important to realize, however, that these were made with great care, over several years. Ruby is still Ruby. Much of the beauty of Ruby is derived from the fact that it changes slowly and deliberately, crafted by the wisdom of Matz and the other developers.

Today we have a proliferation of books on Ruby and more articles published than we can bother to notice. Web-based tutorials and documentation resources abound.

New tools and libraries have appeared. The most common of these seem to be tools by developers for other developers: web frameworks, blogging tools, markup tools, and interfaces to exotic data stores. But there are many others, of course—GUIs, number-crunching, web services, image manipulation, source control, and more.

Ruby editor support is widespread and sophisticated. IDEs are available that are useful and mature (and which share some overlap with the GUI builders).

It's also undeniable that the community has grown and changed. Ruby is by no means a niche language today; it is used in government departments such as NASA and NOAA, enterprise companies such as IBM and Motorola, and well-known websites such as Wikipedia, GitHub, and Twitter. It is used for graphics work, database work, numerical analysis, web development, and more. In short—and I mean this in the positive sense—Ruby has gone mainstream.

Updating this book has been a labor of love. I hope it is useful to you.

How This Book Works

You probably won't learn Ruby from this book. There is relatively little in the way of introductory or tutorial information. If you are totally new to Ruby, you might want start with another book.

Having said that, programmers are a tenacious bunch, and I grant that it might be possible to learn Ruby from this book. Chapter 1, "Ruby in Review," does contain a brief introduction and some tutorial information.

Chapter 1 also contains a comprehensive "gotcha" list (which has been difficult to keep up to date). The usefulness of this list in Chapter 1 will vary widely from one reader to another because we cannot all agree on what is intuitive.

This book is largely intended to answer questions of the form "How do I…?." As such, you can expect to do a lot of skipping around. I'd be honored if everyone read every page from front to back, but I don't expect that. It's more my expectation that you will browse the table of contents in search of techniques you need or things you find interesting.

As it turns out, I have talked to many people since the first edition, and they *did* in fact read it cover to cover. What's more, I have had more than one person report to me that they did learn Ruby here. So anything is possible.

Some things this book covers may seem elementary. That is because people vary in background and experience; what is obvious to one person may not be to another. I have tried to err on the side of completeness. On the other hand, I have tried to keep the book at a reasonable size (obviously a competing goal).

This book can be viewed as a sort of "inverted reference." Rather than looking up the name of a method or a class, you will look things up by function or purpose. For example, the `String` class has several methods for manipulating case: `capitalize`, `upcase`, `casecmp`, `downcase`, and `swapcase`. In a reference work, these would quite properly be listed alphabetically, but in this book they are all listed together.

Of course, in striving for completeness, I have sometimes wandered onto the turf of the reference books. In many cases, I have tried to compensate for this by offering more unusual or diverse examples than you might find in a reference.

I have tried for a high code-to-commentary ratio. Overlooking the initial chapter, I think I've achieved this. Writers may grow chatty, but programmers always want to see the code. (If not, they *should* want to.)

The examples here are sometimes contrived, for which I must apologize. To illustrate a technique or principle *in isolation from a real-world problem* can be difficult. However, the more complex or high level the task was, the more I attempted a real-world solution. Thus, if the topic is concatenating strings, you may find an unimaginative code fragment involving `"foo"` and `"bar"`, but when the topic is something like parsing XML, you will usually find a much more meaningful and realistic piece of code.

This book has two or three small quirks to which I'll confess up front. One is the avoidance of the "ugly" Perl-like global variables such as `$_` and the others. These are present in Ruby, and they work fine; they are used daily by most or all Ruby programmers. But in nearly all cases, their use can be avoided, and I have taken the liberty of omitting them in most of the examples.

Another quirk is that I avoid using standalone expressions when they don't have side effects. Ruby is expression oriented, and that is a good thing; I have tried to take advantage of that feature. But in a code fragment, I prefer to not write expressions that merely return a value that is not usable. For example, the expression `"abc" + "def"` can illustrate string concatenation, but I would write something like `str = "abc" + "def"` instead. This may seem wordy to some, but it may seem more natural to you if you are a C programmer who really notices when functions are void or nonvoid (or an old-time Pascal programmer who thinks in procedures and functions).

My third quirk is that I don't like the "pound" notation to denote instance methods. Many Rubyists will think I am being verbose in saying "instance method `crypt` of class `String`" rather than saying `String#crypt`, but I think no one will be confused. (Actually, I am slowly being converted to this usage, as it is obvious the pound notation is not going away.)

I have tried to include "pointers" to outside resources whenever appropriate. Time and space did not allow putting everything into this book that I wanted, but I hope I have partially made up for that by telling you where to find related materials. The ruby-doc.org and rdoc.info websites are probably the foremost of these sources; you will see them referenced many times in this book.

Here, at the front of the book, there is usually a gratuitous reference to the typefaces used for code, and how to tell code fragments from ordinary text. But I won't insult your intelligence; you've read computer books before.

I want to point out that roughly 10 percent of this book was written by other people. That does not even include tech editing and copy editing. You should read the acknowledgments in this (and every) book. Most readers skip them. Go read them now. They're good for you, like vegetables.

About the Book's Source Code

Every significant code fragment has been collected into an archive for the reader to download. Look for this archive on the informit.com site or at the book's own site, therubyway.io.

It is offered both as a `.tgz` file and as a `.zip` file. Code fragments that are very short or can't be run "out of context" will usually not appear in the archive.

What Is the "Ruby Way"?

> *Let us prepare to grapple with the ineffable itself, and see if we may not eff it after all.*
>
> —*Douglas Adams,* Dirk Gently's Holistic Detective Agency

What do we mean by the Ruby Way? My belief is that there are two related aspects: One is the philosophy of the design of Ruby; the other is the philosophy of its usage. It is natural that design and use should be interrelated, whether in software or hardware; why else should there be such a field as ergonomics? If I build a device and put a handle on it, it is because I expect someone to grab that handle.

Ruby has a nameless quality that makes it what it is. We see that quality present in the design of the syntax and semantics of the language, but it is also present in the programs written for that interpreter. Yet as soon as we make this distinction, we blur it.

Clearly Ruby is not just a tool for creating software, but it is a piece of software in its own right. Why should the workings of Ruby *programs* follow laws different from those that guide the workings of the *interpreter*? After all, Ruby is highly dynamic and extensible. There might be reasons that the two levels should differ here and there, probably for accommodating to the inconvenience of the real world. But in general, the thought processes can and should be the same. Ruby could be implemented in Ruby, in true Hofstadter-like fashion, though it is not at the time of this writing.

We don't often think of the etymology of the word *way*, but there are two important senses in which it is used. On the one hand, it means *a method or technique*, but it can also mean *a road or path*. Obviously these two meanings are interrelated, and I think when I say "the Ruby Way," I mean both of them.

So what we are talking about is a thought process, but it is also a path that we follow. Even the greatest software guru cannot claim to have reached perfection but only to follow the path. And there may be more than one path, but here I can only talk about one.

The conventional wisdom says that *form follows function*. And the conventional wisdom is, of course, conventionally correct. But Frank Lloyd Wright (speaking in his own field) once said, "Form follows function—that has been misunderstood. Form and function should be one, joined in a spiritual union."

What did Wright mean? I would say that this truth is not something you learn from a book, but from experience.

However, I would argue that Wright expressed this truth elsewhere in pieces easier to digest. He was a great proponent of simplicity, saying once, "An architect's most useful tools are an eraser at the drafting board and a wrecking bar at the site."

So, one of Ruby's virtues is simplicity. Shall I quote other thinkers on the subject? According to Antoine de St. Exupéry, "Perfection is achieved, not when there is nothing left to add, but when there is nothing left to take away."

But Ruby is a complex language. How can I say that it is simple?

If we understood the universe better, we might find a "law of conservation of complexity"—a fact of reality that disturbs our lives like entropy so that we cannot avoid it but can only redistribute it.

And that is the key. We can't avoid complexity, but we can push it around. We can bury it out of sight. This is the old "black box" principle at work; a black box performs a complex task, but it possesses simplicity *on the outside.*

If you haven't already lost patience with my quotations, a word from Albert Einstein is appropriate here: "Everything should be as simple as possible, but no simpler."

So in Ruby, we see simplicity embodied from the programmer's view (if not from the view of those maintaining the interpreter). Yet we also see the capacity for compromise. In the real world, we must bend a little. For example, every entity in a Ruby program should be a true object, but certain values such as integers are stored as immediate values. In a trade-off familiar to computer science students for decades, we have traded elegance of design for practicality of implementation. In effect, we have traded one kind of simplicity for another.

What Larry Wall said about Perl holds true: "When you say something in a small language, it comes out big. When you say something in a big language, it comes out small." The same is true for English. The reason that biologist Ernst Haeckel could say "Ontogeny recapitulates phylogeny" in only three words was that he had these powerful words with highly specific meanings at his disposal. We allow inner complexity of the language because it enables us to shift the complexity away from the individual utterance.

I would state this guideline this way: Don't write 200 lines of code when ten will do.

I'm taking it for granted that brevity is generally a good thing. A short program fragment will take up less space in the programmer's brain; it will be easier to grasp as a single entity. As a happy side effect, fewer bugs will be injected while the code is being written.

Of course, we must remember Einstein's warning about simplicity. If we put brevity too high on our list of priorities, we will end up with code that is hopelessly obfuscated. Information theory teaches us that compressed data is statistically similar to random noise; if you have looked at C or APL or regular expression notation—especially badly written—you have experienced this truth firsthand. "Simple, but not too simple"; that is the key. Embrace brevity, but do not sacrifice readability.

It is a truism that both brevity and readability are good. But there is an underlying reason for this—one so fundamental that we sometimes forget it. The reason is that computers exist for humans, not humans for computers.

In the old days, it was almost the opposite. Computers cost millions of dollars and ate electricity at the rate of many kilowatts. People acted as though the computer was a minor deity and the programmers were humble supplicants. An hour of the computer's time was more expensive than an hour of a person's time.

When computers became smaller and cheaper, high-level languages also became more popular. These were inefficient from the computer's point of view but efficient from the human perspective. Ruby is simply a later development in this line of thought. Some, in fact, have called it a *VHLL* (Very High-Level Language); though this term is not well-defined, I think its use is justified here.

The computer is supposed to be the servant, not the master, and, as Matz has said, a smart servant should do a complex task with a few short commands. This has been true through all the history of computer science. We started with machine languages and progressed to assembly language and then to high-level languages.

What we are talking about here is a shift from a *machine-centered* paradigm to a *human-centered* one. In my opinion, Ruby is an excellent example of human-centric programming.

I'll shift gears a little. There was a wonderful little book from the 1980s called *The Tao of Programming* (by Geoffrey James). Nearly every line is quotable, but I'll repeat only this: "A program should follow the 'Law of Least Astonishment.' What is this law? It is simply that the program should always respond to the user in the way that astonishes him least." (Of course, in the case of a language interpreter, the *user* is the programmer.)

I don't know whether James coined this term, but his book was my first introduction to the phrase. This is a principle that is well known and often cited in the Ruby community, though it is usually called the *Principle of Least Surprise*, or *POLS*. (I myself stubbornly prefer the acronym *LOLA*.)

Whatever you call it, this rule is a valid one, and it has been a guideline throughout the ongoing development of the Ruby language. It is also a useful guideline for those who develop libraries or user interfaces.

The only problem, of course, is that different people are surprised by different things; there is no universal agreement on how an object or method "ought" to behave. We can strive for consistency and strive to justify our design decisions, and each person can train his own intuition.

For the record, Matz has said that "least surprise" should refer to *him* as the designer. The more you think like him, the less Ruby will surprise you. And I assure you, imitating Matz is not a bad idea for most of us.

No matter how logically constructed a system may be, your intuition needs to be trained. Each programming language is a world unto itself, with its own set of assumptions, and human languages are the same. When I took German, I learned that all nouns were capitalized, but the word *deutsch* was not. I complained to my professor; after all, this was the *name* of the language, wasn't it? He smiled and said, "Don't fight it."

What he taught me was to *let German be German.* By extension, that is good advice for anyone coming to Ruby from some other language. Let Ruby be Ruby. Don't expect it to be Perl, because it isn't; don't expect it to be LISP or Smalltalk, either. On the other hand, Ruby has common elements with all three of these. Start by following your expectations, but when they are violated, don't fight it (unless Matz agrees it's a needed change).

Every programmer today knows the orthogonality principle (which would better be termed the *orthogonal completeness principle*). Suppose we have an imaginary pair of axes with a set of comparable language entities on one and a set of attributes or capabilities on the other. When we talk of "orthogonality," we usually mean that the space defined by these axes is as "full" as we can logically make it.

Part of the Ruby Way is to strive for this orthogonality. An array is in some ways similar to a hash, so the operations on each of them should be similar. The limit is reached when we enter the areas where they are different.

Matz has said that "naturalness" is to be valued over orthogonality. But to fully understand what is natural and what is not may take some thinking and some coding.

Ruby strives to be friendly to the programmer. For example, there are aliases or synonyms for many method names; `size` and `length` will both return the number of entries in an array. Some consider this sort of thing to be an annoyance or anti-feature, but I consider it a good design.

Ruby strives for consistency and regularity. There is nothing mysterious about this; in every aspect of life, we yearn for things to be regular and parallel. What makes it a little more tricky is learning when to violate this principle.

For instance, Ruby has the habit of appending a question mark (`?`) to the name of a predicate-like method. This is well and good; it clarifies the code and makes the namespace a little more manageable. But what is more controversial is the similar use of the exclamation point in marking methods that are "destructive" or "dangerous" in the sense that they modify their receivers. The controversy arises because *not all* of the destructive methods are marked in this way. Shouldn't we be consistent?

No, in fact we should not. Some of the methods by their very nature change their receiver (such as the `Array` methods `replace` and `concat`). Some of them are "writer" methods allowing assignment to a class attribute; we should *not* append an exclamation point to the attribute name or the equal sign. Some methods arguably change the state of the receiver, such as `read`; this occurs too frequently to be marked in this way. If every destructive method name ended in a `!`, our programs soon would look like sales brochures for a multilevel marketing firm.

Do you notice a kind of tension between opposing forces, a tendency for all rules to be violated? Let me state this as Fulton's Second Law: *Every rule has an exception, except Fulton's Second Law.* (Yes, there is a joke there, a very small one.)

What we see in Ruby is not a "foolish consistency" nor a rigid adherence to a set of simple rules. In fact, perhaps part of the Ruby Way is that it is *not* a rigid and inflexible approach. In language design, as Matz once said, you should "follow your heart."

Yet another aspect of the Ruby philosophy is, do *not fear change at runtime; do not fear what is dynamic.* The world is dynamic; why should a programming language be static? Ruby is one of the most dynamic languages in existence.

I would also argue that another aspect is, do not be a slave to performance issues. When performance is unacceptable, the issue must be addressed, but it should normally not be the first thing you think about. Prefer elegance over efficiency where efficiency is less than critical. Then again, if you are writing a library that may be used in unforeseen ways, performance may be critical from the start.

When I look at Ruby, I perceive a balance between different design goals, a complex interaction reminiscent of the *n*-body problem in physics. I can imagine it might be modeled as an Alexander Calder mobile. It is perhaps this interaction itself, the harmony, that embodies Ruby's philosophy rather than just the individual parts. Programmers know that their craft is not just science and technology but art. I hesitate to say that there is a spiritual aspect to computer science, but just between you and me, there certainly is. (If you have not read Robert Pirsig's *Zen and the Art of Motorcycle Maintenance*, I recommend that you do so.)

Ruby arose from the human urge to create things that are useful and beautiful. Programs written in Ruby should spring from the same source. That, to me, is the essence of the Ruby Way.

This page intentionally left blank

CHAPTER 2

Working with Strings

Atoms were once thought to be fundamental, elementary building blocks of nature; protons were then thought to be fundamental, then quarks. Now we say the string is fundamental.

—David Gross, professor of theoretical physics, Princeton University

A computer science professor in the early 1980s started out his data structures class with a single question. He didn't introduce himself or state the name of the course; he didn't hand out a syllabus or give the name of the textbook. He walked to the front of the class and asked, "What is the most important data type?"

There were one or two guesses. Someone guessed "pointers," and he brightened but said no, that wasn't it. Then he offered his opinion: The most important data type was *character* data.

He had a valid point. Computers are supposed to be our servants, not our masters, and character data has the distinction of being human readable. (Some humans can read binary data easily, but we will ignore them.) The existence of characters (and therefore strings) enables communication between humans and computers. Every kind of information we can imagine, including natural language text, can be encoded in character strings.

A *string* is simply a sequence of characters. Like most entities in Ruby, strings are first-class objects. In everyday programming, we need to manipulate strings in many ways. We want to concatenate strings, tokenize them, analyze them, perform searches and substitutions, and more. Ruby makes most of these tasks easy.

For much of the history of Ruby, a single byte was considered a character. That is not true of special characters, emoji, and most non-Latin scripts. For a more detailed discussion of the ways that bytes and characters are often not the same, refer to Chapter 4, "Internationalization in Ruby."

2.1 Representing Ordinary Strings

A string in Ruby is composed simply of a sequence of 8-bit bytes. It is not null terminated as in C, so it may contain null characters. Strings containing bytes above 0xFF are always legal, but are only meaningful in non-ASCII encodings. Strings are assumed to use the UTF-8 encoding. Before Ruby 2.0, they were assumed to be simple ASCII. (For more information on encodings, refer to Chapter 4.)

The simplest string in Ruby is single quoted. Such a string is taken absolutely literally; the only escape sequences recognized are the single quote (`\'`) and the escaped backslash itself (`\\`). Here are some examples:

```
s1 = 'This is a string'    # This is a string
s2 = 'Mrs. O\'Leary'       # Mrs. O'Leary
s3 = 'Look in C:\\TEMP'    # Look in C:\TEMP
```

A double-quoted string is more versatile. It allows many more escape sequences, such as backspace, tab, carriage return, and linefeed. It allows control characters to be embedded as octal numbers, and Unicode code points to be embedded via their hexadecimal reference number. Consider these examples:

```
s1 = "This is a tab: (\t)"
s2 = "Some backspaces: xyz\b\b\b"
s3 = "This is also a tab: \011"
s4 = "And these are both bells: \a \007"
s5 = "This is the unicode snowman: \u2603"
```

Non-ASCII characters will be shown "backslash escaped" when their string is inspected, but will print normally. Double-quoted strings also allow expressions to be embedded inside them. See Section 2.21, "Embedding Expressions within Strings."

2.2 Representing Strings with Alternate Notations

Sometimes we want to represent strings that are rich in metacharacters, such as single quotes, double quotes, and more. For these situations, we have the %q and %Q notations. Following either of these is a string within a pair of delimiters; I personally favor square brackets ([ ]).

The difference between the %q and %Q variants is that the former acts like a single-quoted string, and the latter like a double-quoted string:

```
S1 = %q[As Magritte said, "Ceci n'est pas une pipe."]
s2 = %q[This is not a tab: (\t)]  # same as: 'This is not a tab: \t'
s3 = %Q[This IS a tab: (\t)]      # same as: "This IS a tab: \t"
```

Both kinds of notation can be used with different delimiters. Besides brackets, there are other paired delimiters (parentheses, braces, and angle brackets):

```
s1 = %q(Bill said, "Bob said, 'This is a string.'")
s2 = %q{Another string.}
s3 = %q<Special characters '"[](){} in this string.>
```

There are also “nonpaired” delimiters. Basically any character may be used that is printable, but not alphanumeric, not whitespace, and not a paired character:

```
s1 = %q:"I think Mrs. O'Leary's cow did it," he said.:
s2 = %q*\r is a control-M and \n is a control-J.*
```

2.3 Using Here-Documents

If you want to represent a long string spanning multiple lines, you can certainly use a regular quoted string:

```
str = "Once upon a midnight dreary,
       While I pondered, weak and weary..."
```

However, the indentation will be part of the string.

Another way is to use a *here-document,* a string that is inherently multiline. (This concept and term are borrowed from older languages and contexts.) The syntax is the << symbol, followed by an end marker, then zero or more lines of text, and finally the same end marker on a line by itself:

```
str = <<EOF
Once upon a midnight dreary,
While I pondered weak and weary,...
EOF
```

Be careful about things such as trailing spaces on the final end marker line. Current versions of Ruby will fail to recognize the end marker in those situations.

Note that here-documents may be "stacked"; for example, here is a method call with three such strings passed to it:

```
some_method(<<STR1, <<STR2, <<STR3)
first piece
of text...
STR1
second piece...
STR2
third piece
of text.
STR3
```

By default, a here-document is like a double-quoted string—that is, its contents are subject to interpretation of escape sequences and interpolation of embedded expressions. But if the end marker is single-quoted, the here-document behaves like a single-quoted string:

```
str = <<'EOF'
This isn't a tab: \t
and this isn't a newline: \n
EOF
```

If a here-document's end marker is preceded by a hyphen, the end marker may be indented. *Only* the spaces before the end marker are deleted from the string, not those on previous lines:

```
str = <<-EOF
  Each of these lines
  starts with a pair
  of blank spaces.
  EOF
```

To delete the spaces from the beginning of each line, we need another method. The `ActiveSupport` gem (included in Rails) defines a `strip_heredoc` method that works similarly to this one:

```
class String
  def strip_heredoc
    # Find the margin whitespace on the first line
    margin = self[/\A\s*/]
    # Remove margin-sized whitespace from each line
    gsub(/\s{#{margin.size}}/,"")
  end
end
```

The amount of whitespace before the start of the first line is detected, and that amount of whitespace is then stripped off of each line. It's used in this way:

```
str = <<end.strip_heredoc
  This here-document has a "left margin"
  set by the whitespace on the first line.

  We can do inset quotations here,
  hanging indentions, and so on.
end
```

The word *end* is used naturally enough as an end marker. (This, of course, is a matter of taste. It looks like the reserved word `end` but is really just an arbitrary marker.) Many text editors use the end marker as a hint for syntax highlighting. As a result, using `<<SQL` or `<<RUBY` can make it dramatically easier to read blocks of code inside here-docs in those editors.

2.4 Finding the Length of a String

The method `length` can be used to find a string's length. A synonym is `size`:

```
str1 = "Carl"
x = str1.length     # 4
str2 = "Doyle"
x = str2.size       # 5
```

2.5 Processing a Line at a Time

A Ruby string can contain newlines. For example, a file can be read into memory and stored in a single string. Strings provide an iterator, `each_line`, to process a string one line at a time:

```
str = "Once upon\na time...\nThe End\n"
num = 0
str.each_line do |line|
  num += 1
  print "Line #{num}: #{line}"
end
```

The preceding code produces three lines of output:

```
Line 1: Once upon
Line 2: a time...
Line 3: The End
```

Iterators (such as `each_line`) can be chained together with other iterators (such as `with_index`). Connecting function outputs and inputs in a line like this is a technique sometimes called *function composition* (or *method chaining*). Instead of tracking the line number manually, `with_index` can be *composed* with `each_line` to produce the exact same output:

```
str = "Once upon\na time...\nThe End\n"
str.each_line.with_index do |line, num|
  print "Line #{num + 1}: #{line}"
end
```

2.6 Processing a Character or Byte at a Time

Ruby used to treat each byte as a character, but that is no longer the case. The bytes in a string are available as an array via the `bytes` method. To process the bytes, one at a time, use the `each_byte` iterator:

```
str = "ABC"
str.each_byte {|byte| print byte, " " }
puts
# Produces output: 65 66 67
```

A character is essentially the same as a one-character string. In multibyte encodings, a one-character string may be more than one byte:

```
str = "ABC"
str.each_char {|char| print char, " " }
puts
# Produces output: A B C
```

In any version of Ruby, you can break a string into an array of one-character strings by using scan with a simple wildcard regular expression matching a single character:

```
str = "ABC"
chars = str.scan(/./)
chars.each {|char| print char, " " }
puts
# Produces output: A B C
```

2.7 Performing Specialized String Comparisons

Ruby has built-in ideas about comparing strings; comparisons are done lexicographically, as we have come to expect (that is, based on character set order). But if we want, we can introduce rules of our own for string comparisons, and these can be of arbitrary complexity.

For example, suppose that we want to ignore the English articles *a*, *an*, and *the* at the front of a string, and we also want to ignore most common punctuation marks. We can do this by overriding the built-in method <=> (which is called for <, <=, >, and >=). Listing 2.1 shows how we do this.

Listing 2.1 Specialized String Comparisons

```
class String

  alias old_compare <=>

  def <=>(other)
    a = self.dup
    b = other.dup
    # Remove punctuation
    a.gsub!(/[\,\.\?\!\:\;]/, "")
    b.gsub!(/[\,\.\?\!\:\;]/, "")
    # Remove initial articles
    a.gsub!(/^(a |an |the )/i, "")
    b.gsub!(/^(a |an |the )/i, "")
    # Remove leading/trailing whitespace
    a.strip!
    b.strip!
    # Use the old <=>
    a.old_compare(b)
  end

end


title1 = "Calling All Cars"
title2 = "The Call of the Wild"

# Ordinarily this would print "yes"

if title1 < title2
  puts "yes"
else
  puts "no"         # But now it prints "no"
end
```

Note that we "save" the old <=> with an alias and then call it at the end. This is because if we tried to use the < method, it would call the new <=> rather than the old one, resulting in infinite recursion and a program crash.

Note also that the == operator does not call the <=> method (mixed in from `Comparable`). This means that if we need to check equality in some specialized way, we will have to override the == method separately. But in this case, == works as we want it to anyhow.

Suppose that we wanted to do case-insensitive string comparisons. The built-in method `casecmp` will do this; we just have to make sure that it is used instead of the usual comparison.

Here is one way:

```
class String
  def <=>(other)
    casecmp(other)
  end
end
```

But there is a slightly easier way:

```
class String
  alias <=> casecmp
end
```

However, we haven't finished. We need to redefine == so that it will behave in the same way:

```
class String
  def ==(other)
    casecmp(other) == 0
  end
end
```

Now all string comparisons will be strictly case insensitive. Any sorting operation that depends on <=> will likewise be case insensitive.

2.8 Tokenizing a String

The `split` method parses a string and returns an array of tokenized strings. It accepts two parameters: a delimiter and a field limit (which is an integer).

The delimiter defaults to whitespace. Actually, it uses `$;` or the English equivalent `$FIELD_SEPARATOR`. If the delimiter is a string, the explicit value of that string is used as a token separator:

```
s1 = "It was a dark and stormy night."
words = s1.split          # ["It", "was", "a", "dark", "and",
                          #  "stormy", "night"]
s2 = "apples, pears, and peaches"
list = s2.split(", ")     # ["apples", "pears", "and peaches"]

s3 = "lions and tigers and bears"
zoo = s3.split(/ and /)   # ["lions", "tigers", "bears"]
```

The `limit` parameter places an upper limit on the number of fields returned, according to these rules:

- If it is omitted, trailing null entries are suppressed.
- If it is a positive number, the number of entries will be limited to that number (stuffing the rest of the string into the last field as needed). Trailing null entries are retained.

- If it is a negative number, there is no limit to the number of fields, and trailing null entries are retained.

These three rules are illustrated here:

```
str = "alpha,beta,gamma,,"
list1 = str.split(",")     # ["alpha","beta","gamma"]
list2 = str.split(",",2)   # ["alpha", "beta,gamma,,"]
list3 = str.split(",",4)   # ["alpha", "beta", "gamma", ","]
list4 = str.split(",",8)   # ["alpha", "beta", "gamma", "", ""]
list5 = str.split(",",-1)  # ["alpha", "beta", "gamma", "", ""]
```

Similarly, the `scan` method can be used to match regular expressions or strings against a target string:

```
str = "I am a leaf on the wind..."

# A string is interpreted literally, not as a regex
arr = str.scan("a")   # ["a","a","a"]

# A regex will return all matches
arr = str.scan(/\w+/)
# ["I", "am", "a", "leaf", "on", "the", "wind"]

# A block will be passed each match, one at a time
str.scan(/\w+/) {|x| puts x }
```

The `StringScanner` class, from the standard library, is different in that it maintains state for the scan rather than doing it all at once:

```
require 'strscan'
str = "Watch how I soar!"
ss = StringScanner.new(str)
loop do
  word = ss.scan(/\w+/)    # Grab a word at a time
  break if word.nil?
  puts word
  sep = ss.scan(/\W+/)     # Grab next non-word piece
  break if sep.nil?
end
```

2.9 Formatting a String

Formatting a string is done in Ruby as it is in C: with the sprintf method. It takes a string and a list of expressions as parameters and returns a string. The format string contains essentially the same set of specifiers available with C's sprintf (or printf):

```
name = "Bob"
age = 28
str = sprintf("Hi, %s... I see you're %d years old.", name, age)
```

You might ask why we would use this instead of simply interpolating values into a string using the #{expr} notation. The answer is that sprintf makes it possible to do extra formatting, such as specifying a maximum width, specifying a maximum number of decimal places, adding or suppressing leading zeroes, left-justifying, right-justifying, and more:

```
str = sprintf("%-20s  %3d", name, age)
```

The String class has the method %, which does much the same thing. It takes a single value or an array of values of any type:

```
str = "%-20s  %3d" % [name, age]  # Same as previous example
```

We also have the methods ljust, rjust, and center; these take a length for the destination string and pad with spaces as needed:

```
str = "Moby-Dick"
s1 = str.ljust(13)           # "Moby-Dick"
s2 = str.center(13)          # "  Moby-Dick  "
s3 = str.rjust(13)           # "    Moby-Dick"
```

If a second parameter is specified, it is used as the pad string (which may possibly be truncated as needed):

```
str = "Captain Ahab"
s1 = str.ljust(20,"+")       # "Captain Ahab++++++++"
s2 = str.center(20,"-")      # "----Captain Ahab----"
s3 = str.rjust(20,"123")     # "12312312Captain Ahab"
```

2.10 Using Strings as IO Objects

Besides `sprintf` and `scanf`, there is another way to fake input/output to a string—the `StringIO` class.

Because this is a very IO-like object, we cover it in a later chapter. See Section 10.1.24, "Treating a String as a File."

2.11 Controlling Uppercase and Lowercase

Ruby's `String` class offers a rich set of methods for controlling case. This section offers an overview of these.

The `downcase` method converts a string to all lowercase. Likewise, `upcase` converts it to all uppercase. Here is an example each:

```
s1 = "Boston Tea Party"
s2 = s1.downcase              # "boston tea party"
s3 = s2.upcase                # "BOSTON TEA PARTY"
```

The `capitalize` method capitalizes the first character of a string while forcing all the remaining characters to lowercase:

```
s4 = s1.capitalize            # "Boston tea party"
s5 = s2.capitalize            # "Boston tea party"
s6 = s3.capitalize            # "Boston tea party"
```

The `swapcase` method exchanges the case of each letter in a string:

```
s7 = "THIS IS AN ex-parrot."
s8 = s7.swapcase              # "this is an EX-PARROT."
```

There is also the `casecmp` method, which acts like the `<=>` method but ignores case:

```
n1 = "abc".casecmp("xyz")     # -1
n2 = "abc".casecmp("XYZ")     # -1
n3 = "ABC".casecmp("xyz")     # -1
n4 = "ABC".casecmp("abc")     # 0
n5 = "xyz".casecmp("abc")     # 1
```

Each of these also has an in-place equivalent (upcase!, downcase!, capitalize!, and swapcase!).

There are no built-in methods for detecting case, but this is easy to do with regular expressions, as shown in the following example:

```
if string =~ /[a-z]/
  puts "string contains lowercase characters"
end

if string =~ /[A-Z]/
  puts "string contains uppercase characters"
end

if string =~ /[A-Z]/ and string =~ /a-z/
  puts "string contains mixed case"
end

if string[0..0] =~ /[A-Z]/
  puts "string starts with a capital letter"
end
```

Regular expressions of this sort will only match ASCII characters. To match Unicode uppercase or lowercase characters, use a named character class, as shown here:

```
if string =~ /\p{Upper}/
  puts "string contains uppercase Unicode characters like Ü"
end
```

For more information about regular expressions, see Chapter 3, "Working with Regular Expressions."

2.12 Accessing and Assigning Substrings

In Ruby, substrings may be accessed in several different ways. Normally the bracket notation is used, as for an array, but the brackets may contain a pair of Fixnums, a range, a regex, or a string. Each case is discussed in turn.

If a pair of Fixnum values is specified, they are treated as an offset and a length, and the corresponding substring is returned:

```
str = "Humpty Dumpty"
sub1 = str[7,4]         # "Dump"
sub2 = str[7,99]        # "Dumpty" (overrunning is OK)
sub3 = str[10,-4]       # nil (length is negative)
```

It is important to remember that these are an offset and a length (number of characters), not beginning and ending offsets.

A negative index counts backward from the end of the string. In this case, the index is one based, not zero based, but the length is still added in the forward direction:

```
str1 = "Alice"
sub1 = str1[-3,3]   # "ice"
str2 = "Through the Looking-Glass"
sub3 = str2[-13,4]  # "Look"
```

A range may be specified. In this case, the range is taken as a range of indices into the string. Ranges may have negative numbers, but the numerically lower number must still be first in the range. If the range is "backward" or if the initial value is outside the string, `nil` is returned, as shown here:

```
str = "Winston Churchill"
sub1 = str[8..13]    # "Church"
sub2 = str[-4..-1]   # "hill"
sub3 = str[-1..-4]   # nil
sub4 = str[25..30]   # nil
```

If a regular expression is specified, the string matching that pattern will be returned. If there is no match, `nil` will be returned:

```
str = "Alistair Cooke"
sub1 = str[/l..t/]   # "list"
sub2 = str[/s.*r/]   # "stair"
sub3 = str[/foo/]    # nil
```

If a string is specified, that string will be returned if it appears as a substring (or `nil` if it does not):

```
str = "theater"
sub1 = str["heat"]  # "heat"
sub2 = str["eat"]   # "eat"
```

```
sub3 = str["ate"]   # "ate"
sub4 = str["beat"]  # nil
sub5 = str["cheat"] # nil
```

Finally, in the trivial case, using a `Fixnum` as the index will yield a single character (or `nil` if out of range):

```
str = "Aaron Burr"
ch1 = str[0]     # "A"
ch1 = str[1]     # "a"
ch3 = str[99]    # nil
```

It is important to realize that the notations described here will serve for assigning values as well as for accessing them:

```
str1 = "Humpty Dumpty"
str1[7,4] = "Moriar"     # "Humpty Moriarty"

str2 = "Alice"
str2[-3,3] = "exandra"   # "Alexandra"

str3 = "Through the Looking-Glass"
str3[-13,13]  = "Mirror" # "Through the Mirror"

str4 = "Winston Churchill"
str4[8..13] = "H"        # "Winston Hill"

str5 = "Alistair Cooke"
str5[/e$/] ="ie Monster" # "Alistair Cookie Monster"

str6 = "theater"
str6["er"] = "re"        # "theatre"

str7 = "Aaron Burr"
str7[0] = "B"            # "Baron Burr"
```

Assigning to an expression evaluating to `nil` will have no effect.

2.13 Substituting in Strings

We've already seen how to perform simple substitutions in strings. The sub and gsub methods provide more advanced pattern-based capabilities. There are also sub! and gsub!, their in-place counterparts.

The sub method substitutes the first occurrence of a pattern with the given substitute-string or the given block:

```
s1 = "spam, spam, and eggs"
s2 = s1.sub(/spam/,"bacon")
# "bacon, spam, and eggs"

s3 = s2.sub(/(\w+), (\w+),/,'\2, \1,')
# "spam, bacon, and eggs"

s4 = "Don't forget the spam."
s5 = s4.sub(/spam/) { |m| m.reverse }
# "Don't forget the maps."

s4.sub!(/spam/) { |m| m.reverse }
# s4 is now "Don't forget the maps."
```

As this example shows, the special symbols \1, \2, and so on may be used in a substitute string. However, special variables (such as $& or its English equivalent $MATCH) may not.

If the block form is used, the special variables may be used. However, if all you need is the matched string, it will be passed into the block as a parameter. If it is not needed at all, the parameter can of course be omitted.

The gsub method (global substitution) is essentially the same except that all matches are substituted rather than just the first:

```
s5 = "alfalfa abracadabra"
s6 = s5.gsub(/a[bl]/,"xx")     # "xxfxxfa xxracadxxra"
s5.gsub!(/[lfdbr]/) { |m| m.upcase + "-" }
# s5 is now "aL-F-aL-F-a aB-R-acaD-aB-R-a"
```

The method Regexp.last_match is essentially identical to $& or $MATCH.

2.14 Searching a String

Besides the techniques for accessing substrings, there are other ways of searching within strings. The index method returns the starting location of the specified substring, character, or regex. If the item is not found, the result is nil:

```
str = "Albert Einstein"
pos1 = str.index(?E)        # 7
pos2 = str.index("bert")    # 2
pos3 = str.index(/in/)      # 8
pos4 = str.index(?W)        # nil
pos5 = str.index("bart")    # nil
pos6 = str.index(/wein/)    # nil
```

The method rindex (right index) starts from the right side of the string (that is, from the end). The numbering, however, proceeds from the beginning, as usual:

```
str = "Albert Einstein"
pos1 = str.rindex(?E)       # 7
pos2 = str.rindex("bert")   # 2
pos3 = str.rindex(/in/)     # 13 (finds rightmost match)
pos4 = str.rindex(?W)       # nil
pos5 = str.rindex("bart")   # nil
pos6 = str.rindex(/wein/)   # nil
```

The include? method, shown next, simply tells whether the specified substring or character occurs within the string:

```
str1 = "mathematics"
flag1 = str1.include? ?e         # true
flag2 = str1.include? "math"     # true
str2 = "Daylight Saving Time"
flag3 = str2.include? ?s         # false
flag4 = str2.include? "Savings"  # false
```

The scan method repeatedly scans for occurrences of a pattern. If called without a block, it returns an array. If the pattern has more than one (parenthesized) group, the array will be nested:

```
str1 = "abracadabra"
sub1 = str1.scan(/a./)
# sub1 now is ["ab","ac","ad","ab"]
```

```
str2 = "Acapulco, Mexico"
sub2 = str2.scan(/(.)(c.)/)
# sub2 now is [ ["A","ca"], ["l","co"], ["i","co"] ]
```

If a block is specified, the method passes the successive values to the block, as shown here:

```
str3 = "Kobayashi"
str3.scan(/[^aeiou]+[aeiou]/) do |x|
  print "Syllable: #{x}\n"
end
```

This code produces the following output:

```
Syllable: Ko
Syllable: ba
Syllable: ya
Syllable: shi
```

2.15 Converting Between Characters and ASCII Codes

Single characters in Ruby are returned as one-character strings. Here is an example:

```
str = "Martin"
print str[0]        # "M"
```

The `Integer` class has a method called `chr` that will convert an integer to a character. By default, integers will be interpreted as ASCII, but other encodings may be specified for values greater than 127. The `String` class has an `ord` method that is in effect an inverse:

```
str = 77.chr            # "M"
s2  = 233.chr("UTF-8")  # "é"
num = "M".ord           # 77
```

2.16 Implicit and Explicit Conversion

At first glance, the `to_s` and `to_str` methods seem confusing. They both convert an object into a string representation, don't they?

There are several differences, however. First, *any* object can in principle be converted to some kind of string representation; that is why nearly every core class has a `to_s` method. But the `to_str` method is never implemented in the core.

As a rule, `to_str` is for objects that are really very much like strings—that can "masquerade" as strings. Better yet, think of the short name `to_s` as being *explicit conversion* and the longer name `to_str` as being *implicit conversion.*

You see, the core does not *define* any `to_str` methods. But core methods do *call* `to_str` sometimes (if it exists for a given class).

The first case we might think of is a *subclass* of `String`; but, in reality, any object of a subclass of `String` already "is a" `String`, so `to_str` is unnecessary there.

In real life, `to_s` and `to_str` usually return the same value, but they don't have to do so. The implicit conversion should result in the "real string value" of the object; the explicit conversion can be thought of as a "forced" conversion.

The `puts` method calls an object's `to_s` method in order to find a string representation. This behavior might be thought of as an implicit call of an explicit conversion. The same is true for string interpolation. Here's a crude example:

```
class Helium
  def to_s
    "He"
  end

  def to_str
    "helium"
  end
end

e = Helium.new
print "Element is "
puts e                    # Element is He
puts "Element is " + e    # Element is helium
puts "Element is #{e}"    # Element is He
```

So you can see how defining these appropriately in your own classes can give you a little extra flexibility. But what about honoring the definitions of the objects passed into your methods?

For example, suppose that you have a method that is "supposed" to take a `String` as a parameter. Despite our "duck typing" philosophy, this is frequently done and is often completely appropriate. For example, the first parameter of `File.new` is "expected" to be a string.

The way to handle this is simple. When you expect a string, check for the existence of to_str and call it as needed:

```
def set_title(title)
  if title.respond_to? :to_str
    title = title.to_str
  end
  # ...
end
```

Now, what if an object *doesn't* respond to to_str? We could do several things. We could force a call to to_s, we could check the class to see whether it is a String or a subclass thereof, or we could simply keep going, knowing that if we apply some meaningless operation to this object, we will eventually get an ArgumentError anyway.

A shorter way to do this is

```
title = title.to_str if title.respond_to?(:to_str)
```

which replaces the value of title only if it has a to_str method.

Double-quoted string interpolation will implicitly call to_s, and is usually the easiest way to turn multiple objects into strings at once:

```
e = Helium.new
str = "Pi #{3.14} and element #{e}
# str is now "3.14 and element He"
```

Implicit conversion *would* allow you to make strings and numbers essentially equivalent. You could, for example, do this:

```
class Fixnum
  def to_str
    self.to_s
  end
end

str = "The number is " + 345     # The number is 345
```

However, I don't recommend this sort of thing. There is such a thing as "too much magic"; Ruby, like most languages, considers strings and numbers to be different, and I believe that most conversions should be explicit for the sake of clarity.

There is nothing *magical* about the `to_str` method. It is intended to return a string, but if you code your own, it is your responsibility to see that it does.

2.17 Appending an Item onto a String

The append operator (<<) can be used to append a string onto another string. It is "stackable" in that multiple operations can be performed in sequence on a given receiver:

```
str = "A"
str << [1,2,3].to_s << " " << (3.14).to_s
# str is now "A123 3.14"
```

2.18 Removing Trailing Newlines and Other Characters

Often we want to remove extraneous characters from the end of a string. The prime example is a newline on a string read from input.

The `chop` method removes the last character of the string (typically a trailing newline character). If the character before the newline is a carriage return (`\r`), it will be removed also. The reason for this behavior is the discrepancy between different systems' conceptions of what a newline is. On systems such as UNIX, the newline character is represented internally as a linefeed (`\n`). On others, such as Windows, it is stored as a carriage return followed by a linefeed (`\r\n`):

```
str = gets.chop         # Read string, remove newline
s2 = "Some string\n"    # "Some string" (no newline)
s3 = s2.chop!           # s2 is now "Some string" also
s4 = "Other string\r\n"
s4.chop!                # "Other string" (again no newline)
```

Note that the "in-place" version of the method (`chop!`) will modify its receiver.

It is also important to note that in the absence of a trailing newline, the last character will be removed anyway:

```
str = "abcxyz"
s1 = str.chop           # "abcxy"
```

Because a newline may not always be present, the chomp method may be a better alternative:

```
str = "abcxyz"
str2 = "123\n"
str3 = "123\r"
str4 = "123\r\n"
s1 = str.chomp          # "abcxyz"
s2 = str2.chomp         # "123"
# With the default record separator, \r and \r\n are removed
# as well as \n
s3 = str3.chomp         # "123"
s4 = str4.chomp         # "123"
```

There is also a chomp! method, as we would expect.

If a parameter is specified for chomp, it will remove the set of characters specified from the end of the string rather than the default record separator. Note that if the record separator appears in the middle of the string, it is ignored, as shown here:

```
str1 = "abcxyz"
str2 = "abcxyz"
s1 = str1.chomp("yz")   # "abcx"
s2 = str2.chomp("x")    # "abcxyz"
```

2.19 Trimming Whitespace from a String

The strip method removes whitespace from the beginning and end of a string, whereas its counterpart, strip!, modifies the receiver in place:

```
str1 = "\t  \nabc  \t\n"
str2 = str1.strip         # "abc"
str3 = str1.strip!        # "abc"
# str1 is now "abc" also
```

Whitespace, of course, consists mostly of blanks, tabs, and end-of-line characters.

If we want to remove whitespace only from the beginning or end of a string, we can use the lstrip and rstrip methods:

```
str = "  abc  "
s2 = str.lstrip       # "abc  "
s3 = str.rstrip       # "  abc"
```

There are in-place variants (`rstrip!` and `lstrip!`) also.

2.20 Repeating Strings

In Ruby, the multiplication operator (or method) is overloaded to enable repetition of strings. If a string is multiplied by *n,* the result is *n* copies of the original string concatenated together. Here is an example:

```
etc = "Etc. "*3                       # "Etc. Etc. Etc. "
ruler = "+" + ("."*4+"5"+"."*4+"+")*3
# "+....5....+....5....+....5....+"
```

2.21 Embedding Expressions within Strings

The `#{}` notation makes embedding expressions within strings easy. We need not worry about converting, appending, and concatenating; we can interpolate a variable value or other expression at any point in a string:

```
puts "#{temp_f} Fahrenheit is #{temp_c} Celsius"
puts "The discriminant has the value #{b*b - 4*a*c}."
puts "#{word} is #{word.reverse} spelled backward."
```

Bear in mind that full statements can also be used inside the braces. The last evaluated expression will be the one returned:

```
str = "The answer is #{ def factorial(n)
                          n==0 ? 1 : n*factorial(n-1)
                        end

                        answer = factorial(3) * 7}, of course."
# The answer is 42, of course.
```

There are some shortcuts for global, class, and instance variables, in which case the braces can be dispensed with:

```
puts "$gvar = #$gvar and ivar = #@ivar."
```

Note that this technique is not applicable for single-quoted strings (because their contents are not expanded), but it does work for double-quoted here-documents and regular expressions.

2.22 Delayed Interpolation of Strings

Sometimes we might want to delay the interpolation of values into a string. There is no perfect way to do this.

A naive approach is to store a single-quoted string and then evaluate it:

```
str = '#{name} is my name, and #{nation} is my nation.'
name, nation = "Stephen Dedalus", "Ireland"
s1  = eval('"' + str + '"')
# Stephen Dedalus is my name, and Ireland is my nation.
```

However, using `eval` is almost always the worst option. Any time you use `eval`, you are opening yourself up to many problems, including extremely slow execution and unexpected security vulnerabilities, so it should be avoided if at all possible.

A much less dangerous way is to use a block:

```
str = proc do |name, nation|
 "#{name} is my name, and #{nation} is my nation."
end
s2 = str.call("Gulliver Foyle", "Terra")
# Gulliver Foyle is my name, and Terra is my nation.
```

2.23 Parsing Comma-Separated Data

The use of comma-delimited data is common in computing. It is a kind of "lowest common denominator" of data interchange used (for example) to transfer information between incompatible databases or applications that know no other common format.

We assume here that we have a mixture of strings and numbers and that all strings are enclosed in quotes. We further assume that all characters are escaped as necessary (commas and quotes inside strings, for example).

The problem becomes simple because this data format looks suspiciously like a Ruby array of mixed types. In fact, we can simply add brackets to enclose the whole expression, and we have an array of items:

```
string = gets.chop!
# Suppose we read in a string like this one:
# "Doe, John", 35, 225, "5'10\"", "555-0123"
data = eval("[" + string + "]")   # Convert to array
data.each {|x| puts "Value = #{x}"}
```

This fragment produces the following output:

```
Value = Doe, John
Value = 35
Value = 225
Value = 5' 10"
Value = 555-0123
```

For a more heavy-duty solution, refer to the CSV library (which is a standard library).

2.24 Converting Strings to Numbers (Decimal and Otherwise)

Basically there are two ways to convert strings to numbers: the `Kernel` method `Integer` and `Float` and the `to_i` and `to_f` methods of `String`. (Capitalized method names such as `Integer` are usually reserved for special data conversion functions like this.)

The simple case is trivial, and these are equivalent:

```
x = "123".to_i          # 123
y = Integer("123")      # 123
```

When a string is not a valid number, however, their behaviors differ:

```
x = "junk".to_i         # silently returns 0
y = Integer("junk")     # error
```

`to_i` stops converting when it reaches a non-numeric character, but `Integer` raises an error:

```
x = "123junk".to_i      # 123
y = Integer("123junk")  # error
```

Both allow leading and trailing whitespace:

```
x = " 123 ".to_i        # 123
y = Integer(" 123 ")    # 123
```

Floating point conversion works much the same way:

```
x = "3.1416".to_f      # 3.1416
y = Float("2.718")     # 2.718
```

Both conversion methods honor scientific notation:

```
x = Float("6.02e23")   # 6.02e23
y = "2.9979246e5".to_f # 299792.46
```

`to_i` and `Integer` also differ in how they handle different bases. The default, of course, is decimal or base ten; but we can work in other bases also. (The same is not true for floating point.)

When talking about converting between numeric bases, strings always are involved. After all, an integer is an integer, and they are all stored in binary.

Base conversion, therefore, always means converting to or from some kind of string. Here, we're looking at converting *from* a string. (For the reverse, see Section 5.18, "Performing Base Conversions," and Section 5.5, "Formatting Numbers for Output.")

When a number appears in program text as a literal numeric constant, it may have a "tag" in front of it to indicate base. These tags are 0b for binary, a simple 0 for octal, and 0x for hexadecimal.

These tags are honored by the `Integer` method but *not* by the `to_i` method, as demonstrated here:

```
x = Integer("0b111")     # binary      - returns 7
y = Integer("0111")      # octal       - returns 73
z = Integer("0x111")     # hexadecimal - returns 291

x = "0b111".to_i         # 0
y = "0111".to_i          # 0
z = "0x111".to_i         # 0
```

`to_i`, however, allows an optional parameter to indicate the base. Typically, the only meaningful values are 2, 8, 10 (the default), and 16. However, tags are not recognized even with the base parameter:

```
x = "111".to_i(2)         # 7
y = "111".to_i(8)         # octal       - returns 73
z = "111".to_i(16)        # hexadecimal - returns 291

x = "0b111".to_i          # 0
y = "0111".to_i           # 0
z = "0x111".to_i          # 0
```

Because of the "standard" behavior of these methods, a digit that is inappropriate for the given base will be treated differently:

```
x = "12389".to_i(8)       # 123    (8 is ignored)
y = Integer("012389")     # error  (8 is illegal)
```

Although it might be of limited usefulness, `to_i` handles bases up to 36, using all letters of the alphabet. (This may remind you of the Base64 encoding; for information on that, see Section 2.37, "Encoding and Decoding Base64 Strings.")

```
x = "123".to_i(5)         # 66
y = "ruby".to_i(36)       # 1299022
```

It's also possible to use the `scanf` standard library to convert character strings to numbers. This library adds a `scanf` method to `Kernel`, to `IO`, and to `String`:

```
str = "234 234 234"
x, y, z = str.scanf("%d %o %x")    # 234, 156, 564
```

The `scanf` methods implement all the meaningful functionality of their C counterparts: `scanf`, `sscanf`, and `fscanf`. However, `scanf` does not handle binary.

2.25 Encoding and Decoding `rot13` Text

The `rot13` method is perhaps the weakest form of encryption known to humankind. Its historical use is simply to prevent people from "accidentally" reading a piece of text. It was commonly seen in Usenet posts; for example, a joke that might be considered offensive might be encoded in `rot13`, or you could post the entire plot of *Star Wars: Episode 12* on the day before the premiere.

The encoding method consists simply of "rotating" a string through the alphabet, so that *A* becomes *N*, *B* becomes *O*, and so on. Lowercase letters are rotated in the same way; digits, punctuation, and other characters are ignored. Because 13 is half of

26 (the size of our alphabet), the function is its own inverse; applying it a second time will "decrypt" it.

The following example is an implementation as a method added to the `String` class. We present it without further comment:

```
class String

  def rot13
    self.tr("A-Ma-mN-Zn-z","N-Zn-zA-Ma-m")
  end

end

joke = "Y2K bug"
joke13 = joke.rot13    # "L2X oht"

episode2 = "Fcbvyre: Naanxva qbrfa'g trg xvyyrq."
puts episode2.rot13
```

2.26 Encrypting Strings

There are times when we don't want strings to be immediately legible. For example, passwords should not be stored in plaintext, no matter how tight the file permissions are.

The standard method `crypt` uses the standard function of the same name to DES-encrypt a string. It takes a "salt" value as a parameter (similar to the seed value for a random number generator). On non-UNIX platforms, this parameter may be different.

A trivial application for this follows, where we ask for a password that Tolkien fans should know:

```
coded = "hfCghHIE5LAM."

puts "Speak, friend, and enter!"

print "Password: "
password = gets.chop

if password.crypt("hf") == coded
  puts "Welcome!"
```

```
else
  puts "What are you, an orc?"
end
```

It is worth noting that you should never use encryption to store passwords. Instead, employ *password hashing* using a hashing algorithm designed specifically for passwords, such as bcrypt. Additionally, never rely on encryption of this nature for communications with a server-side web application. To secure web applications, use the HTTPS protocol and Secure Sockets Layer (SSL) to encrypt all traffic. Of course, you could still use encryption on the server side, but for a different reason—to protect the data as it is stored rather than during transmission.

2.27 Compressing Strings

The `Zlib` library provides a way of compressing and decompressing strings and files.

Why might we want to compress strings in this way? Possibly to make database I/O faster, to optimize network usage, or even to obscure stored strings so that they are not easily read.

The `Deflate` and `Inflate` classes have class methods named `deflate` and `inflate`, respectively. The `deflate` method (which obviously compresses) has an extra parameter to specify the style of compression. The styles show a typical trade-off between compression quality and speed; `BEST_COMPRESSION` results in a smaller compressed string, but compression is relatively slow; `BEST_SPEED` compresses faster but does not compress as much. The default (`DEFAULT_COMPRESSION`) is typically somewhere in between in both size and speed.

```
require 'zlib'
include Zlib

long_string = ("abcde"*71 + "defghi"*79 + "ghijkl"*113)*371
# long_string has 559097 characters

s1 = Deflate.deflate(long_string,BEST_SPEED)        # 4188 chars
s2 = Deflate.deflate(long_string)                   # 3568 chars
s3 = Deflate.deflate(long_string,BEST_COMPRESSION)  # 2120 chars
```

Informal experiments suggest that the speeds vary by a factor of two, and the compression amounts vary inversely by the same amount. Speed and compression are greatly dependent on the contents of the string. Speed, of course, also is affected by hardware.

Be aware that there is a "break-even" point below which it is essentially useless to compress a string (unless you are trying to make the string unreadable). Below this point, the overhead of compression may actually result in a *longer* string.

2.28 Counting Characters in Strings

The `count` method counts the number of occurrences of any of a set of specified characters:

```
s1 = "abracadabra"
a  = s1.count("c")      # 1
b  = s1.count("bdr")    # 5
```

The string parameter is like a simple regular expression. If it starts with a caret, the list is negated:

```
c = s1.count("^a")      # 6
d = s1.count("^bdr")    # 6
```

A hyphen indicates a range of characters:

```
e = s1.count("a-d")     # 9
f = s1.count("^a-d")    # 2
```

2.29 Reversing a String

A string may be reversed simply by using the `reverse` method (or its in-place counterpart `reverse!`):

```
s1 = "Star Trek"
s2 = s1.reverse         # "kerT ratS"
s1.reverse!             # s1 is now "kerT ratS"
```

Suppose that you want to reverse the word order (rather than character order). You can use `String#split`, which gives you an array of words. The `Array` class also has a `reverse` method, so you can then reverse the array and join to make a new string:

```
phrase = "Now here's a sentence"
phrase.split(" ").reverse.join(" ") # "sentence a here's Now"
```

2.30 Removing Duplicate Characters

Runs of duplicate characters may be removed using the `squeeze` method. If a parameter is specified, only those characters will be squeezed.

```
s1 = "bookkeeper"
s2 = s1.squeeze         # "bokeper"
s3 = "Hello..."
s4 = s3.squeeze         # "Helo."
```

If a parameter is specified, only those characters will be squeezed.

```
s5 = s3.squeeze(".")    # "Hello."
```

This parameter follows the same rules as the one for the `count` method (see Section 2.28, "Counting Characters in Strings," earlier in this chapter); that is, it understands the hyphen and the caret.

There is also a `squeeze!` method.

2.31 Removing Specific Characters

The `delete` method removes characters from a string if they appear in the list of characters passed as a parameter:

```
s1 = "To be, or not to be"
s2 = s1.delete("b")            # "To e, or not to e"
s3 = "Veni, vidi, vici!"
s4 = s3.delete(",!")           # "Veni vidi vici"
```

This parameter follows the same rules as the one for the `count` method (see Section 2.28, "Counting Characters in Strings," earlier in this chapter); that is, it understands the hyphen and the caret.

There is also a `delete!` method.

2.32 Printing Special Characters

The `dump` method (like `inspect`) provides explicit printable representations of characters that may ordinarily be invisible or print differently. Here is an example:

```
s1 = "Listen" << “\007\007\007” # Add three ASCII BEL characters
puts s1.dump                      # Prints: Listen\007\007\007
s2 = "abc\t\tdef\tghi\n\n"
puts s2.dump                      # Prints: abc\t\tdef\tghi\n\n
s3 = "Double quote: \""
puts s3.dump                      # Prints: Double quote: \"
```

2.33 Generating Successive Strings

On rare occasions, we may want to find the "successor" value for a string; for example, the successor for "aaa" is "aab" (then "aad", "aae", and so on).

Ruby provides the method succ (successor) for this purpose:

```
droid = "R2D2"
improved  = droid.succ         # "R2D3"
pill  = "Vitamin B"
pill2 = pill.succ              # "Vitamin C"
```

We don't recommend the use of this feature unless the values are predictable and reasonable. If you start with a string that is esoteric enough, you will eventually get strange and surprising results.

There is also an upto method that applies succ repeatedly in a loop until the desired final value is reached:

```
"Files, A".upto "Files, X" do |letter|
  puts "Opening: #{letter}"
end

# Produces 24 lines of output
```

Again, we stress that this is not used frequently, and you use it at your own risk. In addition, there is no corresponding "predecessor" function.

2.34 Calculating a 32-Bit CRC

The Cyclic Redundancy Check (CRC) is a well-known way of obtaining a "signature" for a file or other collection of bytes. The CRC has the property that the chance of data being changed and keeping the same CRC is 1 in 2***N*, where *N* is the number of bits in the result (most often 32 bits).

The zlib library, created by Ueno Katsuhiro, enables you to do this.

The method crc32 computes a CRC given a string as a parameter:

```
require 'zlib'
include Zlib
crc = crc32("Hello")            # 4157704578
crc = crc32(" world!",crc)      # 461707669
crc = crc32("Hello world!")     # 461707669 (same as above)
```

A previous CRC can be specified as an optional second parameter; the result will be as if the strings were concatenated and a single CRC was computed. This can be used, for example, to compute the checksum of a file so large that we can only read it in chunks.

2.35 Calculating the SHA-256 Hash of a String

The `Digest::SHA256` class produces a 256-bit *hash* or *message digest* of a string of arbitrary length. This hashing function is one-way, and does not allow for the discovery of the original message from the digest. There are also `MD5`, `SHA384`, and `SHA512` classes inside `Digest` for each of those algorithms.

The most commonly used class method is `hexdigest`, but there are also `digest` and `base64digest`. They all accept a string containing the message and return the digest as a string, as shown here:

```
require 'digest'
Digest::SHA256.hexdigest("foo")[0..20]    # "2c26b46b68f"
Digest::SHA256.base64digest("foo")[0..20] # "LCa0a2j/xo/"
Digest::SHA256.digest(“foo")[0..5]        # ",&\xB4kh\xFF"
```

Although the `digest` method provides a 64-byte string containing the 512-bit digest, the `hexdigest` method is actually the most useful. It provides the digest as an ASCII string of 64 hex characters representing the 64 bytes.

Instances and the `update` method allow the hash to be built incrementally, perhaps because the data is coming from a streaming source:

```
secret = Digest::SHA256.new
source.each { |chunk| secret.update(chunk) }
```

Repeated calls are equivalent to a single call with concatenated arguments:

```
# These two statements...
cryptic.update("Data...")
cryptic.update(" and more data.")

# ...are equivalent to this one.
cryptic.update("Data... and more data.”)

cryptic.hexdigest[0..20] # "50605ba0a90"
```

2.36 Calculating the Levenshtein Distance Between Two Strings

The concept of distance between strings is important in inductive learning (AI), cryptography, proteins research, and in other areas.

The Levenshtein distance is the minimum number of modifications needed to change one string into another, using three basic modification operations: *del*(-etion), *ins*(-ertion), and *sub*(-stitution). A substitution is also considered to be a combination of a deletion and insertion (*indel*).

There are various approaches to this, but we will avoid getting too technical. Suffice it to say that this Ruby implementation (shown in Listing 2.2) allows you to provide optional parameters to set the cost for the three types of modification operations and defaults to a single indel cost basis (cost of insertion = cost of deletion).

Listing 2.2 The Levenshtein distance

```
class String

  def levenshtein(other, ins=2, del=2, sub=1)
    # ins, del, sub are weighted costs
    return nil if self.nil?
    return nil if other.nil?
    dm = []        # distance matrix

    # Initialize first row values
    dm[0] = (0..self.length).collect { |i| i * ins }
    fill = [0] * (self.length - 1)

    # Initialize first column values
    for i in 1..other.length
      dm[i] = [i * del, fill.flatten]
    end

    # populate matrix
    for i in 1..other.length
      for j in 1..self.length
    # critical comparison
        dm[i][j] = [
             dm[i-1][j-1] +
               (self[j-1] == other[i-1] ? 0 : sub),
                 dm[i][j-1] + ins,
             dm[i-1][j] + del
     ].min
      end
    end
    # The last value in matrix is the
    # Levenshtein distance between the strings
    dm[other.length][self.length]
  end

end


s1 = "ACUGAUGUGA"
s2 = "AUGGAA"
d1 = s1.levenshtein(s2)    # 9


s3 = "pennsylvania"
s4 = "pencilvaneya"
d2 = s3.levenshtein(s4)    # 7


s5 = "abcd"
s6 = "abcd"
d3 = s5.levenshtein(s6)    # 0
```

Now that we have the Levenshtein distance defined, it's conceivable that we could define a similar? method, giving it a threshold for similarity. Here is an example:

```
class String

  def similar?(other, thresh=2)
    self.levenshtein(other) < thresh
  end

end


if "polarity".similar?("hilarity")
  puts "Electricity is funny!"
end
```

Of course, it would also be possible to pass in the three weighted costs to the similar? method so that they could in turn be passed into the levenshtein method. We have omitted these for simplicity.

2.37 Encoding and Decoding Base64 Strings

Base64 is frequently used to convert machine-readable data into a text form with no special characters in it. For example, images and fonts stored inline inside CSS files are encoded with Base64.

The easiest way to do a Base64 encode/decode is to use the built-in `Base64` module. The `Base64` class has an `encode64` method that returns a Base64 string (with a newline appended). It also has the method `decode64`, which changes the string back to its original bytes, as shown here:

```
require "base64"
str = "\xAB\xBA\x02abdce"
encoded  = Base64.encode64(str)     # "q7oCYWJkY2U=\n"
original = Base64.decode64(encoded) # "\xAB\xBA\x02abdce"
```

2.38 Expanding and Compressing Tab Characters

Occasionally we have a string with tabs in it and we want to convert them to spaces (or vice versa). The two methods shown here do these operations:

```
class String

  def detab(ts=8)
    str = self.dup
    while (leftmost = str.index("\t")) != nil
      space = " "*(ts-(leftmost%ts))
      str[leftmost]=space
    end
    str
  end

  def entab(ts=8)
    str = self.detab
    areas = str.length/ts
    newstr = ""
    for a in 0..areas
      temp = str[a*ts..a*ts+ts-1]
      if temp.size==ts
        if temp =~ / +/
          match=Regexp.last_match[0]
          endmatch = Regexp.new(match+"$")
          if match.length>1
            temp.sub!(endmatch,"\t")
          end
        end
      end
      newstr += temp
    end
    newstr
  end

end

foo = "This       is        only   a     test.          "

puts foo
puts foo.entab(4)
puts foo.entab(4).dump
```

Note that this code is not smart enough to handle backspaces.

2.39 Wrapping Lines of Text

Occasionally we may want to take long text lines and print them within margins of our own choosing. The code fragment shown here accomplishes this, splitting only on word boundaries and honoring tabs (but not honoring backspaces or preserving tabs):

```
str = <<-EOF
  When in the Course of human events it becomes necessary
  for one people to dissolve the political bands which have
  connected them with another, and to assume among the powers
  of the earth the separate and equal station to which the Laws
  of Nature and of Nature's God entitle them, a decent respect
  for the opinions of mankind requires that they should declare
  the causes which impel them to the separation.
EOF

max = 20

line = 0
out = [""]
```

```
input = str.gsub(/\n/," ")
words = input.split(" ")

while input != ""
  word = words.shift
  break if not word
  if out[line].length + word.length > max
    out[line].squeeze!(" ")
    line += 1
    out[line] = ""
  end
  out[line] << word + " "
end

out.each {|line| puts line}  # Prints 24 very short lines
```

The `ActiveSupport` gem includes similar functionality in a method named `word_wrap`, along with many other string manipulation helpers. Search for it online.

2.40 Conclusion

In this chapter, we have seen the basics of representing strings (both single-quoted strings and double-quoted strings). We've seen how to interpolate expressions into double-quoted strings, and how the double quotes also allow certain special characters to be inserted with escape sequences. We've seen the `%q` and `%Q` forms, which permit us to choose our own delimiters for convenience. Finally, we've seen the here-document syntax, carried over from older contexts such as UNIX shells.

This chapter has demonstrated all the important operations a programmer wants to perform on a string, including concatenation, searching, extracting substrings, tokenizing, and much more. We have seen how to iterate over a string by line or by byte. We have seen how to transform a string to and from a coded form such as Base64 or compressed form.

It's time now to move on to a related topic—regular expressions. Regular expressions are a powerful tool for detecting patterns in strings. We'll cover this topic in the next chapter.

Index

Symbols

& operator, 45
&& operator, 45
* (array expansion operator), 60
** operator, 157
@@ prefix, 45
` (backtick), 11, 533-534
- (set difference) operator, arrays as mathematical sets, 245
-= operator, 52
$0 == __FILE__ idiom, 53
$SAFE global variable and threads, 502
$SAFE levels (security), 430-432
=end markers, syntax issues, 43
== operator
  specialized string comparisons, 70
  testing object equality, 377
=== (threequal) operator, 18, 48, 59, 376
=~ operator, testing object equality, 378
! (exclamation points), syntax issues, 43
!= operator, testing object equality, 378
!~ operator, testing object equality, 378
<< >> (append operator)
  appending arrays, 253
  appending items to strings, 83
  arrays as queues, 254
<=> method
  modules as mixins, 388
  testing object equality, 377
%Q notation and strings, 65
.. range operator, 50
… range operator, 50
| operator, 45
|| operator, 45
|| operator, 16
+ (concatenation) operator, arrays as mathematical sets, 245
+= operator, 52
# (pound sign)
  instance methods and, 51
  strings, 43
  syntax issues, 43
# {} notation, embedding expressions within strings, 85-86
#dup method, 381-382
? (question mark), syntax issues, 43
/m multiline modifier, 58

Numbers

32-bit CRC (Cyclic Redundancy Checksum) calculations in strings, 94-95
2001: A Space Odyssey, 585

A

abstract classes, defining, 7
accessing
  files randomly, 315-316
  methods, controlling access, 378-381

queues, securing queues against illegal access, 297
stacks, securing stacks against illegal access, 293
accessors, 59
accidental object attribute assignments, preventing, 367
ACL (Access Control Lists), drb, 694
actions in Rake utility (development tools), 707-708
ActiveRecord library
databases and, 674-676
online resources, 676
Adams, Ansel, 443
adding items to sets, 289
adjacency matrices, graphs as, 304-307
administration (system)
command-line
ARGF global constant, 539
ARGV global variable, 538
parsing, 540-541
current platform/OS, determining, 554
deleting files based on criteria, 549-550
directory trees, copying, 548-549
distributing Ruby programs, 551
environment variables
retrieving/setting, 545-546
storing as arrays, 546
storing as hashes, 546
Etc module, 554-555
external programs, running
capturing command output, 533-534
exec method, 533
process manipulation, 534-538
system method, 532-533
free space on disks, determining, 550-551
interactivity testing, 553
piping into Ruby interpreter, 552-553
Shell library
file method, 544
fileUtils method, 544
foreach iterator, 544
I/O redirection, 542-543
popdir method, 544
pushdir method, 544
Transact method, 544
text filters, 547-548
age, deleting files by, 549-550
agile language, Ruby as, 8
airline ticket example (Ruby/GTK3 GUI toolkit), 474-477
alert dialog box and button example (Shoes 4 GUI toolkit), 445
alias keyword, 34, 50
all-caps words, matching in regular expressions, 127
all? quantifier and enumerators, 275
allocate method, 384
ancestors (nodes), 298
anchors, regular expressions, 105-106
and-or operators, 45
Anglocentric bias in computer development, 129
Animal Farm, 377
any? quantifier and enumerators, 275
AOP (Aspect-Oriented Programming), 414
API (Application Programming Interface), 686
drawing API (RMagick image manipulation), 576-579
JSON and, 686
REST API, 687
append operator (<< >>)
appending arrays, 253
arrays as queues, 254
items to strings, 83
appending
arrays, 253
files, 315
application layer (networking), 625
applications (web)
asset pipeline
CoffeeScript and JavaScript, 683-685
Sass and CSS, 682-683
data storage
data stores, 676
databases, 674-676
HTML
ERB, 678-679
Haml library, 680
layouts, 677
Liquid templates, 681
Mustache library, 681
partials, 677
templates, 677

    templates and ERB, 677-679
    templates and Haml library, 679-680
    templates and Liquid, 681
    templates and Mustache library, 681
  HTTP servers
    Rack library, 664-666
    simple server example, 662-663
  Rails framework, 667
    asset pipeline, 681-685
    CoffeeScript and JavaScript, 683-685
    ERB and HTML, 678-679
    Haml library and HTML, 680
    parameters, 671-672
    Rails 4 Way, The, 673, 676
    routing, 668-670
    Sass and CSS, 682-683
  Ramaze framework, 667
  Sinatra framework, 666
    parameters, 671
    routing, 668-669
    Sinatra: Up and Running, 673
  static sites
    Jekyll, 690
    Middleman, 689-690
    Nanoc, 690
    Octopress, 690
  web services and HTTP, 686
    JSON for API, 686
    REST API, 687
architecture byte orders, 181
ARGF global constant, 539
arguments, symbols as arguments, 197
ARGV global variable, 538
ARGV [0] command-line parameter, 45
Aristotle, 723
arrays, 11, 231-232
  * (array expansion operator), 60
  appending, 253-254
  associative arrays. *See* hashes
  comparing, 281-282
  concatenating, 253-254
  converting
    enumerators to arrays, 278
    hashes to arrays, 266
    objects to arrays, 389-390
    ranges to arrays, 202
    trees, to strings, 303-304
  counting, 281-282
  creating, 232
  defining, 260
  delimiters, interposing to form strings, 255
  elements
    accessing, 233-234
    assigning, 233-234
    finding elements in one array but not another, 250
    removing duplicate elements, 256
    removing specific elements, 251-252
    selecting from criteria, 240-241
  enumerator objects, 278-280
  enumerators, converting to arrays, 278
  environment variables, storing as arrays, 546
  extracting, 283-284
  hashes
    converting to arrays, 266
    inverting arrays to form hashes, 257
  heterogenous design, 232
  indexing functions, 242-244
  initializing, 232
  inject method, 274-275
  interleaving, 256
  inverting, 257
  iterating, 254-255, 277, 282
  "lazy" arrays, 284
  mapping, 250-251
  mathematical sets, 244-248
  multidimensional arrays, 249-250
  nil values, removing, 251
  objects, converting to arrays, 389-390
  partition method, 276
  quantifiers, 275
  queues, 254, 291
  randomizing, 248-249
  range operators versus, 51
  ranges, converting to arrays, 202
  reversing, 256
  searching, 280-281
  selecting, 280-281
  size of, finding, 235
  sorting, 237-240, 258-259
  space matrices, 244
  stacks, 254, 291
  strings, creating arrays from, 69
  symbol arrays, 27

transforming, 250-251
trees, converting to strings, 303-304
values
counting frequency of, 257
default values for new elements, 259-260
ASCII (American Standard Code for Information Interchange)
ASCII codes, converting characters to in strings, 80
collation, 142
internationalization, 131-134
UTF-8, 134
asctime method, 226
assertions and testing, 591-592
asset pipeline
CoffeeScript and JavaScript, 683-685
Sass and CSS, 682-683
assignment operators, 46
assignments (multiple), 16
associative arrays. *See* hashes
ATK library (Ruby/GTK3 GUI toolkit), 479
Atom feeds, 566
generating, 568-569
parsing, 567-568
attachments, encoding/decoding, 649-651
attr method, creating instance methods, 364
attributes, 59
class attributes, defining, 3
defining, 2
object attributes, defining, 3
OOP attributes
class-level attributes, 368-372
instance attributes, 364-366
preventing accidental assignments, 367
attr_accessor method, 365-366
attr_reader method, 365
attr_writer method, 365

B

Babbage, Charles, 155
back-quotes in command output strings, 11
backreferences and regular expressions, 111-115
backtick (`), 11, 533-534
backward ranges, 202
“bare” scope operator, 53
Barry, Dave, 611
base classes. *See* superclasses
base conversion, 88
Base64 strings, encoding/decoding, 98
Basho, 400
BEGIN keyword, 43
begin keyword, 43
begin-end blocks, 23
big-endians/little-endians, 181
Bigdecimal standard library and large integers, 163-166
Bignum, large integers, 163
binary files, 316-317
binary numbers, 177-179
binary set operators, 289
binary trees
breadth-first insertion, 299
implementing, 298-300
lookup tables, 302-303
sorting data via, 300-302
traversing, 299-300
bit level number operations, 177-179
Black, David, 60
blank spaces, syntax issues, 42
blocks
iterators
calling within blocks, 58
passing blocks to, 54
multiline blocks, 54
single-line blocks, 54
symbols as blocks, 406
syntax issues, 43
Boolean operations, 45
Bouchard, Mathieu, 493
bounding boxes, Prawn and PDF documents, 580
branches, 18
breadth-first insertion, binary trees, 299
break keyword and loops, 22
Britt, James, 723
Browning, Robert, 691
buffered/unbuffered I/O operations, 320-321
bug reports (online resources), 724
built-in classes, 26-28
Bundler
gems, managing
creating gems, 617
Gemfile.lock files, 614

Gemfiles, 614
git dependencies, 616
private gems, 617
requiring gems, 615
running gem commands, 615
semantic versioning, 616
updating gems, 616
buttons
QtRuby GUI toolkit, 481-482
Ruby/GTK3 GUI toolkit, 469-471, 476
Ruby/Tk GUI toolkit, 455-459, 463-466
Shoes 4 GUI toolkit, 445
Swing GUI toolkit, 492-493
Byebug debugging library, 596-599
bytes
byte orders, 181
characters and, 134
defining, 132
file iteration, 337
internationalization, 132, 134

C

C++ and QtRuby GUI toolkit, 490-491
caching mathematic functions via memoization, 190-191
calendars
day of the week, determining, 214
day of the year, finding, 219
Easter, determining the date, 211, 215
Gregorian calendar, 211, 224
Julian calendar, 211
leap years, 221
months, dividing into weeks, 229
Nth weekday, finding in a month, 215-216
number of days in a month, determining, 228
Ruby/Tk GUI toolkit calendar example, 453, 455
weeks
dividing months into, 229
finding week of the year, 220
call stacks, program introspection, 435
capturing constants globally, 427
case comparison (threequel) operator (===),18, 48, 59, 376
case statements, 16-19, 47-50
chaining
directories, 342
methods, 393
characters
bytes and, 134
character classes, regular expressions, 116-118
character data and strings, 63
character encodings, 135
collation, 141-143
encoding conversions, 139-140
normalization, 136-139
transliteration, 141
codepoints, composing/decomposing, 136
commas, formatting numbers with, 162
converting to ASCII codes in strings, 80
defining, 133
diaresis (dieresis), 137
file iteration, 337
grabbing from a keyboard (I/O), 336
internationalization, 133
regular expressions, escaping special characters, 105
strings
counting characters in, 92
printing special characters, 93
removing duplicate characters from strings, 83-84, 93
removing specific characters from strings, 83-84, 93
tab characters, expanding/compressing in strings, 98-99
umlauts, 137
chats (IRC), 725
check boxes
check boxes example (Ruby/Tk GUI toolkit), 463-464
QtRuby GUI toolkit, 486
Ruby/GTK3 GUI toolkit, 476
Chelimsky, David, 586
chess server (peer-to-peer) networking example, 630-637
child classes. *See* subclasses
children (nodes), 298
chomp! method, removing newlines/characters from strings, 84
chomp! operations, 16

chop method, removing newlines/characters from strings, 83
chruby utility, version management, 721
Clark, Jason R., 444
class attributes, defining, 3
class hierarchies (networking), partial inheritance hierarchy, 626
class instance variables, 31, 56, 62, 371
class methods
  class associations, 51
  defining, 3
class variables, 45, 62
class-level attributes (OOP), creating, 368-370, 372
class-level methods (OOP)
  creating, 368-372
  private_class_method, 372
classes
  abstract classes, defining, 7
  built-in classes, 26, 28
  class attributes, defining, 3
  class method associations, 51
  concrete type classes, defining, 7
  creating, 29-32, 34
  data-only classes (structs), creating, 390-391, 399
  defining, 3
  definitions, tracking changes, 435-438
  instances versus, 51
  modules, inclusion, 406-408
  nesting, 399-400
  new method and instatiating new objects, 362
  object classes, testing, 374-377
  open classes, 62
  parametric classes, creating, 400-403
  queue classes, thread synchronization, 515-516
  readers/writers, defining, 412-413
  removing, 427
  retrieving by name, 418
  singleton classes, 61, 396-397
  singletons, 53
  structs (data-only classes), creating, 390-391, 399
  subclasses, 4, 53
  superclasses
    defining, 4
    inheriting from, 372-374
  Symbol class, symbols as blocks, 406
  TracePoint class, monitoring program execution, 439
  variables, 45
classify method and sets, 290
classless (prototype-based) OOP, 414
clear method, removing specific elements from arrays, 253
click event (Shoes 4 GUI toolkit), 451
clients (networking)
  attachments, encoding/decoding, 649-651
  IMAP server interactions, 647-649
  mail-news gateway example, 651-656
  NTP servers, contacting, 641
  Open-URI library, 658
  POP server interactions, 642-643
  random number generator example, 638-641
  SMTP, sending email with, 644-647
  timeservers (official), contacting, 641
  web pages, retrieving from a URL, 657
cliques in graphs, 308
clone method, 381-382
closed/open ranges, 199
closing/opening files, 313-314
closure variable, 55
closures, creating, 54
code
  code coverage tools, 608
  DBC concept, 415
  evaluating dynamically, 416-418
  irb utility (development tools)
    adding code to, 712-713
    xmpfilter library, 714
  poetry mode, 59
  reflection, 38-40
  reusing. *See* inheritance
  runtime coding, 36-38
  source code, viewing with pry utility (development tools), 716
  static code analysis tools, 608
  storing as
    Method objects, 405-406
    Proc objects, 403-405
  thread-safe code, 496
codepoints
  composing/decomposing, 136
  defining, 132

internationalization, 132-136
normalization, 136-138
Unicode codepoints, 133
coefficiants (correlation), 187-189
coercing numeric values, 176-177
CoffeeScript and JavaScript, 683-685
Cohens, Danny, 181
collation, 141-143
collect method, mapping arrays, 250-251
collection searches, thread synchronization, 525-526
comma-separated data, parsing in strings, 86-87
command output strings, 11
command-line
ARGF global constant, 539
ARGV global variable, 538
parameters, 45
parsing, 540-541
commands
pry utility (development tools)
basic commands list, 715
finding help, 716
sending commands, 716
rbenv utility, 720
commas, formatting numbers with, 162
comments, 10
communities (resources)
bug reports, 724
conferences, 725-726
feature requests, 724
forums, 724
IRC, 725
local Ruby groups, 726
mailing lists, 724
podcasts, 724
websites, 723
compact method, removing nil values from arrays, 251
Comparable module and mixins, 388
comparing
arrays, 281-282
enumerators, 281-282
files, 334
floating point numbers, 160-161
hashes, 281-282
strings
case sensitivity, 71
specialized comparisons, 69
time values, 223
compiling regular expressions, 104-105
complex numbers, 171-172
composing/decomposing codepoints, 136
compressing/decompressing strings, 91
compressing/expressing tab characters in strings, 98-99
concatenating
arrays, 253-254
strings, syntax issues, 44
concatenation operator (+), arrays as mathematical sets, 245
concrete type classes, defining, 7
concurrency and threads
$SAFE global variable, 502
creating, 497
accessing thread-local variables, 498-500
changing thread status, 500-503
querying thread status, 500-501
strings, 499
deadlocks, 505
debugging threads, 507-508
defining, 495
disadvantages of, 496
exception-handling, 504-508
exiting threads, 501
fibers and cooperative multitasking, 527-530
grouping threads, 508-509
JRuby and, 496
killing threads, 501
passing threads, 503
performance and, 496
prioritizing threads, 502
race conditions, 496
rendezvous, 505
return values, capturing, 505
Rubinius and, 496
stopping threads, 501
synchronizing threads, 509-510
collection searches in parallel, 525-526
condition variables, 517-518
monitored queues, 520-521
mutexes, 512-514
nested locks, 518-519

    queue classes, 515-516
    recursive deletion in parallel, 526-527
    simple synchronization, 511-512
    timeouts, 522-523
    unsynchronized threads, 496
    waiting for events, 524-525
  thread-safe code, 496
  unsynchronized threads, 496
  uses for, 496
  waking stopped threads, 503
condition variables, thread synchronization, 517-518
conferences, 725-726. *See also* online resources
Confucius, 193
“connected” graphs, 307-308
constants, 51
  capturing globally, 427
  ENV global constant, retrieving/setting environment variables, 545-546
  global constants, ARGF, 539
  nonexistent constants, handling references to, 427-428
  numeric constants, 11
  retrieving by name, 418
constructors (OOP)
  defining, 3
  elaborate constructors, 366-368
  multiple constructors, 362-363
  objects, creating without constructors, 384
const_get method, retrieving classes and constants by name, 418
control characters, internationalization, 131
converting
  base conversion, 88
  characters to ASCII codes in strings, 80
  dates/times, 151
  enumerables to sets, 288
  enumerators to arrays or sets, 278
  epochs, converting to/from, 217
  hashes to arrays, 266
  images via RMagick image manipulation, 572
  numbers
    base conversions, 179-180
    implicit/explicit conversions, 175-176
  objects to
    arrays, 389-390
    printable representations, 390
    strings, 388-389
  ranges to arrays, 202
  seconds into larger units of time, 217
  strings
    implicit/explicit conversions, 80-82
    to numbers, 87-89
    to/from symbols, 197-199
    to_s method, 80-82
    to_str method, 80-82
  symbols, 197-199
  time zones, 227-228
  trees to
    arrays, 303-304
    strings, 303-304
Conway, Damian, 6
cooperative multitasking and fibers, 527-530
coordinates (page), Prawn and PDF documents, 580
copying
  “deep copying”, 346
  directory trees, 548-549
  files, 335
  objects, 381-384
  streams, 339
correlation coefficients, 187-189
count method
  counting characters in strings, 92
  counting frequency of values in arrays, 257
counting
  arrays, 281-282
  enumerators, 281-282
  hashes, 281-282
covector method, vector conversion, 170
coverage tools (code), 608
Cowan, John, 144
CRC (Cyclic Redundancy Checksum) calculations in strings, 94-95
crypt method, string encryption, 90
CSS (Cascading Style Sheets)
  online resources, 682
  Sass and, 682-683
CSV (Comma-Separated Values) data format, 350-352
cube roots, 180
Cucumber Book, The, 596
Cucumber testing tool, 594-596
currencies, formatting (localized) and internationalization, 153

current directory, changing/setting, 341
current root, changing, 342
cursors, Prawn and PDF documents, 580
customizing
  data marshalling, 346-347
  ranges, 206-209
  widgets in QtRuby GUI toolkit, 487-489

D

Dale, Richard, 480
dangling pointers, 41
data formats, 557
  Atom feeds, 566
    generating, 568-569
    parsing, 567-568
  HTML, parsing
    document parsing, 561-564
    stream parsing, 564-566
  JSON, parsing
    libraries, 560-561
    navigating data, 559-560
    non-JSON data types, 560
    objects, 558
  Prawn and PDF documents
    basic concepts/techniques, 579-580
    bounding boxes, 580
    cursors, 580
    document example, 580-583
    margin boxes, 580
    page coordinates, 580
    points (unit of measurement), 580
  RMagick image manipulation, 569
    converting images, 572
    drawing API, 576-579
    resizing images, 572-573
    retrieving image information, 570-572
    special effects/transformations, 573-576
  RSS feeds, 566
    generating, 568-569
    parsing, 567-568
  XML, parsing
    document parsing, 561-564
    stream parsing, 564-566
data hiding, 3
data link layer (networking), 625
data sets
  mean/median/mode, finding, 185-186
  standard deviation, determining, 187
  variance, determining, 187
data storage, 311-312, 673
  CSV data, 350-352
  data stores, 676
  databases, 674-676
  directories
    chaining, 342
    current directory, 341
    current root, 342
    defining, 313
    deleting, 343
    differentiating files from, 326
    finding, 343-344
    iteration, 342
    listing entries, 342
  external data storage, 353
    MySQL databases, 354-356
    PostgreSQL databases, 356-358
  files
    appending, 315
    binary files, 316-317
    command-level manipulation, 334
    comparing, 334
    copying, 335
    copying streams, 339
    defining, 313
    deleting, 334-335
    determining size of, 325
    differentiating directories from, 326
    finding, 343-344
    finding statistics on, 327
    hard links, 327, 334
    installing, 335
    iteration by bytes, 337
    iteration by characters, 337
    iteration by lines, 337
    locking, 318
    moving, 335
    opening/closing, 313-314
    ownership, 321-323
    pathnames, 332-334
    permissions, 321-323
    randomly accessing, 315-316
    reading embedded data, 339

reading into memory, 336
reading program sources, 340
renaming, 334
streams and, 326
strings as files, 338-339
symbolic links, 327, 334
temporary files, 340-341
testing characteristics of, 326
timestamps, 323-324
truncating, 334
updating, 314
verifying existence of, 325
impedence mismatches, 358
marshalling data, 344-345
customizing, 346-347
“deep copying”, 346
YAML, 347-349
ORM, 358-359
persisting data via JSON, 349-350
Redis data stores, 359-360
SQL data storage via SQLite3, 352-353
data stores, 359-360, 676
data-only classes (structs), creating, 390-391, 399
databases, 674-676
dates/times
asctime method, 226
converting, 151
seconds to larger units of time, 217
to/from epochs, 217
current time, determining, 212
Date class, 225
Date standard library, 224, 226
date/time strings
matching in regular expressions, 125-126
parsing, 225-226
DateTime class, 225
Daylight Savings Time, 212
days
days_in_month method, 228
determining day of the week, 214
determining number of days in a month, 228
finding day of the year, 219
Easter, determining the date of, 211, 215
epochs, 212, 217
formatting (localized) and internationalization, 151
GMT, 212-213, 224
Gregorian calendar, 211, 224
hours, working with, 222
Julian calendar, 211
leap seconds, 218
leap years, 221
minutes, working with, 222
mktime method, 213
months
determining number of days in a month, 228
dividing into weeks, 229
finding Nth weekday in a month, 215-216
printing dates, 151
seconds
converting to larger units of time, 217
leap seconds, 218
specific dates (pre-epoch), working with, 224
specific times (post-epoch), handling, 212-214
strftime method, 214, 222, 227
Time class, 225
Time standard library, 226
time values
adding intervals to, 223
comparing, 223
computing the difference between two time values, 224
formatting, 226
printing, 226
time zones, 222, 227-228
UTC, 212-213, 227
validating, 219-220
weeks
dividing months into, 229
finding week of the year, 220
Davis, Ryan, 589
day of the year, finding, 219
Daylight Savings Time, 212
days_in_month method, 228
DBC (Design by Contract) concept, 415
deadlocks (threads), 505
debugging
bug reports (online resources), 724
Byebug debugging library, 596-599
objects, 390

Pry debugging tool, 600-601
Redmine bug tracking system, 724
threads, 507-508
decoding/encoding
attachments, 649-651
Base64 strings, 98
rot13 text, 89
decomposing/composing codepoints, 136
decompressing/compressing strings, 91
"deep copying", 346
deep_copy method, 382
defaults (translations), 146-147
defined entities lists, obtaining, 423-425
define_method method, 419-422
definite integral computations, 182-183
definitions
class definitions, tracking, 435-438
executing, 53
objects definitions, tracking, 435-438
removing
classes, 427
methods, 425-426
deflate method, string compression, 91
degrees (trigonometry), calculating, 183-184
delayed interpolation of strings, 86
delegating method calls, 409-412
delete method
removing specific characters from strings, 93
removing specific elements from arrays, 251
delete_at method, removing specific elements from arrays, 251
delete_if method, removing specific elements from arrays, 252
deleting
directories, 343
files, 334-335
files based on criteria, 549-550
key-value pairs from hashes, 264
recursive deletion and thread synchronization, 526-527
delimiters, forming strings from arrays, 255
dependencies (gems)
Bundler, managing dependencies via
creating gems, 617
Gemfile.lock files, 614
Gemfiles, 614
git dependencies, 616
private gems, 617
requiring gems, 615
running gem commands, 615
semantic versioning, 616
updating gems, 616
git dependencies, 616
installing via Rubygems, 612
dequeue operator, 292
derived classes. *See* subclasses
descendants (nodes), 298
Design Patterns, 695
destructors, defining, 3
detect method, selecting elements from arrays by criteria, 240
detect_hardware method, 368-369
development tools
editor support
Emacs, 718
graphical text editors, 717
Vim, 717-718
irb utility, 710-711
adding code to, 712-713
initializing, 712
lexer capabilities, 714-715
subsessions, 713
tab completion, 712
xmpfilter library, 714
pry utility
basic commands list, 715
documentation, 716
evaluating expressions, 715
finding help, 716
keyboard input, 715
sending commands, 716
shell-mode feature, 716
viewing source code, 716
Rake utility
actions, 707-708
documentation, 710
online resources, 710
Rakefiles, 706-710
tasks, 706
terminology of, 706
uses for, 706
ri utility, 716-717
version management
chruby utility, 721

rbenv utility, 720-721
rvm, 719-720
deviation (standard) of data sets, determining, 187
diamond inheritance problem, defining, 5
diaresis (dieresis), 137
dictionaries. *See* hashes
digest method, SHA-256 hash calculations of strings, 95
digraphs (directed graphs), 304. *See also* graphs
directories
chaining, 342
current directory, 341
defining, 313
deleting, 343
files, differentiating directories from, 326
finding, 343-344
gem directories in Rubygems, 612
iteration, 342
listing entries, 342
roots, changing current root, 342
directory trees, copying, 548-549
disks, determining free space, 550-551
distributing code
Bundler
creating gems, 617
Gemfile.lock files, 614
Gemfiles, 614
git dependencies, 616
private gems, 617
requiring gems, 615
running gem commands, 615
semantic versioning, 616
updating gems, 616
Rubygems
creating gems, 613
directory of gems, 612
installing dependencies, 612
installing gems, 612
distributing code via drb (Distributed Ruby), 692
ACL, 694
components of, 692
DRbObjects, creating, 693
overview of, 692
Rinda
class matches in tuplespaces, 702
creating tuplespaces, 699
defining tuplespaces, 698
development of, 698
examples of tuples, 698
expiring/renewing tuples, 702
nil values as wildcards, 702
notify operations on tuplespaces, 701
read all operations on tuplespaces, 700
read operations on tuplespaces, 699
synchronizing tuplespaces, 700
take operations on tuplespaces, 700
write operations on tuplespaces, 700
Rinda::Ring service discovery, 703-704
security, 693-694
stock ticker simulation case study, 695-698
threaded drb and server joins, 693
distributing Ruby programs, 551
divide method and sets, 290-291
division (numerical calculations), 157
document parsing (XML and HTML), 561-564
documentation
code documentation via RDoc, 618-619
advanced documentation with YARD, 622
simple markup example, 620-622
comments, 10
embedded documentation, 10
online resources, 723
PDF documents and Prawn
basic concepts/techniques, 579-580
bounding boxes, 580
cursors, 580
document example, 580-583
margin boxes, 580
page coordinates, 580
points (unit of measurement), 580
pry utility (development tools), 716
Rake utility (development tools), 710
dot and newline matches in regular expressions, 119
dotted decimal strings, 122
dotted quad strings, 122
double-quoted strings, 11, 64
doubled words, detecting in regular expressions, 126
drawing API (RMagick image manipulation), 576-579
drb (Distributed Ruby)
ACL, 694
components of, 692

DRbObjects, creating, 693
overview of, 692
Rinda
class matches in tuplespaces, 702
creating tuplespaces, 699
defining tuplespaces, 698
development of, 698
examples of tuples, 698
expiring/renewing tuples, 702
nil values as wildcards, 702
notify operations on tuplespaces, 701
read all operations on tuplespaces, 700
read operations on tuplespaces, 699
synchronizing tuplespaces, 700
take operations on tuplespaces, 700
write operations on tuplespaces, 700
Rinda::Ring service discovery, 703-704
security, 693-694
stock ticker simulation case study, 695-698
threaded drb and server joins, 693
duck typing, 60
dump method, printing special characters from strings, 93
dynamic functionality of Ruby
GC, 40
missing methods, 40
reflection, 38-40
runtime coding, 36-38
dynamic OOP (Object-Oriented Programming). *See* OOP
dynamicity (dynamic features)
$SAFE levels, 430-432
classes
removing, 427
retrieving by names, 418
code evaluations, 416-418
constants
capturing globally, 427
handling references to nonexistent constants, 427-428
defined entities lists, obtaining, 423-425
define_method, 419-422
definitions, removing, 425-427
methods
handling calls to nonexistent methods, 429-430
removing, 425-426
objects, defining finalizers, 432-433
retrieving classes and constants by name, 418
security, 430-432
undefining
classes, 427
methods, 425-426

E

Easter, determining the date of, 211, 215
editors
QtRuby GUI toolkit text editor example, 483-484
Ruby/GTK3 GUI toolkit text editor example, 471-473
support for
Emacs, 718
graphical text editors, 717
Vim, 717-718
Eiffel and DBC, 415
eigenclasses. *See* singleton classes
Emacs, 718
email, sending email with SMTP, 644-647
embedded data, reading, 339
embedded documentation, 10
embedded expressions within strings, 85-86
embedded options in regular expressions, 119-120
embedded subexpressions in regular expressions, 120-122
encapsulation, defining, 2
encoding/decoding
attachments, 649-651
Base64 strings, 98
conversions, 139-140
rot13 text, 89
encryption
password hashing, 91
rot13 text, 89
strings, 90
END keyword, 43
end keyword, 43
endpoints (ranges), finding, 200
enqueue operator, 292
ensure clauses, 25

enumerables
  arrays, 231-232
    accssing elements, 233-234
    appending, 253-254
    as queues, 291
    as stacks, 291
    assigning elements, 233-234
    concatenating, 253-254
    creating, 232
    finding elements in one array but not another, 250
    finding size of, 235
    heterogeneous design, 232
    indexing functions, 242-244
    initializing, 232
    interposing delimiters to form strings, 255
    iterating, 254-255
    mapping, 250-251
    mathematical sets, 244-248
    multidimensional arrays, 249-250
    queues, 254
    randomizing, 248-249
    removing nil values, 251
    removing specific elements, 251-252
    selecting elements from criteria, 240-241
    sorting, 237-240
    space matrices, 244
    stacks, 254
    transforming, 250-251
  defining, 273
  Enumerable module and mixins, 388
  hashes, 231
  sets, converting enumerables to, 288
enumerators
  arrays
    comparing, 281-282
    counting, 281-282
    counting frequency of values, 257
    default values for new elements, 259-260
    defining, 260
    enumerator conversion to arrays, 278
    enumerator objects, 278-280
    extracting, 283-284
    hashes and, 257
    inject method, 274-275
    interleaving, 256
    inverting, 257
    iterating, 277, 282
    "lazy" arrays, 284
    partition method, 276
    quantifiers, 275
    removing duplicate elements, 256
    reversing, 256
    searching, 280-281
    selecting, 280-281
    sorting, 258-259
  comparing, 281-282
  converting to arrays or sets, 278
  counting, 281-282
  extracting, 283-284
  hashes
    accessing key-value pairs, 262-263
    adding key-value pairs, 262-263
    comparing, 281-282
    converting to arrays, 266, 278
    counting, 281-282
    creating, 260-261, 268
    creating by inverting arrays, 257
    creating from arrays, 268
    defining, 260
    deleting key-value pairs, 264
    detecting keys and values, 265-266
    enumerator objects, 278-280
    extracting, 283-284
    finding difference/intersection of hash keys, 268
    implementing with duplicate keys, 270-272
    indexing, 273
    inject method, 274-275
    inverting, 265
    inverting arrays to form hashes, 257
    iterating, 264-265, 277, 282
    key values, 273
    keys, 260
    "lazy" hashes, 284
    merging, 268
    partition method, 276
    quantifiers, 275
    searching, 280-281
    selecting, 280-281
    selecting key-value pairs by criteria, 266-267
    sorting, 267
    sparse matrices, 269
    specifying a default value, 261-262

inject method, 274-275
iterating, 277, 282
"lazy" enumerators, 284
objects, 278-280
partition method, 276
quantifiers, 275
searching, 280-281
selecting, 280-281
sets, converting enumerators to sets, 278
symbols as, 195
ENV global constant, retrieving/setting environment variables, 545-546
environment variables
arrays, storing as, 546
hashes, storing as, 546
retrieving/setting, 545-546
epochs, 212, 217
Eppstein, Chris, 683
eql? method, testing object equality, 377-378
equal? method, testing object equality, 377
ERB and HTML generation, 678-679
err variable, 24
Etc module, 554-555
Euler circuits, graphs, 308-309
Euler paths, graphs, 309
EuRuKo (European Ruby Conference), 725
evaluating code dynamically, 416-418
events
Shoes 4 GUI toolkit, 450-451
thread synchronization, 524-525
exception-handling
retry keyword, 53
threads, 504, 506-508
exceptions, 22, 24-25
exclamation points (!), syntax issues, 43
exec method, running external systems, 533
executing programs, monitoring, 439-441
exist? method and files, 325
exit method, process manipulation, 536
exiting threads, 501
expectation expressions, 588
expiring/renewing tuples (Rinda), 702
explicit messages, sending to objects, 394-395
explicit/implicit numeric conversions, 175-176
explicit/implicit string conversions, 80-82
exponentiation, 157
expressing/compressing tab characters in strings, 98-99
expressions
extended regular expressions, 118-119
named matches, 114-115
orientation, 57
regular expressions, 135
Ruby as expression-oriented language, 8
strings, embedding expressions within, 85-86
external data storage, 353
MySQL databases, 354-356
PostgreSQL databases, 356-358
external programs, running
command output, capturing, 533-534
exec method, 533
IO (standard), manipulating, 537-538
processes, manipulating, 534
exit method, 536
fork method, 535-536
Kernel.trap method, 537
kill method, 536
pid method, 536
ppid method, 536
trap method, 537
system method, 532-533
extracting
arrays, 283-284
enumerators, 283-284
hashes, 283-284

F

false values, representing, 45
Faustino, Kevin, 673
fcntl method and I/O, 330
feature requests (online resources), 724
Fernandez, Obie, 673
fibers and cooperative multitasking, 527-530
Fielding, Roy, 687
FIFO (First-In, First-Out) data structures. *See* queues
file formats, 557
Atom feeds, 566
generating, 568-569
parsing, 567-568
HTML, parsing
document parsing, 561-564
stream parsing, 564-566
JSON, parsing

libraries, 560-561
navigating data, 559-560
non-JSON data types, 560
objects, 558
Prawn and PDF documents
basic concepts/techniques, 579-580
bounding boxes, 580
cursors, 580
document example, 580-583
margin boxes, 580
page coordinates, 580
points (unit of measurement), 580
RMagick image manipulation, 569
converting images, 572
drawing API, 576-579
resizing images, 572-573
retrieving image information, 570-572
special effects/transformations, 573-576
RSS feeds, 566
generating, 568-569
parsing, 567-568
XML, parsing
document parsing, 561-564
stream parsing, 564-566
file method, Shell library, 544
fileno method and I/O, 330
files
appending, 315
binary files, 316-317
characteristics of, testing, 326
command-level manipulation, 334
comparing, 334
copying, 335
defining, 313
deleting, 334-335
deleting based on criteria, 549-550
directories, differentiating from files, 326
embedded data, reading, 339
existence of, verifying, 325
finding, 343-344
hard links, 327, 334
installing, 335
iteration, 337
locking, 318
memory, reading files into, 336
moving, 335
opening/closing, 313-314
ownership, 321-323
pathnames, 332-334
permissions, 321-323
program sources, reading, 340
randomly accessing, 315-316
renaming, 334
size of, determining, 325
statistics, finding, 327
streams and, 326, 339
strings as files, 338-339
symbolic links, 327, 334
temporary files, 340-341
timestamps, 323-324
truncating, 334
updating, 314
fileUtils method, Shell library, 544
filters (text), 547-548
finalizers, defining for objects, 432-433
find method, selecting elements from arrays by criteria, 240
finding
directories, 343-344
files, 343-344
find_all method, selecting elements from arrays by criteria, 240
Fixnum, 45
bit-level number operations, 177-179
large integers, 163
flat_map method, arrays as mathematical sets, 248
flip-flop operator and ranges, 203-206
Float method, converting strings to numbers, 87-89
floating point numbers, 156-157
comparing, 160-161
rounding, 158-160
floating point ranges, 201
for construct, 53
for loops, 21, 46
foreach iterator, Shell library, 544
fork method, process manipulation, 534-536
formatting
localized formatting and internationalization
currencies, 153
dates/times, 151
numbers, 152
numbers, 162
strings, 73
time values, 226

forums (online resources), 724
forwarding method calls, 409-412
fourth roots, 180
FOX (Free Objects for X), FXRuby GUI toolkit, 493
frameworks
  Rails, 667
    asset pipeline, 681-685
    CoffeeScript and JavaScript, 683-685
    ERB and HTML, 678-679
    Haml library and HTML, 680
    parameters, 671-672
    Rails 4 Way, The, 673, 676
    routing, 668-670
    Sass and CSS, 682-683
  Ramaze, 667
  Sinatra, 668
    parameters, 671
    routing, 668-669
    Sinatra: Up and Running, 673
Franklin, Benjamin, 705
free space, determining on disks, 550-551
freeze method, 27
freezing
  objects, 391-393
  strings, 392
Friedl, Jeffrey, 104
"fully connected" graphs, determining, 307-308
function composition, 68
functions
  indexing functions and arrays, 242-244
  mathematical functions, caching via memoization, 190-191
FXRuby GUI toolkit, 493
FXRuby: Create Lean and Mean GUIs with Ruby, 494

G

GC (Garbage Collection), 40
GCD (Greatest Common Divisors), 173
GDK library (Ruby/GTK3 GUI toolkit), 479
GdkPixbuf library (Ruby/GTK3 GUI toolkit), 479
gems
  Bundler, managing dependencies via
    creating gems, 617
    Gemfile.lock files, 614
    Gemfiles, 614
    git dependencies, 616
    private gems, 617
    requiring gems, 615
    running gem commands, 615
    semantic versioning, 616
    updating gems, 616
  creating
    Bundler, 617
    Rubygems, 613
  dependencies
    git dependencies, 616
    installing, 612
    managing, 614-617
  Gemfile.lock files in Bundler, 614
  Gemfiles in Bundler, 614
  online resources, 613
  private gems, 617
  Rubygems, packaging/distributing code via
    creating gems, 613
    directory of gems, 612
    installing dependencies, 612
    installing gems, 612
  semantic versioning, 616
  updating, 616
geometry managers, Ruby/Tk GUI toolkit, 453
getters and setters, 56, 59
Gherkin notation language, 594
Gibson, William, 287
Gilbert, W. S., 182
GIMP (GNU Image Manipulation Program), Ruby/GTK3 GUI toolkit, 467
GIO library (Ruby/GTK3 GUI toolkit), 479
git dependencies and gems, 616
Glib library (Ruby/GTK3 GUI toolkit), 479
global constants
  ARGF, 539
  ENV, retrieving/setting environment variables, 545-546
global variables, ARGV, 538
glyphs, internationalization, 132
GMT (Greenwich Mean Time), 212-213, 224

GNOME (GNU Network Object Model Environment), Ruby-GNOME2 project, 479
Golden Gate RubyConf, 726
Gotham RubyConf, 726
grads (trigonometry), calculating, 183-184
grapheme, internationalization, 132
graphical interfaces. *See* GUI toolkits
graphical text editors, 717
graphics
  resizing, 572-573
  RMagick image manipulation, 569
    converting graphics, 572
    drawing API, 576-579
    resizing images, 572-573
    retrieving graphics information, 570-572
    special effects/transformations, 573-576
  Shoes 4 GUI toolkit, 450
graphs, 287. *See also* digraphs (directed graphs)
  adjacency matrices, 304-307
  cliques, 308
  Euler circuits, 308-309
  Euler paths, 309
  fully connected graphs, determining, 307-308
  iteration, 307
  libraries, 310
  unicursive graphs, 308
  vertices, 304, 307
  weighted graphs, 304
grave accents in command output strings, 11
Gray III, James Edward, 350
Gregorian calendar, 211, 224
grep method, selecting elements from arrays by criteria, 240
Gross, David, 63
grouping threads, 508-509
groups (local), Ruby communities, 726
gsub method, string substitutions, 78
GTK. *See* Ruby/GTK3 GUI toolkit
GUI toolkits, 443
  FXRuby GUI toolkit, 493
  QtRuby
    buttons, 481-482
    C++ and, 490-491
    check boxes, 486
    development of, 480
    licensing, 480
    overview of, 480
    radio buttons, 486
    text, 483-484
    widgets, 480-489
    windowed application example, 480-481
  Ruby/GTK3
    airline ticket example, 474-477
    buttons, 469-471, 476
    check boxes, 476
    development of, 467-468
    libraries, 479
    menus, 477-478
    overview of, 467-468
    radio buttons, 476
    scrolling, 473
    strings, 468
    text, 471-473
    widgets, 469-478
    windowed application example, 468-469
  Ruby/Tk, 467
    buttons, 455-459, 463-466
    calendar example, 453, 455
    check boxes example, 463-464
    geometry managers, 453
    images, 458-459, 466
    list boxes, 466
    menus, 466
    overview of, 452-453
    radio buttons example, 465-466
    roots, 452
    scrolling, 466
    telnet client example, 460-463
    text, 459-463
    thermostat example, 456-459
    Tk development, 452
    widgets, 452, 455-466
    windowed application example, 453-455
  Ruby/X11, 493
  RubyMotion, 494
  Shoes 4, 444
    alert dialog box and button example, 445
    buttons, 445
    events, 450-451
    graphics, 450
    images, 450
    installing, 445
    JRuby installation, 444

layouts, 448-449
online resources, 452
popularity of, 452
Quatrain generator example, 449
shapes, 450
sizing window, 445
text, 446-448
Swing
buttons, 492-493
windowed application example, 491-492
Win32API as a GUI, 494

H

Haase, Konstantin, 673
HAL 9000, *2001: A Space Odyssey*, 585
Haml library and HTML generation, 680
Hansson, David Heinemeier, 661
hard links and files, 327, 334
hardware, detect_hardware method, 368-369
Harris, Alan, 673
hashes, 12, 231
arrays
converting hashes into arrays, 266
converting hashes to, 278
creating hashes from arrays, 268
inverting to form hashes, 257
comparing, 281-282
counting, 281-282
creating, 257, 260-261, 268
default value, specifying, 261-262
defining, 260
enumerator objects, 278-280
environment variables, storing as hashes, 546
extracting, 283-284
implementing with duplicate keys, 270-272
indexing, 273
inject method, 274-275
inverting, 265
iterating, 264-265, 277, 282
key-value pairs
accessing, 262-263
adding, 262-263
deleting, 264
selecting by criteria, 266-267
keys, 260
detecting, 265-266
finding difference/intersection of, 268
implementing hashes with duplicate keys, 270-272
values of, 273
"lazy" hashes, 284
merging, 268
partition method, 276
password hashing, 91
quantifiers, 275
searching, 280-281
selecting, 280-281
sorting, 267
sparse matrices, 269
syntax issues, 42
values, detecting, 265-266
heckle testing tool, 608
Hellesøy, Aslak, 596
Hello, world! programs. *See* sample programs
here-documents
strings and, 65-67
whitespace in, 67
heterogeneous design of arrays, 232
hexadecimal numbers, 177-179
hexdigest method, SHA-256 hash calculations of strings, 95
hiding data, 3
hierarchies (class), partial inheritance hierarchy and networking, 626
Hodel, Eric, 618
hours, working with, 222
hover event (Shoes 4 GUI toolkit), 451
HTML (Hypertext Markup Language)
layouts, 677
parsing
document parsing, 561-564
stream parsing, 564-566
partials, 677
templates
ERB and, 678-679
Haml library and, 680
Liquid and, 681
Mustache library and, 681
HTTP (Hypertext Transfer Protocol)
API, 686
JSON and, 686
REST API, 687

HTTP servers
HTTP Responses, 662
Rack library, 664-666
simple server example, 662-663
Net::HTTP library, 686
web services and, 686
JSON for API, 686
REST API, 687
Hunt, Andy, 712
Hunter, Tim, 569

I

i18n
internationalization, 132
string interpretation, 132
translations, 144-145
IBM, 231
IBM Austin, 414
identifiers
rules for, 9
syntax issues, 42
idioms and Rubyisms list, 50-56
if statements, 16-17, 42
if-modifiers, 15
illegal access, securing against
queues, 297
stacks, 293
images
ImageMagick library, 569
resizing, 572-573
RMagick image manipulation, 569
converting images, 572
drawing API, 576-579
resizing images, 572-573
retrieving image information, 570-572
special effects/transformations, 573-576
Ruby/Tk GUI toolkit, 458-459, 466
Shoes 4 GUI toolkit, 450
IMAP servers, 647-649
impedence mismatches. *See* ORM
implicit/explicit numeric conversions, 175-176
implicit/explicit string conversions, 80-82
in operator, 247
include operations and modules, 28
include? method, string searches, 79
included method and modules, 385-386
index method
selecting elements from arrays by criteria, 241
string searches, 79
index variables, modifying, 46
indexing
hashes, 273
indexing functions and arrays, 242-244
inflate method, string compression, 91
inheritance, 53
defining, 4
diamond inheritance problem, 5
inheritance polymorphism, defining, 6
MI, 5, 372
partial inheritance hierarchy, networking, 626
single inheritance with implementation sharing, 7
superclasses, inheriting from, 372-374
initialize method, 53
elaborate (complex) constructors, 366
initializing object attributes, 362
initialize_copy method, 383-384
initializing
arrays, 232
irb utility (development tools), 712
inject method, 274-275
inspect method
converting objects to printable representations, 390
debugging objects, 390
installing
files, 335
JRuby, 444
Shoes 4 GUI toolkit, 445
instances
class instance variables, 31, 56, 62
class versus, 51
instance attributes (OOP), creating, 364-366
instance methods
modules and, 385-387
pound sign and, 51
instance_eval method, preventing accidental object attribute assignments, 367
instance_of? method, testing object classes, 375
instantiated objects, defining, 3
Integer method, converting strings to numbers, 87-89

integers
  definite integral computations, 182-183
  large integers, 163-166
interactivity, testing, 553
interface polymorphism, defining, 6
interleaving arrays, 256
International Ruby Conference (RubyConf), 725
internationalization
  Anglocentric bias in computer development, 129
  ASCII, 131-132
    collation, 142
    UTF-8, 134
  background of, 131-134
  bytes, 132-134
  character encodings, 135
    collation, 141-143
    encoding conversions, 139-140
    normalization, 136-139
    transliteration, 141
  characters, defining, 133
  codepoints, 135
    composing/decomposing codepoints, 136
    defining, 132
    Unicode codepoints, 133
  collation, 141-143
  control characters, 131
  defining, 130
  development of, 131-134
  encoding conversions, 139-140
  glyphs, defining, 132
  grapheme, defining, 132
  i18n, 132
  ISO-8859-1 (Latin-1), 134
  Latin-1 (ISO-8859-1), 134
  localization, defining, 130
  localized formatting
    currencies, 153
    dates/times, 151
    numbers, 152
  Lojban, 144
  m17n, 135
  multibyte characters, 131
  multilingualization, defining, 130
  normalization, 136-139
  precomposed notation, 133
  strings, interpreting, 134
  terminology, 132-134
  translations
    defaults, 146-147
    i18n, 144-145
    interpolation, 148
    keys, 144
    messages, 144
    namespaces, 147
    pluralization, 149-150
    YAML, 144-145
  transliteration, 141
  UCS-2, 133
  Unicode, 131, 133
  UTF-8, 133
    ASCII compatibility, 134
    collation, 142
  UTF-16, 133
  wide characters, 131
interpolation, 86, 148
interpreted language, Ruby as, 8
interpreter, piping into, 552-553
intersections, sets, 288
introspection (program), 433
  call stacks, 435
  classes, tracking definitions, 435-438
  objects
    space, 434
    tracking definitions, 435-438
  program execution, monitoring, 439-441
inverting
  arrays to form hashes, 257
  hashes, 265
I/O (Input/Output), 311-312
  buffered/unbuffered operations, 320-321
  characters, grabbing from a keyboard, 336
  CSV data format, 350-352
  directories
    chaining, 342
    current directory, 341
    current root, 342
    defining, 313
    deleting, 343
    differentiating files from, 326
    finding, 343-344
    iteration, 342
    listing entries, 342

embedded data, reading, 339
external data storage, 353
  MySQL databases, 354-356
  PostgreSQL databases, 356-358
fcntl method, 330
fileno method, 330
files
  appending, 315
  binary files, 316-317
  command-level manipulation, 334
  comparing, 334
  copying, 335
  copying streams, 339
  defining, 313
  deleting, 334-335
  determining size of, 325
  differentiating directories from, 326
  finding, 343-344
  finding statistics on, 327
  hard links, 327, 334
  installing, 335
  iteration by bytes, 337
  iteration by characters, 337
  iteration by lines, 337
  locking, 318
  moving, 335
  opening/closing, 313-314
  ownership, 321-323
  pathnames, 332-334
  permissions, 321-323
  randomly accessing, 315-316
  reading embedded data, 339
  reading into memory, 336
  reading program sources, 340
  renaming, 334
  streams and, 326
  strings as files, 338-339
  symbolic links, 327, 334
  temporary files, 340-341
  testing characteristics of, 326
  timestamps, 323-324
  truncating, 334
  updating, 314
  verifying existence of, 325
impedence mismatches, 358
ioctl (I/O control) method, 330
marshalling data, 344-345
    customizing, 346-347
    "deep copying", 346
    YAML, 347-349
  nonblocking I/O, 330-331
  ORM, 358-359
  pathnames, 332-334
  persisting data via JSON, 349-350
  pipes, 328-329
  program sources, reading, 340
  readpartial method, 331
  redirecting via Shell library, 542-543
  Redis data stores, 359-360
  select method, 330
  simple I/O routines, 318-320
  special operations, 330
  SQL data storage via SQLite3, 352-353
  streams
    copying, 339
    readpartial method, 331
  syscall method, 330
IO (standard), manipulating, 537-538
IO objects, strings as, 74
ioctl (I/O control) method, 330
iOS, RubyMotion GUI toolkit, 494
IP (Internet Protocol)
  IP addresses, matching in regular expressions, 122-123
  networking and, 625
irb (interactive Ruby) utility (development tools), 710-711
  adding code to, 712-713
  initializing, 712
  lexer capabilities, 714-715
  subsessions, 713
  tab completion, 712
  xmpfilter library, 714
IRC (Internet Relay Chats), 725
is_a? method, testing object classes, 375
Ishitsuka, Keiju, 172
ISO-8859-1 (Latin-1), internationalization, 134
iteration
  arrays, 254-255, 282
  directories, 342
  enumerators, 277, 282
  files, 337
  graphs, 307
  hashes, 264-265, 282

iterators, 22
blocks, calling iterators in, 58
chaining together, 68
foreach iterator, Shell library, 544
function composition, 68
passing blocks to, 54
retry keyword, 53
ranges, 200-201
sets, 290
iwanttolearnruby.com website, 724

J

James, Geoffrey, 531
Java and Swing GUI toolkit
buttons, 492-493
windowed application example, 491-492
JavaScript
CoffeeScript and, 683-685
JSON and API, 686
Jekyll static site generator, 690
Johnson, Andrew
lookbehinds, 110
recursion in regular expressions, 121-122
Johnson, Lyle, 493
JRuby
installing, 444
threads and concurrency, 496
JSON (JavaScript Object Notation)
API and, 686
libraries
json standard library, 560
json-stream library, 560
Oj library, 560-561
yajl-ruby library, 561
objects, 558
parsing, 558
libraries, 560-561
navigating data, 559-560
non-JSON data types, 560
persisting data, 349-350
Julian calendar, 211

K

Katsuhiro, Ueno, 94
Kernel#trace_var method, monitoring program execution, 440
Kernel.trap method, process manipulation, 537
keyboard, grabbing characters from (I/O), 336
keypress event (Shoes 4 GUI toolkit), 451
keys (translations), 144
keyword arguments. *See* named parameters
keyword-value pairs, matching in regular expressions, 123
keywords
alias keyword, 34, 50
BEGIN keyword, 43
begin keyword, 43
break keyword and loops, 22
END keyword, 43
end keyword, 43
list of, 9
method names as keywords, 43
next keyword and loops, 22
redo keyword and loops, 22
retry keyword, 53
syntax issues, 43
then keyword, 43
yield keyword, 22, 52
Khayyam, Omar, 218
kill method and process manipulation, 536
killing threads, 501
Kilmer, Alfred Joyce, 298
kind_of? method, testing object classes, 375
Klabnik, Steve, 444
Kosako, K., 110-111

L

Latin-1 (ISO-8859-1), internationalization, 134
Lay of the Last Minstrel, The, 661
layouts
HTML templates, 677
Shoes 4 GUI toolkit, 448-449
“lazy” enumerators, 284
LCM (Least Common Multiples), 173
leap seconds, 218
leave event (Shoes 4 GUI toolkit), 451
length of strings, finding, 67
Levenshtein distance, calculating between two strings, 96-97
lexers, irb utility (development tools), 714-715

libraries
  ActiveRecord library
    databases and, 674-676
    online resources, 676
  Byebug debugging library, 596-599
  graph libraries, 310
  Haml library and HTML generation, 680
  ImageMagick library, 569
  JSON libraries
    json standard library, 560
    json-stream library, 560
    Oj library, 560-561
    yajl-ruby library, 561
  Mustache library and HTML generation, 681
  Net::FTP library, 627
  Net::HTTP library, 686
  Oj library (JSON), 560-561
  Open-URI library, 658
  Open3 library, IO (standard) manipulation, 537-538
  OptionParser library, command-line parsing, 540-541
  pp library, printing readable objects, 606-608
  Rack library, 664-666
  RSS standard library, 569
  Ruby/GTK3 GUI toolkit, 479
  Shell library
    file method, 544
    fileUtils method, 544
    foreach iterator, 544
    I/O redirection, 542-543
    popdir method, 544
    pushdir method, 544
    Transact method, 544
  yajl-ruby library (JSON), 561
licensing QtRuby, 480
LIFO (Last-In, First-Out) data structures. *See* stacks
lightweight processes. *See* threads
line-oriented language, Ruby as, 8
links
  hard links, 327, 334
  symbolic links, 327, 334
Liquid templates and HTML generation, 681
list boxes, Ruby/Tk GUI toolkit, 466
listing
  defined entities, 423-425
  directory entries, 342
little-endians/big-endians, 181
load operations and modules, 29
Lobachevsky, Nikolai, 308
local Ruby groups, 726
local variables versus object attributes, 46
localization, defining, 130
localized formatting and internationalization
  currencies, 153
  dates/times, 151
  numbers, 152
locking files, 318
locks (nested), thread synchronization, 518-519
logarithms, 184-185
Lojban, 144
Lone Star Ruby Conference, 726
lookaheads, 109
lookbehinds, 110-111
lookup tables, binary trees as, 302-303
loops, 45
  break keyword, 22
  for loops, 21, 46
  loop method, 19
  next keyword, 22
  pretest loops, 19
  redo keyword, 22
  until loops, 21-22, 58
  while loops, 21-22, 58
Love's Labours Lost, 495
Lowell, Amy, 101
lowercase/uppercase characters
  regular expressions, controlling in, 75
  strings, controlling in, 74
Lucas, Edouard, 295

M

m17n, internationalization, 135
Mac OS X, RubyMotion GUI toolkit, 494
MacRuby project, 494
Madison+Ruby conference, 726
mail-news gateway network client example, 651-656
mailing lists (online resources), 724
mapping arrays, 250-251
margin boxes, Prawn and PDF documents, 580

marshalling data, 344-345
  customizing, 346-347
  “deep copying”, 346
  YAML, 347-349
Marx, Groucho, 644
Masatoshi, Seki, 692
Mastering Regular Expressions, 104
Math.sqrt function, 180
mathematical functions, caching via memoization, 190-191
mathematical sets
  arrays as, 244-248
  universal sets, 247
mathn library, 172
  prime factorization, 173
  prime numbers, 174-175
matrices, 167-169. *See also* vectors
  adjacency matrices, graphs as, 304-307
  space matrices and arrays, 244
  sparse matrices, hashes as, 269
Matusimoto, Yukihiro, 7
max method, selecting elements from arrays by criteria, 241
McDonald, Ian, 409
mean/median/mode (data sets), finding, 185-186
median/mean/mode (data sets), finding, 185-186
membership tests, sets, 289
memberships (range), testing, 201
memoization, 190-191
memory
  conserving, UTF-8, 134
  files, reading into memory, 336
  leaks, 41
menus
  Ruby/GTK3 GUI toolkit, 477-478
  Ruby/Tk GUI toolkit, 466
merging
  hashes, 268
  items in sets, 289
messages (explicit), sending to objects, 394-395
messages (translations), 144
metaclasses. *See* singleton classes
metavalues, symbols as, 196
Method objects, storing code as, 405-406
methods, 34-35
  accessing, controlling access, 378-381
  asctime method, 226
  chaining, 393
  chomp! method, removing newlines/characters from strings, 84
  chop method, removing newlines/characters from strings, 83
  class methods
    class associations, 51
    defining, 3
  classify method and sets, 290
  clear method, removing specific elements from arrays, 253
  collect method, mapping arrays, 250-251
  compact method, removing nil values from arrays, 251
  controlling access, 378-381
  count method
    counting characters in strings, 92
    counting frequency of values in arrays, 257
  covector method, vector conversion, 170
  days_in_month method, 228
  defining, 2
  deflate method, string compression, 91
  delegating calls, 409-412
  delete method
    removing specific characters from strings, 93
    removing specific elements from arrays, 251
  delete_at method, removing specific elements from arrays, 251
  delete_if method, removing specific elements from arrays, 252
  detect method, selecting elements from arrays by criteria, 240
  digest method, SHA-256 hash calculations of strings, 95
  divide method and sets, 290-291
  dump method, printing special characters from strings, 93
  exec method, running external systems, 533
  exist? method and files, 325
  exit method, process manipulation, 536
  fcntl method and I/O, 330
  file method, Shell library, 544
  fileno method and I/O, 330
  fileUtils method, Shell library, 544

find method, selecting elements from arrays by criteria, 240
find_all method, selecting elements from arrays by criteria, 240
flat_map method, arrays as mathematical sets, 248
Float method, converting strings to numbers, 87-89
fork method, process manipulation, 534-536
forwarding calls, 409-412
freeze method, 27
grep method, selecting elements from arrays by criteria, 240
gsub method, string substitutions, 78
hexdigest method, SHA-256 hash calculations of strings, 95
include? method, string searches, 79
index method
  selecting elements from arrays by criteria, 241
  string searches, 79
inflate method, string compression, 91
initialize methods, 53
inject method, 274-275
instance methods
  modules and, 385, 387
  pound sign and, 51
Integer method, converting strings to numbers, 87-89
ioctl (I/O control) method and I/O, 330
Kernel.trap method, process manipulation, 537
Kernel#trace_var method, monitoring program execution, 440
keywords as method names, 43
kill method, process manipulation, 536
loop method, 19
max method, selecting elements from arrays by criteria, 241
min method, selecting elements from arrays by criteria, 241
missing methods, 40
mktime method, 213
named parameters, 35
naming, 47
nonexistent methods, handling calls to, 429-430
operators as methods, 45
overriding methods, defining, 4
parentheses in method calls, 42
partition method, 276
pid method, process manipulation, 536
pop method, removing specific elements from arrays, 252
popdir method, Shell library, 544
ppid method, process manipulation, 536
printf method, formatting numbers for output, 162
private methods, 378-379
protected methods, 380-381
public methods, 381
push method, appending arrays, 253
pushdir method, Shell library, 544
quadratic method
  Cucumber testing, 594-595
  Minitest testing, 592-594
  RSpec testing, 586-588
readpartial method, I/O and streams, 331
reject method
  removing specific elements from arrays, 252
  selecting elements from arrays by criteria, 241
removing, 425-426
returning expressions, 58
rewind method, randomly accessing files, 316
rindex method, string searches, 79
scan method
  matching strings against a target string, 72
  string searches, 79
searching for, 40
select method and I/O, 330
setter method, 53
shift method, removing specific elements from arrays, 252
shuffle method, randomizing arrays, 248-249
singleton method, 35-36, 61, 396-397
size? method and files, 325
slice! method, removing specific elements from arrays, 252
split method, tokenizing strings, 71-72
squeeze method, removing duplicate characters from strings, 93
strftime method, 214, 222, 227
string method and crypt method, 90

strip method, removing whitespace from strings, 84
strip! method, removing whitespace from strings, 84
sub method, string substitutions, 78
succ method
generating successive strings, 94
prime numbers, 174
symbols and, 197
syntax issues, 42
syscall method and I/O, 330
system method, running external systems, 532-533
top level methods, 53
to_f method, 16, 87-89
to_i method, converting strings to numbers, 87-89
to_s method, 27, 80-82
to_str method, string conversions, 80-82
Transact method, Shell library, 544
trap method, process manipulation, 537
unshift method, appending arrays, 253
update method, SHA-256 hash calculations of strings, 95
upto method, generating successive strings, 94
word_wrap method, wrapping text lines in strings, 100
zero method, matrices, 168
methods (OOP)
class-level methods, 368-370, 372
define_method, 419-422
private_class_method, 372
Meyer, Bertrand, 6
MI (Multiple Inheritance), 5, 372
Middleman static site generator, 688-689
middleware, 665
min method, selecting elements from arrays by criteria, 241
Minitest testing tool, 589-594, 606
minutes, working with, 222
missing methods, 40
mixins, 28-29, 385, 388. *See also* modules
mktime method, 213
mode strings, opening/closing files, 313
mode/mean/median (data sets), finding, 185-186
modifiers, if-modifier, 15
modules. *See also* mixins
defined entities lists, obtaining, 423-425
defining, 7
Etc module, 554-555
include operations, 28
included method and, 385-386
inclusion, 406-408
instance methods and, 385-387
load operations, 29
mixins, modules as, 385, 388
namespace management, 384
nesting, 399-400
require operations, 29
modulus operator, 44
money (currencies), formatting (localized) and internationalization, 153
monitoring
program executions, 439-441
thread synchronization via monitored queues, 520-521
"monkey-patching", 62
months
dividing into weeks, 229
Nth weekday, finding in a month, 215-216
number of days in a month, determining, 228
motion event (Shoes 4 GUI toolkit), 451
MountainWest RubyConf, 726
moving files, 335
multibyte characters, internationalization, 131
multidimensional arrays, 249-250
multiline blocks, 54
multilingualization, defining, 130
multiple assignments, 16
multitasking, fibers and cooperative multitasking, 527-530
Mustache library and HTML generation, 681
mutexes (mutual exclusions), thread synchronization, 512-514
MySQL databases, data storage, 354-356

N

Nakada, Nobu, 409
named matches, 114-115
named parameters, 35
namespaces
in translations, 147
managing via modules, 384

naming
  files, 334
  methods, 47
  variables, 47
Nanoc static site generator, 690
NArray library, 167
negative/positive lookaheads, 109
negative/positive lookbehinds, 110-111
nesting
  classes, 399-400
  modules, 399-400
  nested locks, thread synchronization, 518-519
network layer (networking), 625
networking
  application layer, 625
  clients
    contacting NTP servers, 641
    contacting timeservers (official), 641
    encoding/decoding attachments, 649-651
    IMAP server interactions, 647-649
    mail-news gateway example, 651-656
    Open-URI library, 658
    POP server interactions, 642-643
    retrieving random number generator example, 638-641
    sending email with SMTP, 644-647
    web pages, retrieving from a URL, 657
  data link layer, 625
  IP, 625
  network layer, 625
  Net::FTP library, 627
  Net::Protocol class, 627
  Net::Telnet class, 627
  servers
    contacting NTP servers, 641
    HTTP servers, 662-666
    IMAP server interactions, 647-649
    peer-to-peer chess server example, 630-637
    POP server interactions, 642-643
    simple server example, 627-629
    threaded servers, 629-630
    time of day simple server example, 627-629
  TCP, 626
  transport layer, 625
  UDP, 626
Net::HTTP library, 686
Neukirchen, Christian, 664
new method, classes and instantiating new objects, 362
newlines
  dot matches in regular expressions, 119
  removing from strings, 83-84
news-mail gateway network client example, 651-656
next keyword and loops, 22
NFC (Normalization Form KC), 137
Nicholas, Nick, 144
nil values
  arrays, removing from, 251
  as wildcards in tuplespaces, 702
nil variables, 51
nodes (trees), 298
Nokogiri, XML and HTML parsing
  document parsing, 561-564
  stream parsing, 564-566
nonblocking I/O, 330-331
noncapturing groups, regular expressions, 112
normalization, 136-139
notify operations on tuplespace (Rinda), 701
Nth weekday, finding in a month, 215-216
NTP (Network Time Protocol) servers, contacting, 641
null characters, representing, 45
null set tests, 289
null strings, 45
numbered global variables, 50
numbers
  ** operator, 157
  architecture byte order, 181
  base conversions, 179-180
  big-endians, 181
  Bigdecimal standard library and large integers, 163-166
  Bignum large integers, 163
  binary numbers, 177-179
  bit-level operations, 177-179
  coercing values, 176-177
  comparing, 160-161
  complex numbers, 171-172
  converting strings to numbers, 87-89
  correlation coefficients, 187-189
  data sets
    finding mean/median/mode, 185-186
    standard deviation, 187
    variance, 187

definite integral computations, 182-183
degrees (trigonometry), calculating, 183-184
division, 157
exponentiation, 157
Fixnum
bit-level operations, 177-179
large integers, 163
floating point numbers, 156-157
comparing, 160-161
rounding, 158-160
formatting
for output, 162
with commas, 162
formatting (localized) and internationalization, 152
GCD, 173
grads (trigonometry), calculating, 183-184
hexadecimal numbers, 177-179
implicit/explicit conversions, 175-176
large integers, 163-166
LCM, 173
little-endians, 181
logarithms with arbitrary bases, 184-185
Math.sqrt function, 180
mathematic functions, caching via memoization, 190-191
mathn library, 172
prime factorization, 173
prime numbers, 174-175
matrices, 167-171
memoization, 190-191
octal numbers, 177-179
prime factorization, 173
prime numbers, 174-175
radians (trigonometry), calculating, 183-184
random number generation, 189, 638-641
rational numbers, 166-167
rounding, 158-160
Ruby's representation of, 156
square roots, 180
trigonometry, 183-184
vectors, 170-171
numeric constants, 11, 125

O

object literals, 26
Object#clone method, 381-382
Object-Oriented Perl, 6
objects
attributes
defining, 2-3
local variables versus attributes, 46
preventing accidental assignments, 367
class attributes, defining, 3
class methods, defining, 3
classes
defining, 3
testing, 374-377
constructors, defining, 3
converting to
arrays, 389-390
printable representations, 390
strings, 388-389
copying, 381-384
creating without constructors, 384
debugging, 390
defining, 2
definitions, tracking changes, 435-438
destructors, defining, 3
encapsulation, defining, 2
enumerator objects, 278-280
equality of, testing, 377-378
explicit messages, sending to objects, 394-395
finalizers, defining, 432-433
freezing, 391-393
initialize methods, 53
instantiated objects, defining, 3
JSON objects, 558
Method objects, storing code as, 405-406
methods, defining, 2
object attributes, defining, 3
printing readable objects, 606-608
Proc object, 54, 403-405
Ruby, 26
space and program introspection, 434
specializing an individual object, 396-399
octal numbers, 177-179
Octopress static site generator, 690
Oj library (JSON), 560-561
Old King Cole, 517
Omnibus, distributing Ruby programs, 551

Once in a Lifetime, 435
Onigmo and regular expressions, 101
online resources. *See also* conferences
  ActiveRecord library, 676
  bug reports, 724
  CoffeeScript, 685
  CSS, 682
  feature requests, 724
  forums, 724
  gems, 613
  IRC, 725
  JavaScript, 685
  mailing lists, 724
  Middleman static site generator, 689
  Minitest testing tool, 594
  Nokogiri, 564-566
  podcasts, 724
  Puma web server, 666
  Rake utility (development tools), 710
  RMagick image manipulation, 576, 579
  Ruby-GNOME2 project, 479
  Sass, 683
  Shoes 4 GUI toolkit, 452
  Unicorn web server, 666
  websites, 723
  YARD, 622
OOP (Object-Oriented Programming)
  abstract classes, defining, 7
  allocate method, 384
  AOP, 414
  attributes
    class-level attributes, 3, 368-372
    defining, 2
    instance attributes, 364-366
  blocks, symbols as, 406
  class methods, defining, 3
  classes
    built-in classes, 26, 28
    creating, 29-32, 34
    creating data-only classes (structs), 390-391, 399
    creating structs (data-only classes), 390-391, 399
    defining, 3
    defining class-level readers/writers, 412-413
    inheriting from superclasses, 372-374
    module inclusion, 406-408
    nesting, 399-400
    parametric classes, 400-403
    testing object classes, 374-377
  classless (prototype-based) OOP, 414
  concrete type classes, defining, 7
  constructors
    creating objects without constructors, 384
    defining, 3
    elaborate (complex) constructors, 366-367
    elaborate constructors, 368
    multiple constructors, 362-363
  DBC, 415
  destructors, defining, 3
  diamond inheritance problem, defining, 5
  encapsulation, defining, 2
  inheritance
    defining, 4
    single inheritance with implementation sharing, 7
  inheritance polymorphism, defining, 6
  instantiated objects, defining, 3
  interface polymorphism, defining, 6
  methods
    accessing, 378-381
    chaining, 393
    class-level methods, 368-372
    controlling access, 378-381
    defining, 2
    delegating calls, 409-412
    forwarding calls, 409-412
    private methods, 378-379
    protected methods, 380-381
    public methods, 381
    Ruby, 34-36
  MI, defining, 5
  mixins, Ruby, 28-29
  modules
    defining, 7
    included method and, 385-386
    inclusion, 406-408
    instance methods and, 385-387
    mixins, 385, 388
    namespace management, 384
    nesting, 399-400
    Ruby, 28-29
  named parameters, 35

objects
  attributes, 3
  converting to arrays, 389-390
  converting to printable representations, 390
  converting to strings, 388-389
  copying, 381-384
  debugging, 390
  defining, 2
  freezing, 391-393
  Method objects, 405-406
  Proc objects, 403-405
  Ruby, 26
  sending explicit messages to objects, 394-395
  specializing and individual object, 396-399
  storing code as Method objects, 405-406
  storing code as Proc objects, 403-405
  testing equality of objects, 377-378
  testing object classes, 374-377
overriding methods, defining, 4
parameters, detecting default parameters, 409
polymorphism, defining, 6
prototype-based (classless) OOP, 414
readers/writers (class-level), defining, 412-413
Ruby
  built-in classes, 26-28
  creating classes, 29-34
  methods, 34-36
  mixins, 28-29
  modules, 28-29
  named parameters, 35
  objects, 26
  symbols, 27
  variables, 27
strings, freezing, 392
subclasses, defining, 4
superclasses
  defining, 4
  inheriting from, 372-374
symbols
  blocks as, 406
  Ruby, 27
variables, Ruby, 27
writers/readers (class-level), defining, 412-413
OOPSLA and Ruby, 725
open classes, 62
open/closed ranges, 199
Open-URI library, 658
Open3 library, IO (standard) manipulation, 537-538
opening/closing files, 313-314
operators, 44
  & operator, 45
  && operator, 45
  * (array expansion operator), 60
  ** operator, 157
  == operator, specialized string comparisons, 70
  === (threequal) operator, 18, 48, 59, 376
  << >> (append operator)
    appending arrays, 253
    appending items to strings, 83
    arrays as queues, 254
  - (set difference operator), arrays as mathematical sets, 245
  -= operator, 52
  .. operator, 50
  … range operator, 50
  | operator, 45
  || operator, 16, 45
  + (concatenation operator), arrays as mathematical sets, 245
  += operator, 52
  and-or operators, 45
  assignment operators, 46
  “bare” scope, 53
  binary set operators, 289
  dequeue operator, 292
  enqueue operator, 292
  flip-flop operator and ranges, 203-206
  in operator, 247
  list of, 13-14
  methods, operators as, 45
  modulus operator, 44
  overloading, 33, 52
  pop operator
    arrays as stacks, 254
    stacks and, 292
  push operator
    arrays as stacks, 254
    stacks and, 292
  range operators, 50-51
  reflexive assignment operators, 52
  scope operator, 56
  shift operator, arrays as queues, 254
  ternary decision operator, 58
  unshift operator, arrays as queues, 254

optimizing performance, measuring and, 601-606
OptionParser library, command-line parsing, 540-541
ordinary strings, representing, 64
ORM (Object-Relational Mapper), 358-359
  ActiveRecord library
    databases and, 674-676
    online resource, 676
  data stores and, 676
Orwell, George, 377
OS (current), determining, 554
OSCON (Open Source Convention) and Ruby, 725
overloading operators, 33, 52
overriding methods, defining, 4
ownership of files, 321-323

P

packaging/distributing code
  Bundler
    creating gems, 617
    Gemfile.lock files, 614
    Gemfiles, 614
    git dependencies, 616
    private gems, 617
    requiring gems, 615
    running gem commands, 615
    semantic versioning, 616
    updating gems, 616
  Rubygems
    creating gems, 613
    directory of gems, 612
    installing dependencies, 612
    installing gems, 612
page coordinates, Prawn and PDF documents, 580
Pango library (Ruby/GTK3 GUI toolkit), 479
parameters
  default parameters, detecting, 409
  passing via yield keyword, 22
  Rails framework, 671-672
  Sinatra framework, 671
parametric classes, creating, 400-403
parent classes. *See* superclasses
parentheses in method calls, 42
parents (nodes), 298
Parley forum, 724
parsing
  Atom feeds, 567-568
  comma-separated data in strings, 86-87
  command-line, 540-541
  HTML
    document parsing, 561-564
    stream parsing, 564-566
  JSON, 558
    libraries, 560-561
    navigating data, 559-560
    non-JSON data types, 560
  RSS feeds, 567-568
  time/date strings, 225-226
  XML
    document parsing, 561-564
    stream parsing, 564-566
partial inheritance hierarchy, networking, 626
partials (HTML templates), 677
partition method, 276
passing threads, 503
password hashing, 91
pathnames, 332-334
PDF documents and Prawn
  basic concepts/techniques, 579-580
  bounding boxes, 580
  cursors, 580
  document example, 580-583
  margin boxes, 580
  page coordinates, 580
  points (unit of measurement), 580
peer-to-peer chess server networking example, 630-637
performance
  measuring, 601-606
  threads and, 496
  permissions (files), 321-323
persisting data via JSON, 349-350
Pfeiffer, Tobias, 444
pid (process ID), 534
pid method, process manipulation, 536
pipes, 328-329
piping into Ruby interpreter, 552-553
platforms, determining current platform, 554
plug-ins (Vim), 718
pluralization in translations, 149-150
podcasts (online resources), 724

poetry mode, 59
pointers (dangling), 41
points (unit of measurement), Prawn and PDF documents, 580
Politics, 723
polymorphism, defining, 6
POP (Post Office Protocol) servers, 642-643
pop method, removing specific elements from arrays, 252
pop operator
  arrays as stacks, 254
  stacks and, 292
popdir method, Shell library, 544
Pope, Tim, 718
positive/negative lookaheads, 109
positive/negative lookbehinds, 110-111
PostgreSQL databases, data storage, 356-358
pound sign (#)
  instance methods and, 51
  strings, 43
  syntax issues, 43
pp library, printing readable objects, 606-608
ppid method, process manipulation, 536
Prawn and PDF documents
  basic concepts/techniques, 579-580
  bounding boxes, 580
  cursors, 580
  document example, 580-583
  margin boxes, 580
  page coordinates, 580
  points (unit of measurement), 580
precomposed notation, internationalization, 133
pretest loops, 19
prime factorization of numbers, 173
prime numbers, 174-175
printf method, formatting numbers for output, 162
printing
  dates, 151
  readable objects, 606-608
  special characters in strings, 93
  time values, 226
prioritizing threads, 502
private gems, 617
private methods, 378-379
private_class_method, 372
Proc object, 54, 403-405
processes
  defining, 495
  lightweight processes. *See* threads
  manipulating, 534
    exit method, 536
    fork method, 535-536
    Kernel.trap method, 537
    kill method, 536
    pid method, 536
    ppid method, 536
    trap method, 537
  pid, 534
  Process module, 537
  threads and, 495
program introspection, 433
  call stacks, 435
  classes, tracking definitions, 435-438
  objects
    space, 434
    tracking definitions, 435-438
  program execution, monitoring, 439-441
program sources, reading, 340
programming perspectives in Ruby, 44-47
Programming Ruby, 712
programs
  monitoring executions, 439-441
  sample programs, 14-15
protected method, 367, 380-381
prototype-based (classless) OOP, 414
Pry debugging tool, 600-601
pry utility (development tools)
  basic commands list, 715
  documentation, 716
  evaluating expressions, 715
  help, finding, 716
  keyboard input, 715
  sending commands, 716
  shell-mode feature, 716
  viewing source code, 716
pseudovariables, 44
public methods, 381
Puma web server, 666
punctuation (unbalanced), detecting in stacks, 294
push method, appending arrays, 253
push operator
  arrays as stacks, 254
  stacks and, 292
pushdir method, Shell library, 544

Q

QtRuby GUI toolkit
  buttons, 481-482
  C++ and, 490-491
  check boxes, 486
  development of, 480
  licensing, 480
  overview of, 480
  radio buttons, 486
  text, 483-484
  widgets, 485
    button building example, 482
    check boxes, 486
    customizing widgets, 487-489
    radio buttons, 486
    text editor example, 483-484
    TimerClock custom widget example, 487-489
  windowed application example, 480-481
quadratic method
  Cucumber testing, 594-595
  Minitest testing, 592-594
  RSpec testing, 586-588
quantifiers
  enumerators and, 275
  regular expressions, 106-109
Quatrain generator example (Shoes 4 GUI toolkit), 449
queries (thread status), 500-501
question mark (?), syntax issues, 43
queue classes, thread synchronization, 515-516
queues, 287
  arrays as, 254, 291
  dequeue operator, 292
  enqueue operator, 292
  illegal access, securing against, 297
  thread synchronization and monitored queues, 520-521

R

race conditions and threads, 496
Rack library, 664-666
radians (trigonometry), calculating, 183-184
radio buttons
  QtRuby GUI toolkit, 486
  Ruby/GTK3 GUI toolkit, 476
radio buttons example (Ruby/Tk GUI toolkit), 465-466
Rails, 667
  asset pipeline, 681-685
  CoffeeScript and JavaScript, 683-685
  HTML
    ERB and, 678-679
    Haml library and, 680
  parameters, 671-672
  Rails 4 Way, The, 673, 676
  routing, 668-670
  Sass and CSS, 682-683
RailsConf, 725
raise statements, exceptions, 23
Rake utility (development tools)
  actions, 707-708
  command-line options, 710
  documentation, 710
  online resources, 710
  Rakefiles, 706-709
  tasks, 706
  terminology of, 706
  uses for, 706
Ramaze, 667
random number generation, 189, 638-641
randomizing arrays, 248-249
randomly accessing files, 315-316
range operators, 50-51
ranges
  backward ranges, 202
  converting to arrays, 202
  custom ranges, 206-209
  defining, 193, 199
  endpoints, finding, 200
  flip-flop operator and, 203-206
  floating point ranges, 201
  iterating over, 200-201
  memberships, testing, 201
  open/closed ranges, 199
rational numbers, 166-167
rbenv utility, version management, 720-721
RDoc, documenting code via, 618-619
  advanced documentation with YARD, 622
  simple markup example, 620-622
rdoc.info website, 723
read all operations on tuplespace (Rinda), 700
read operations on tuplespace (Rinda), 699

readers/writers (class-level), defining, 412-413
reading
  embedded data, 339
  files into memory, 336
  program sources, 340
readpartial method, I/O and streams, 331
recursion
  regular expressions, 121-122
  stacks, 295-296
recursive deletion, thread synchronization, 526-527
Redis data stores, 359-360
Redmine bug tracking system, 724
redo keyword and loops, 22
reflection, 38-40
reflexive assignment operators, 52
regular expressions, 11, 16, 101, 127-128, 135
  anchors, 105-106
  backreferences, 111-115
  characters
    character classes, 116-118
    escaping special characters, 105
  compiling, 104-105
  doubled words, detecting in text, 126
  embedded options, 119-120
  embedded subexpressions, 120-122
  extended regular expressions, 118-119
  lookaheads, 109
  lookbehinds, 110-111
  matching
    all-caps words, 127
    date/time strings, 125-126
    IP addresses, 122-123
    keyword-value pairs, 123
    numeric constants, 125
    Roman numerals, 124
    version numbers, 127
  named matches, 114-115
  newline and dot matches, 119
  noncapturing groups, 112
  Onigmo engine, 101
  quantifiers, 106-109
  recursion in, 121-122
  symbols/notations list, 102, 104
  syntax of, 102, 104
  uppercase/lowercase characters, controlling, 75
  zero-length matches in, 58
Rehn, Arno, 480
reject method
  removing specific elements from arrays, 252
  selecting elements from arrays by criteria, 241
relationship operator. *See* threequal operator (===)
relationship tests, sets, 289
release event (Shoes 4 GUI toolkit), 451
removing
  characters from strings, 83-84, 93
  classes, 427
  duplicate elements from arrays, 256
  methods, 425-426
  newlines from strings, 83-84
  nil values from arrays, 251
  specific elements from arrays, 251-252
  whitespace from strings, 84
renaming files, 334
rendezvous (threads), 505
renewing expired tuples (Rindas), 702
repeating strings, 85
REPL (Read-Eval-Print-Loop) tools. *See* Pry debugging tool
requesting features (online resources), 724
require operations and modules, 29
rescue clauses, 23
resizing images, 572-573
resources (online). *See also* conferences
  ActiveRecord library, 676
  bug reports, 724
  CoffeeScript, 685
  CSS, 682
  feature requests, 724
  forums, 724
  gems, 613
  IRC, 725
  JavaScript, 685
  mailing lists, 724
  Middleman static site generator, 689
  Minitest testing tool, 594
  Nokogiri, 564, 566
  podcasts, 724
  Puma web server, 666
  Rake utility (development tools), 710
  RMagick image manipulation, 576, 579
  Ruby-GNOME2 project, 479
  Sass, 683

Shoes 4 GUI toolkit, 452
Unicorn web server, 666
websites, 723
YARD, 622
respond_to? method, testing object classes, 376
REST (Representational State Transfer) API, 687
retrieving
classes by name, 418
constants by name, 418
retry keyword, 53
return values in threads, capturing, 505
reusing code. *See* inheritance
reversing
arrays, 256
strings, 92
rewind method, randomly accessing files, 316
ri (Ruby Index) utility (development tools), 716-717
Rinda
development of, 698
Rinda::Ring and service discovery, 703-704
tuples, examples of, 698
tuplespace
class matches, 702
creating, 699
defining, 698
expiring/renewing tuples, 702
nil values as wildcards, 702
notify operations, 701
read all operations, 700
read operations, 699
synchronization, 700
take operations, 700
write operations, 700
rindex (right index) method, string searches, 79
RMagick image manipulation, 569
drawing API, 576-579
image conversions, 572
image information, retrieving, 570-572
resizing images, 572-573
special effects/transformations, 573-576
Roman numerals, matching in regular expressions, 124
roots, 298
cube roots, 180
current root, changing, 342
fourth roots, 180
Ruby/Tk GUI toolkit, 452
rot13 text, encoding/decoding, 89
rounding floating point numbers, 158-160
routing
Rails framework, 669-671
Sinatra framework, 668-669
RSpec testing tool, 586-589, 606
RSS feeds, 566
generating, 568-569
parsing, 567-568
Rubaiyat, The, 218
Rubinius, threads and concurrency, 496
rubocop consistent styling tool, 609
Ruby
agile language, Ruby as, 8
arrays, 11
attributes of, 8
blocks, 43
classes
built-in classes, 26, 28
creating, 29-32, 34
comments, 10
constants (numeric), 11
distributing programs, 551
dynamic functionality
GC, 40
missing methods, 40
reflection, 38-40
runtime coding, 36-38
embedded documentation, 10
exceptions, 22-25
expression-oriented language, Ruby as, 8
GC, 40
hashes, 12
identifiers, rules for, 9
interpreted language, Ruby as, 8
iterators, 22
keywords list, 9
line-oriented language, Ruby as, 8
methods, 34-36, 40
mixins, 28-29
modules, 28-29
named parameters, 35
objects, 26
OOP
built-in classes, 26, 28
creating classes, 29-32, 34

methods, 34-36
mixins, 28-29
modules, 28-29
named parameters, 35
objects, 26
symbols, 27
variables, 27
operators list, 13-14
programming perspectives, 44-47
pseudovariables, 44
reflection, 38-40
regular expressions, 11, 16
runtime coding, 36-38
sample program, 14-15
strings, 11
symbols, 27
syntax issues, 41-44
syntax sugar, 12
variables, 11, 27
pseudovariables, 44
rules for, 9
VHLL, Ruby as, 8
Ruby Central conference, 725
Ruby interpreter, piping into, 552-553
Ruby Rogues podcast, 724
Ruby Weekly mailing list, 724
ruby-doc.org website, 723
Ruby-GNOME2 project, 479
ruby-lang.org website, 723
ruby-talk mailing list, 724
Ruby/ATK library (Ruby/GTK3 GUI toolkit), 479
Ruby/GDK library (Ruby/GTK3 GUI toolkit), 479
Ruby/GdkPixbuf library (Ruby/GTK3 GUI toolkit), 479
Ruby/GIO library (Ruby/GTK3 GUI toolkit), 479
Ruby/Glib library (Ruby/GTK3 GUI toolkit), 479
Ruby/GTK library (Ruby/GTK3 GUI toolkit), 479
Ruby/GTK3 GUI toolkit
airline ticket example, 474-477
buttons, 469-471, 476
check boxes, 476
development of, 467-468
libraries, 479
menus, 477-478
overview of, 467-468
radio buttons, 476
scrolling, 473
strings, 468
text, 471-473
widgets
airline ticket example, 474-477
button building example, 471
menus, 478
text editor example, 472-473
windowed application example, 469
windowed application example, 468-469
Ruby/Pango library (Ruby/GTK3 GUI toolkit), 479
Ruby/Tk GUI toolkit, 467
buttons, 455-459, 463-466
calendar example, 453-455
check boxes example, 463-464
geometry managers, 453
images, 458-459, 466
list boxes, 466
menus, 466
overview of, 452-453
radio buttons example, 465-466
roots, 452
scrolling, 466
telnet client example, 460-463
text, 459-463
thermostat example, 456-459
Tk development, 452
widgets, 452
button building example, 455-459
check boxes example, 463-464
images, 466
list boxes, 466
menus, 466
radio buttons example, 465-466
scrolling, 466
text example, 459-463
windowed application example, 453, 455
Ruby/X11 GUI toolkit, 493
RubyConf (International Ruby Conference), 725
RubyConf Brasil, 725

Rubygems
  creating, 613
  dependencies, 612
  directory of gems, 612
  installing, 612
Rubyisms and idioms list, 50-56
RubyMine graphical text editor, 717
RubyMotion GUI toolkit, 494
runtime coding, 36-38
Russell, Bertrand, 396
rvm (Ruby Version Manager), 719-720

S

sample programs, 14-15
Sansonetti, Laurent, 494
Sass and CSS, 682-683
scan method
  matching strings against a target string, 72
  string searches, 79
scope operator, 56
Scott, Sir Walter, 661
scripts
  command-line
    ARGF global constant, 539
    ARGV global variable, 538
    parsing, 540-541
  directory trees, copying, 548-549
  disks, determining free space on, 550-551
  environment variables
    retrieving/setting, 545-546
    storing as arrays, 546
    storing as hashes, 546
  Etc module, 554-555
  external programs, running
    capturing command output, 533-534
    exec method, 533
    IO (standard) manipulation, 537-538
    process manipulation, 534-537
    system method, 532-533
  files, deleting based on criteria, 549-550
  interactivity testing, 553
  OS (current), determining, 554
  platforms (current), determining, 554
  Ruby interpreter, piping into, 552-553
  Ruby programs, distributing, 551
  Shell library
    file method, 544
    fileUtils method, 544
    foreach iterator, 544
    I/O redirection, 542-543
    popdir method, 544
    pushdir method, 544
    Transact method, 544
  system administration
    ARGF global constant, 539
    ARGV global variable, 538
    copying directory trees, 548-549
    deleting files based on criteria, 549-550
    determining current platform/OS, 554
    determining free space on disks, 550-551
    distributing Ruby programs, 551
    environment variables, 545-546
    Etc module, 554-555
    interactivity testing, 553
    parsing command-line, 540-541
    piping into Ruby interpreter, 552-553
    running external programs, 532-538
    Shell library, 542-544
    text filters, 547-548
scrolling
  Ruby/GTK3 GUI toolkit, 473
  Ruby/Tk GUI toolkit, 466
SCSS (Syntactically Awesome Style Sheets) and Sass, 682-683
searches
  arrays, 280-281
  collection searches and thread synchronization, 525-526
  enumerators, 280-281
  hashes, 280-281
  strings, 79-80
seconds
  converting to larger units of time, 217
  leap seconds, 218
security
  $SAFE levels, 430-432
  drb, 693-694
  dynamicity (dynamic features), 430-432
  illegal access, preventing in
    queues, 297
    stacks, 293
  taint levels, 430-432

select method and I/O, 330
selecting
  arrays, 280-281
  enumerators, 280-281
  hashes, 280-281
semantic versioning and gems, 616
servers (networking)
  HTTP servers
    HTTP Responses, 662
    Rack library, 664-666
    simple server example, 662-663
  IMAP server interactions, 647-649
  NTP server, contacting, 641
  peer-to-peer chess server example, 630-637
  POP server interactions, 642-643
  simple server example, 627-629
  threaded servers, implementing, 629-630
  time of day simple server example, 627-629
  web servers
    Puma web server, 666
    Rack library and, 664-666
    Unicorn web server, 666
service discovery, drb and Rinda::Ring, 703-704
sessions, subsessions in irb utility (development tools), 713
set difference operator (-), arrays as mathematical sets, 245
sets, 287
  adding items to, 289
  binary set operators, 289
  classify method, 290
  converting enumerators to sets, 278
  creating, 288
  divide method, 290-291
  enumerables, converting to sets, 288
  intersections, 288
  iteration, 290
  membership tests, 289
  merging items in, 289
  null set tests, 289
  relationship tests, 289
  unions, 288
setter and getter actions, 56, 59
setter methods, 53
Shakespeare, William, 361, 495
shapes, Shoes 4 GUI toolkit, 450
Shell library
  file method, 544
  fileUtils method, 544
  foreach iterator, 544
  I/O redirection, 542-543
  popdir method, 544
  pushdir method, 544
  Transact method, 544
shell-mode (pry utility), 716
shift method, removing specific elements from arrays, 252
shift operator, arrays as queues, 254
Shoes 4 GUI toolkit
  alert dialog box and button example, 445
  buttons, 445
  events, 450-451
  graphics, 450
  images, 450
  installing, 445
  JRuby installation, 444
  layouts, 448-449
  online resources, 452
  popularity of, 452
  Quatrain generator example, 449
  shapes, 450
  text, 446-448
  window, sizing, 445
shuffle method, randomizing arrays, 248-249
Silicon Valley Ruby Conference, 726
simple-cov code coveraging tool, 608
Sinatra, 668
  parameters, 671
  routing, 668-669
  Sinatra: Up and Running, 673
single inheritance with implementation sharing, 7
single-line blocks, 54
single-quoted strings, 11, 64
singleton classes, 61, 396-397
singleton methods, 61, 396-397
Singleton Pattern, 61
singletons, 35-36, 53, 61
size? method and files, 325
sizing images, 572-573
slice! method, removing specific elements from arrays, 252

SMTP (Simple Mail Transfer Protocol), sending email with, 644-647

Snow Crash, 557

Sonnet 113, 361

sorting

- arrays, 237-240, 258-259
- data via binary trees, 300-302
- hashes, 267

source code, viewing with pry utility (development tools), 716

sources (program), reading, 340

sparse matrices

- arrays and, 244
- hashes as, 269

special characters, escaping in regular expressions, 105

special effects/transformations in images, 573-576

special variables, 50

specialized string comparisons, 69

splat operator. *See* array expansion operator (*)

split method, tokenizing strings, 71-72

SQL (Structured Query Language), data storage, 352-353

square roots, 180

squeeze method, removing duplicate characters from strings, 93

stacks, 287

- arrays as, 254, 291
- illegal access, securing against, 293
- pop operator, 292
- program introspection, 435
- push operator, 292
- recursion, 295-296
- unbalanced punctuation, detecting, 294

standard deviation (data sets), determining, 187

star operator. *See* array expansion operator (*)

statements

- case statements, 16-19, 47-50
- if statements, 16-17, 42
- raise statements, exceptions, 23
- unless statements, 17

static code analysis tools, 608

static sites

- Jekyll, 690
- Middleman, 689
- Nanoc, 690
- Octopress, 690

Steel, Jr., Thomas, 311

step definitions and Cucumber testing, 595

Stephenson, Neal, 557

stock ticker simulation case study and drb, 695-698

stopping threads, 501

storing code

- Method objects, 405-406
- Proc objects, 403-405

storing data, 311-312

- CSV data, 350-352
- data stores, 676
- databases, 674-676
- directories
  - chaining, 342
  - current directory, 341
  - current root, 342
  - defining, 313
  - deleting, 343
  - differentiating files from, 326
  - finding, 343-344
  - iteration, 342
  - listing entries, 342
- external data storage, 353
  - MySQL databases, 354-356
  - PostgreSQL databases, 356-358
- files
  - appending, 315
  - binary files, 316-317
  - command-level manipulation, 334
  - comparing, 334
  - copying, 335
  - copying streams, 339
  - defining, 313
  - deleting, 334-335
  - determining size of, 325
  - differentiating directories from, 326
  - finding, 343-344
  - finding statistics on, 327
  - hard links, 327, 334
  - installing, 335
  - iteration by bytes, 337
  - iteration by characters, 337
  - iteration by lines, 337
  - locking, 318
  - moving, 335
  - opening/closing, 313-314
  - ownership, 321-323

pathnames, 332-334
permissions, 321-323
randomly accessing, 315-316
reading embedded data, 339
reading into memory, 336
reading program sources, 340
renaming, 334
streams and, 326
strings as files, 338-339
symbolic links, 327, 334
temporary files, 340-341
testing characteristics of, 326
timestamps, 323-324
truncating, 334
updating, 314
verifying existence of, 325
impedence mismatches, 358
marshalling data, 344-345
customizing, 346-347
"deep copying", 346
YAML, 347-349
ORM, 358-359
persisting data via JSON, 349-350
Redis data stores, 359-360
SQL data storage via SQLite3, 352-353
streams
copying, 339
files and, 326
I/O and streams, readpartial method, 331
parsing (XML and HTML), 564-566
strftime method, 214, 222, 227
strings
%q notation, 65
32-bit CRC calculations, 94-95
appending items to strings, 83
arrays
creating from strings, 69
interposing delimiters to form strings, 255
Base64 strings, encoding/decoding, 98
characters
counting, 92
data and, 63
printing special characters, 93
removing duplicate characters, 93
removing specific characters, 93
chomp! operations, 16
comma-separated data, parsing, 86-87
command output strings, 11
comparing
case sensitivity, 71
specialized comparisons, 69
compressing/decompressing, 91
concatenating strings, syntax issues, 44
converting
characters to ASCII codes, 80
implicit/explicit conversions, 80-82
to numbers, 87-89
to_s method, 80-82
to_str method, 80-82
date/time strings, matching in regular expressions, 125-126
dotted decimal strings, 122
dotted quad strings, 122
double-quoted strings, 11, 64
embedding expressions within, 85-86
encryption, 90
files as strings, 338-339
formatting, 73
freezing, 392
here-documents, 65-67
i18n interpretation, 132
internationalization, 134
interpolating (delayed), 86
interpreting, 134
IO objects, strings as, 74
length of, finding, 67
Levenshtein distance, calculating between two strings, 96-97
lookaheads, 109
lookbehinds, 110-111
mode strings, opening/closing files, 313
null strings, 45
objects, converting to strings, 388-389
one-character strings, creating, 69
ordinary strings, representing, 64
pound signs in, 43
printing, special characters, 93
processing
one character (byte) at a time, 68-69
one line at a time, 68
removing
duplicate characters, 93
specific characters, 93
trailing newlines/characters, 83-84
whitespace, 84

repeating, 85
reversing, 92
rot13 text, encoding/decoding, 89
Ruby/GTK3 GUI toolkit, 468
scanning, 72
searching, 79-80
single-quoted strings, 11, 64
substitutions in, 78
substrings
accessing, 75-77
assigning, 77
successive strings, generating, 94
symbols, 27, 197-199
syntax issues, 43
tab characters, expanding/compressing, 98-99
text lines, wrapping, 99-100
threads and, 499
time/date strings, parsing, 225-226
tokenizing, 71-72
trees, converting to strings, 303-304
uppercase/lowercase characters, controlling, 74
whitespace, removing, 84
strip method, removing whitespace from strings, 84
strip! method, removing whitespace from strings, 84
Stroustrup, Bjarne, 6-7
structs (data-only classes), creating, 390-391, 399
style, rubocop consistent styling tool, 609
sub method, string substitutions, 78
subclasses, 4, 53
subexpressions, embedded subexpressions in regular expressions, 120-122
SublimeText 3 graphical text editor, 717
subscripted variables. *See* arrays
subsessions, irb utility (development tools), 713
substitutions (string), 78
substrings
accessing, 75-77
assigning, 77
subtrees, 298
succ (successor) method
prime numbers, 174
successive strings, generating, 94
superclass method, testing object classes, 377
superclasses
defining, 4
inheriting from, 372-374
swapping variables, 51
Swing GUI toolkit
buttons, 492-493
windowed application example, 491-492
Symbol class, symbols as blocks, 406
symbolic links and files, 327, 334
symbols
arguments, symbols as, 197
arrays, 27
built-in classes, 27
class attributes, defining, 197
converting to/from, 197-199
defining, 193-195
enumerations, symbols as, 195
metavalues, symbols as, 196
methods and, 197
names and, 194
strings, 27
syntax, 194
variables and, 197
synchronizing
array sorting, 258-259
threads, 509-510
collection searches in parallel, 525-526
condition variables, 517-518
monitored queues, 520-521
mutexes, 512-514
nested locks, 518-519
queue classes, 515-516
recursive deletion in parallel, 526-527
simple synchronization, 511-512
timeouts, 522-523
unsynchronized threads, 496
waiting for events, 524-525
tuplespace (Rinda), 700
syntax
Ruby syntax issues, 41-44
syntax sugar, 12, 52
syscall method and I/O, 330
system administration
command-line
ARGF global constant, 539
ARGV global variable, 538
parsing, 540-541

deleting files based on criteria, 549-550
determining current platform/OS, 554
determining free space on disks, 550-551
directory trees, copying, 548-549
distributing Ruby programs, 551
environment variables
retrieving/setting, 545-546
storing as arrays, 546
storing as hashes, 546
Etc module, 554-555
external programs, running
capturing command output, 533-534
exec method, 533
process manipulation, 534-538
system method, 532-533
interactivity testing, 553
piping into Ruby interpreter, 552-553
Shell library
file method, 544
fileUtils method, 544
foreach iterator, 544
I/O redirection, 542-543
popdir method, 544
pushdir method, 544
Transact method, 544
text filters, 547-548
system method, running external systems, 532-533

T

tab characters, expanding/compressing in strings, 98-99
tab completion via irb utility (development tools), 712
tables (lookup), binary trees as, 302-303
taint levels (security), 430-432
take operations on tuplespace (Rinda), 700
Talking Heads, 435
Tanaka, Akira, 658
Tanaka, Masahiro, 167
Tanenbaum, Andrew S., 625
Tao of Programming, The, 531
tap method, chaining methods, 393
tasks (Rake utility), 706
TCP (Transmission Control Protocol) and networking, 626
telnet client example (Ruby/Tk GUI toolkit), 460-463
templates (HTML)
ERB and, 678-679
Haml library and, 680
Liquid and, 681
Mustache library and, 681
temporary files, 340-341
Terminator 2: Judgment Day, 416
ternary decision operator, 58
testing
assertions and, 591-592
classes, object classes, 374-377
Cucumber testing tool, 594-596
development of, 585
expectation expressions, 588
file characteristics, 326
heckle testing tool, 608
interactivity, 553
Minitest testing tool, 589-594, 606
objects
equality of objects, 377-378
object classes, 374-377
range memberships, 201
RSpec testing tool, 586-589, 606
text
all-caps words, matching in regular expressions, 127
doubled words, detecting in regular expressions, 126
QtRuby GUI toolkit, 483-484
Ruby/GTK3 GUI toolkit, 471-473
Ruby/Tk GUI toolkit, 459-463
Shoes 4 GUI toolkit, 446-448
text editors (graphical), 717
text filters, 547-548
wrapping in strings, 99-100
TextMate 2 graphical text editor, 717
then keyword, syntax issues, 43
thermostat example (Ruby/Tk GUI toolkit), 456-459
Thomas, Dave, 60, 618, 652, 712
threaded servers (networking), implementing, 629-630
threads
$SAFE global variable, 502
creating, 497

accessing thread-local variables, 498-500
changing thread status, 500-503
querying thread status, 500-501
strings, 499
deadlocks, 505
debugging, 507-508
defining, 495
disadvantages of, 496
exception-handling, 504-508
exiting, 501
fibers and cooperative multitasking, 527-530
grouping, 508-509
JRuby and, 496
killing, 501
passing, 503
performance and, 496
prioritizing, 502
race conditions, 496
rendezvous, 505
return values, capturing, 505
Rubinius and, 496
stopping, 501
synchronizing, 509-510
collection searches in parallel, 525-526
condition variables, 517-518
monitored queues, 520-521
mutexes, 512-514
nested locks, 518-519
queue classes, 515-516
recursive deletion in parallel, 526-527
simple synchronization, 511-512
timeouts, 522-523
unsynchronized threads, 496
waiting for events, 524-525
thread-safe code, 496
unsynchronized threads, 496
uses for, 496
waking stopped threads, 503
threequel operator (===), 18, 48, 59, 376
ticket (airline) example, Ruby/GTK3 GUI toolkit, 474-477
time of day simple server example (networking), 627-629
timeouts, thread operation/synchronization, 522-523
TimerClock custom widget example (QtRuby GUI toolkit), 487-489
times/dates
asctime method, 226
converting, 151
seconds to larger units of time, 217
to/from epochs, 217
current time, determining, 212
Date class, 225
Date standard library, 224-226
DateTime class, 225
Daylight Savings Time, 212
days
day of the week, determining, 214
day of the year, finding, 219
number of days in a month, determining, 228
days_in_month method, 228
Easter, determining the date of, 211, 215
epochs, 212, 217
formatting (localized) and internationalization, 151
GMT, 212-213, 224
Gregorian calendar, 211, 224
hours, working with, 222
Julian calendar, 211
leap seconds, 218
leap years, 221
minutes, working with, 222
mktime method, 213
months, dividing into weeks, 229
Nth weekday, finding in a month, 215-216
printing dates, 151
seconds
converting to larger units of time, 217
leap seconds, 218
specific dates (pre-epoch), working with, 224
specific times (post-epoch), handling, 212-214
strftime method, 214, 222, 227
Time class, 225
Time standard library, 226
time values
adding intervals to, 223
comparing, 223
computing the difference between two time values, 224
formatting, 226
printing, 226
time zones, 222, 227-228

time/date strings, parsing, 225-226
UTC, 212-213, 227
validating, 219-220
week of the year, finding, 220
weeks
dividing months into, 229
finding a week of the year, 220
years
finding a week of the year, 220
leap years, 221
timeservers (official), contacting, 641
timestamps and files, 323-324
Tk development, 452. *See also* Ruby/Tk GUI toolkit
to_ary method, converting objects to arrays, 389-390
to_f method, 16, 87-89
to_i method, converting strings to numbers, 87-89
to_s method, 27
converting objects to strings, 388-389
string conversions, 80-82
to_str method
converting objects to strings, 388-389
string conversions, 80-82
tokenizing strings, 71-72
top level variables, 17
Tower of Hanoi puzzle, 295
TracePoint class, monitoring program execution, 439
tracking
class definitions, 435-438
object definitions, 435-438
transact method, Shell library, 544
transformations/special effects in images, 573-576
transforming arrays, 250-251
translations
defaults, 146-147
i18n, 144-145
interpolation, 148
keys, 144
messages, 144
namespaces, 147
pluralization, 149-150
YAML, 144-145
transliteration, 141
transport layer (networking), 625
trap method, process manipulation, 537
traversing trees, 298-300, 303
trees, 287
arrays, converting to, 303-304
binary trees
as lookup tables, 302-303
breadth-first insertion, 299
implementing, 298-300
sorting data via, 300-302
traversing, 299-300
directory trees, copying, 548-549
nodes
ancestors, 298
children, 298
descendants, 298
parents, 298
roots, 298
strings, converting to, 303-304
subtrees, 298
traversing, 298-300, 303
trigonometry, 183-184
truncating files, 334
tuplespace (Rinda)
class matches, 702
creating, 699
defining, 698
expiring/renewing tuples, 702
nil values as wildcards, 702
notify operations, 701
read all operations, 700
read operations, 699
synchronization, 700
take operations, 700
write operations, 700
Tze, Sun, 163

U

UCS-2, internationalization, 133
UDP (User Datagram Protocol) and networking, 626
umlauts, 137
unary unarray operator. *See* array expansion operator (*)
unassigned variables, 51
unbalanced punctuation in stacks, 294

unbuffered/buffered I/O operations, 320-321
undefining
  classes, 427
  methods, 425-426
undirected graphs. *See* graphs
Unicode
  codepoints, 133
  internationalization, 131-133
  normalization, 138
Unicorn web server, 666
unicursive graphs, 308
unions (set), 288
universal sets, 247
UNIX
  Rake utility (development tools), 706
  Ruby/X11 GUI toolkit, 493
unless statements, 17
unless-else, 45
unshift method, appending arrays, 253
unshift operator, arrays as queues, 254
unsynchronized threads, 496
until loops, 21-22, 58
update method, SHA-256 hash calculations of strings, 95
updating
  files, 314
  gems, 616
uppercase/lowercase characters
  regular expressions, controlling in, 75
  strings, controlling in, 74
upto method, generating successive strings, 94
URL (Uniform Resource Locators), retrieving web pages from, 657
UTC (Coordinated Universal Time), 212-213, 227
UTF-8
  ASCII compatibility, 134
  backwards compatibility of, 134
  collation, 142
  internationalization, 133
  interpreting, 134
  memory conservation, 134
UTF-16, internationalization, 133

V

validating times/dates, 219-220
variables, 11. *See also* attributes
  assigning values via for loops, 46
  built-in classes, 27
  class instance variables, 31, 56, 62, 371
  class variables, 45, 62
  classes, 45
  closure variable, 55
  declaring, 45
  environment variables
    retrieving/setting, 545-546
    storing as arrays, 546
    storing as hashes, 546
  err variable, 24
  for loops, assigning variable values via, 46
  global variables, 538
  index variable, 46
  instance variables
    class instance variables, 62
    class instance variables versus, 56
  local variables, 46
  naming, 47
  nil variables, 51
  numbered global variables, 50
  pseudovariables, 44
  Ruby, 27
  rules for, 9
  swapping, 51
  symbols and, 197
  top level variables, 17
  unassigned variables, 51
variance (data sets), determining, 187
vectors, 170-171. *See also* matrices
version management
  chruby utility, 721
  rbenv utility, 720-721
  rvm, 719-720
version numbers, matching in regular expressions, 127
versioning (semantic), gems, 616
vertices (graphs), 304, 307
VHLL (Very High-Level Language), Ruby as, 8
Vim, 717-718
von Neumann, John, 638

W

Wagener, Amanda, 724
waking stopped threads, 503
web applications, 661
  asset pipeline
    CoffeeScript and JavaScript, 683-685
    Sass and CSS, 682-683
  data storage
    data stores, 676
    databases, 674-676
  HTML
    ERB, 678-679
    Haml library, 680
    layouts, 677
    Liquid templates, 681
    Mustache library, 681
    partials, 677
    templates, 677
    templates and ERB, 678-679
    templates and Haml library, 680
    templates and Liquid, 681
    templates and Mustache library, 681
  HTTP servers
    Rack library, 664-666
    simple server example, 662-663
  Rails framework, 667
    asset pipeline, 681-685
    CoffeeScript and JavaScript, 683-685
    ERB and HTML, 678-679
    Haml library and HTML, 680
    parameters, 671-672
    Rails 4 Way, The, 673, 676
    routing, 669-671
    Sass and CSS, 682-683
  Ramaze framework, 667
  Sinatra framework, 668
    parameters, 671
    routing, 668-669
    Sinatra: Up and Running, 673
  static sites
    Jekyll, 690
    Middleman, 688-689
    Nanoc, 690
    Octopress, 690
  web services and HTTP, 686
    JSON for API, 686
    REST API, 687
web pages, retrieving from URL, 657
web resources
  ActiveRecord library, 676
  bug reports, 724
  CoffeeScript, 685
  CSS, 682
  feature requests, 724
  forums, 724
  gems, 613
  IRC, 725
  JavaScript, 685
  mailing lists, 724
  Middleman static site generator, 689
  Minitest testing tool, 594
  Nokogiri, 564, 566
  podcasts, 724
  Puma web server, 666
  Rake utility (development tools), 710
  RMagick image manipulation, 576, 579
  Ruby-GNOME2 project, 479
  Sass, 683
  Shoes 4 GUI toolkit, 452
  Unicorn web server, 666
  websites, 723
  YARD, 622
web servers
  Puma web server, 665-666
  Rack library and, 664-666
  Unicorn web server, 666
web services and HTTP, 686
  JSON for API, 686
  REST API, 687
websites
  iwanttolearnruby.com, 724
  rdoc.info, 723
  ruby-doc.org, 723
  ruby-lang.org, 723
  static sites
    Jekyll, 690
    Middleman, 688-689
    Nanoc, 690
    Octopress, 690
weeks (times/dates)
  day of the week, determining, 214
  dividing months into, 229
  week of the year, finding, 220
weighted graphs, 304

Weirich, Jim, 194, 617
Well-Grounded Rubyist, 60
while loops, 21-22, 58
whitespace
  here-documents, 67
  removing whitespace from strings, 84
Whorf, Benjamin Lee, 1
wide characters, internationalization, 131
widgets
  QtRuby GUI toolkit, 485
    button building example, 482
    check boxes, 486
    customizing widgets, 487-489
    radio buttons, 486
    text editor example, 483-484
    TimerClock custom widget example, 487-489
    windowed application example, 480-481
  Ruby/GTK3 GUI toolkit
    airline ticket example, 474-477
    button building example, 471
    menus, 478
    text editor example, 472-473
    windowed application example, 469
  Ruby/Tk GUI toolkit, 452
    button building example, 455-459
    check boxes example, 463-464
    images, 466
    list boxes, 466
    menus, 466
    radio buttons example, 465-466
    scrolling, 466
    text example, 459-463
wildcards, nil values as wildcards in tuplespaces, 702
Win32API as a GUI, 494
word_wrap method, 100
wrapping text lines in strings, 99-100
write operations on tuplespace (Rinda), 700
writers/readers (class-level), defining, 412-413
Wynne, Matt, 596

X

X11 GUI toolkit, 493
Xlib, 493
XML (Extensible Markup Language)
  parsing
    document parsing, 561-564
    stream parsing, 564-566
  Y2K and, 561
xmpfilter library, 714

Y

Y2K and XML, 561
yajl-ruby library (JSON), 561
YAML (Yet Another Markup Language), 144-145, 347-349
YARD (Yay! A Ruby Documentation tool), 622
years
  day of the year, finding, 219
  leap years, 221
  week of the year, finding, 220
yield keyword, 22, 52
Yoshida, Kazuhiro, 493

Z

zero method and matrices, 168
zero-length matches in regular expressions, 58
zeroes, representing, 45
Ziegler, Austin, 194
zlib library, 94