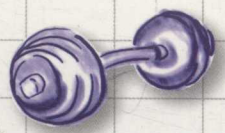
O'REILLY®

Head First Ruby

深入浅出Ruby（影印版）

用更少代码完成更多任务



使用代码块轻易完成繁重任务

掌握Ruby标准库

避免低级面向对象错误

通过40多个Ruby 练习领会

面向全球提供网络应用

東南大學出版社

Jay McGavren 著

深入浅出Ruby（影印版）

Head First Ruby

Jay McGavren 著

O'REILLY®

Beijing • Boston • Farnham • Sebastopol • Tokyo

O'Reilly Media, Inc.授权东南大学出版社出版

南京 东南大学出版社

图书在版编目(CIP)数据

深入浅出 Ruby：英文/(美)杰伊·麦加文(Jay McGavren)著. —影印本. —南京：东南大学出版社，2017.1

书名原文：Head First Ruby

ISBN 978-7-5641-6907-7

Ⅰ.①深… Ⅱ.①杰… Ⅲ.①计算机网络—程序设计—英文 Ⅳ.①TP393.09

中国版本图书馆 CIP 数据核字(2016)第 318070 号

图字：10-2015-253 号

深入浅出 Ruby(影印版)

出版发行：东南大学出版社

地　　址：南京四牌楼 2 号　　邮编：210096

出 版 人：江建中

网　　址：http://www.seupress.com

电子邮件：press@seupress.com

印　　刷：常州市武进第三印刷有限公司

开　　本：787 毫米×980 毫米　　12 开本

印　　张：48

字　　数：803 千字

版　　次：2017 年 1 月第 1 版

印　　次：2017 年 1 月第 1 次印刷

书　　号：ISBN 978-7-5641-6907-7

定　　价：108.00 元

本社图书若有印装质量问题，请直接与营销部联系。电话(传真)：025-83791830

To open source software creators everywhere.
You make all of our lives better.

Author of Head First Ruby

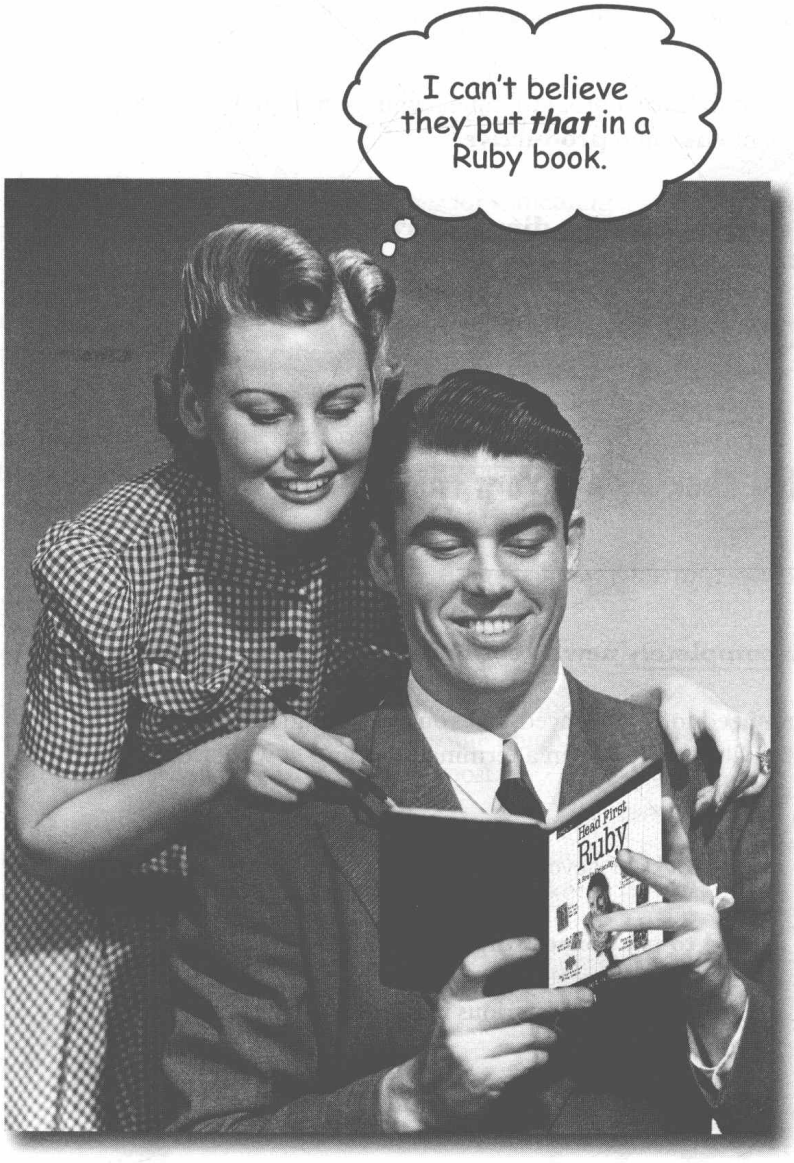
Jay McGavren was working on automation for a hotel services company when a colleague introduced him to *Programming Perl* (a.k.a. the Camel Book). It made him an instant Perl convert, as he liked actually writing code instead of waiting for a 10-person development team to configure a build system. It also gave him the crazy idea to write a technical book someday.

In 2007, with Perl sputtering, Jay was looking for a new interpreted language. With its strong object orientation, excellent library support, and incredible flexibility, Ruby won him over. He's since used Ruby for two game libraries, for a generative art project, and as a Ruby on Rails freelancer. He's been working in the online developer education space since 2011.

You can follow Jay on Twitter at *https://twitter.com/jaymcgavren*, or visit his personal website at *http://jay.mcgavren.com*.

how to use this book

Intro



In this section, we answer the burning question:
"So why DID they put that in a book on Ruby?"

Who is this book for?

If you can answer "yes" to ***all*** of these:

1. Do you have access to a computer with a text editor?

2. Do you want to learn a programming language that makes development **easy** and **productive**?

3. Do you prefer **stimulating dinner-party conversation** to **dry, dull, academic lectures**?

this book is for you.

Who should probably back away from this book?

If you can answer "yes" to any ***one*** of these:

1. **Are you completely new to computers?**

   (You don't need to be advanced, but you should understand folders and files, how to open a terminal app, and how to use a simple text editor.)

2. Are you a ninja rockstar developer looking for a ***reference*** **book**?

3. Are you **afraid to try something new**? Would you rather have a root canal than mix stripes with plaid? Do you believe that a technical book can't be serious if it describes class inheritance using armadillos?

this book is *not* for you.

[Note from Marketing: this book is for anyone with a valid credit card.]

We know what you're thinking

"How can *this* be a serious book on developing in Ruby?"

"What's with all the graphics?"

"Can I actually *learn* it this way?"

We know what your *brain* is thinking

Your brain craves novelty. It's always searching, scanning, *waiting* for something unusual. It was built that way, and it helps you stay alive.

So what does your brain do with all the routine, ordinary, normal things you encounter? Everything it *can* to stop them from interfering with the brain's *real* job—recording things that *matter*. It doesn't bother saving the boring things; they never make it past the "this is obviously not important" filter.

How does your brain *know* what's important? Suppose you're out for a day hike and a tiger jumps in front of you—what happens inside your head and body?

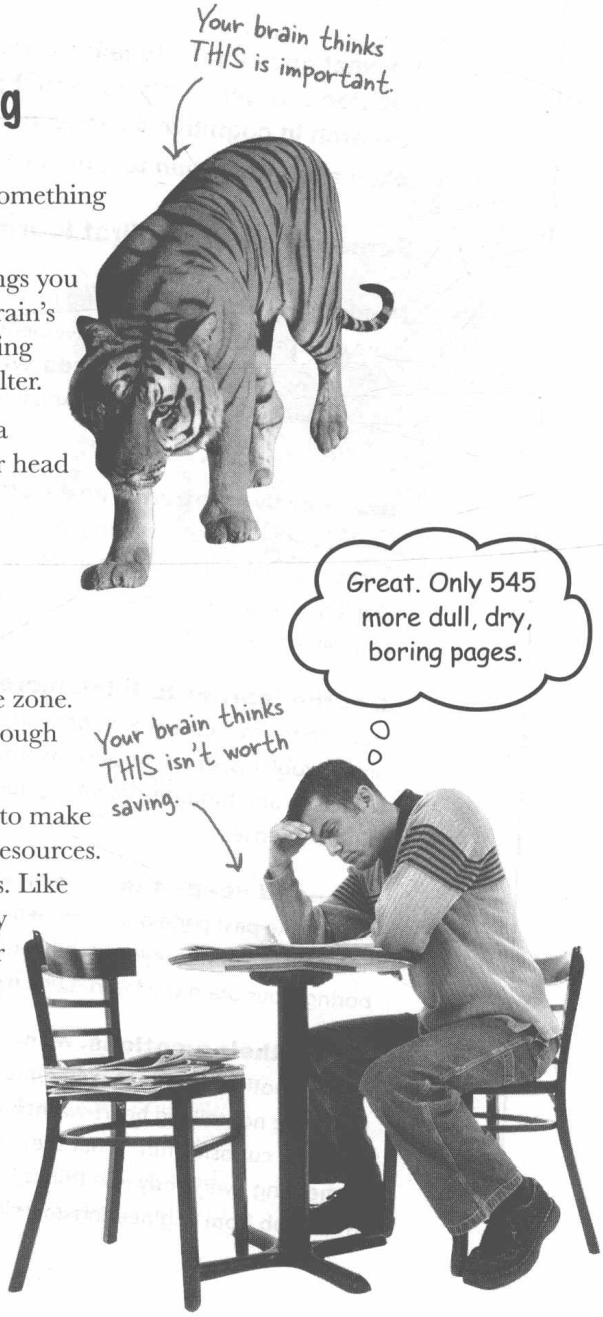
Neurons fire. Emotions crank up. *Chemicals surge*.

And that's how your brain knows…

This must be important! Don't forget it!

But imagine you're at home or in a library. It's a safe, warm, tiger-free zone. You're studying. Getting ready for an exam. Or trying to learn some tough technical topic your boss thinks will take a week, 10 days at the most.

Just one problem. Your brain's trying to do you a big favor. It's trying to make sure that this *obviously* unimportant content doesn't clutter up scarce resources. Resources that are better spent storing the really *big* things. Like tigers. Like the danger of fire. Like how you should never have posted those party photos on your Facebook page. And there's no simple way to tell your brain, "Hey, brain, thank you very much, but no matter how dull this book is, and how little I'm registering on the emotional Richter scale right now, I really *do* want you to keep this stuff around."



We think of a "Head First" reader as a learner.

So what does it take to *learn* something? First, you have to *get* it, then make sure you don't *forget* it. It's not about pushing facts into your head. Based on the latest research in cognitive science, neurobiology, and educational psychology, *learning* takes a lot more than text on a page. We know what turns your brain on.

Some of the Head First learning principles:

Make it visual. Images are far more memorable than words alone, and make learning much more effective (up to 89% improvement in recall and transfer studies). It also makes things more understandable. **Put the words within or near the graphics** they relate to, rather than on the bottom or on another page, and learners will be up to *twice* as likely to solve problems related to the content.

Use a conversational and personalized style. In recent studies, students performed up to 40% better on post-learning tests if the content spoke directly to the reader, using a first-person, conversational style rather than taking a formal tone. Tell stories instead of lecturing. Use casual language. Don't take yourself too seriously. Which would *you* pay more attention to: a stimulating dinner-party companion, or a lecture?

Get the learner to think more deeply. In other words, unless you actively flex your neurons, nothing much happens in your head. A reader has to be motivated, engaged, curious, and inspired to solve problems, draw conclusions, and generate new knowledge. And for that, you need challenges, exercises, and thought-provoking questions, and activities that involve both sides of the brain and multiple senses.

Get—and keep—the reader's attention. We've all had the "I really want to learn this, but I can't stay awake past page one" experience. Your brain pays attention to things that are out of the ordinary, interesting, strange, eye-catching, unexpected. Learning a new, tough, technical topic doesn't have to be boring. Your brain will learn much more quickly if it's not.

Touch their emotions. We now know that your ability to remember something is largely dependent on its emotional content. You remember what you care about. You remember when you *feel* something. No, we're not talking heart-wrenching stories about a boy and his dog. We're talking emotions like surprise, curiosity, fun, "what the…?", and the feeling of "I rule!" that comes when you solve a puzzle, learn something everybody else thinks is hard, or realize you know something that "I'm more technical than thou" Bob from Engineering *doesn't*.

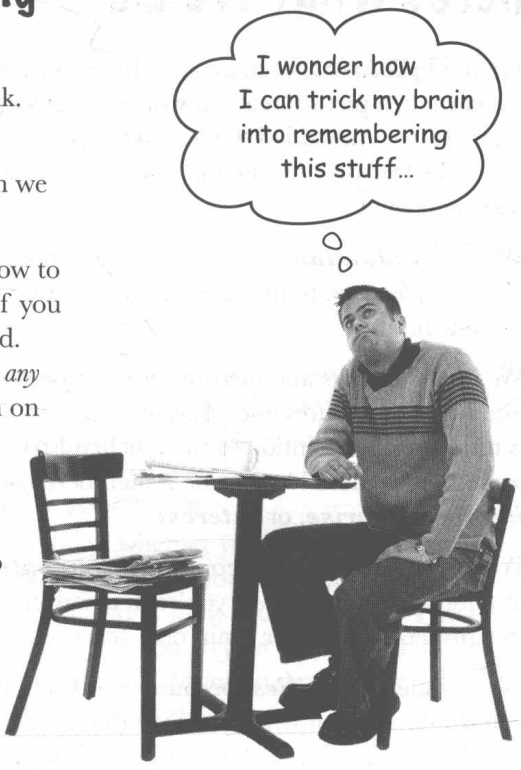
Metacognition: thinking about thinking

If you really want to learn, and you want to learn more quickly and more deeply, pay attention to how you pay attention. Think about how you think. Learn how you learn.

Most of us did not take courses on metacognition or learning theory when we were growing up. We were *expected* to learn, but rarely *taught* to learn.

But we assume that if you're holding this book, you really want to learn how to develop Ruby apps. And you probably don't want to spend a lot of time. If you want to use what you read in this book, you need to *remember* what you read. And for that, you've got to *understand* it. To get the most from this book, or *any* book or learning experience, take responsibility for your brain. Your brain on *this* content.

The trick is to get your brain to see the new material you're learning as Really Important. Crucial to your well-being. As important as a tiger. Otherwise, you're in for a constant battle, with your brain doing its best to keep the new content from sticking.



So just how *DO* you get your brain to treat programming like it was a hungry tiger?

There's the slow, tedious way, or the faster, more effective way. The slow way is about sheer repetition. You obviously know that you *are* able to learn and remember even the dullest of topics if you keep pounding the same thing into your brain. With enough repetition, your brain says, "This doesn't *feel* important to him, but he keeps looking at the same thing *over* and *over* and *over*, so I suppose it must be."

The faster way is to do ***anything that increases brain activity,*** especially different *types* of brain activity. The things on the previous page are a big part of the solution, and they're all things that have been proven to help your brain work in your favor. For example, studies show that putting words *within* the pictures they describe (as opposed to somewhere else in the page, like a caption or in the body text) causes your brain to try to makes sense of how the words and picture relate, and this causes more neurons to fire. More neurons firing = more chances for your brain to *get* that this is something worth paying attention to, and possibly recording.

A conversational style helps because people tend to pay more attention when they perceive that they're in a conversation, since they're expected to follow along and hold up their end. The amazing thing is, your brain doesn't necessarily *care* that the "conversation" is between you and a book! On the other hand, if the writing style is formal and dry, your brain perceives it the same way you experience being lectured to while sitting in a roomful of passive attendees. No need to stay awake.

But pictures and conversational style are just the beginning…

Here's what WE did

We used ***pictures***, because your brain is tuned for visuals, not text. As far as your brain's concerned, a picture really *is* worth a thousand words. And when text and pictures work together, we embedded the text *in* the pictures because your brain works more effectively when the text is *within* the thing it refers to, as opposed to in a caption or buried in the body text somewhere.

We used ***redundancy***, saying the same thing in *different* ways and with different media types, and *multiple senses*, to increase the chance that the content gets coded into more than one area of your brain.

We used concepts and pictures in ***unexpected*** ways because your brain is tuned for novelty, and we used pictures and ideas with at least *some* ***emotional*** *content*, because your brain is tuned to pay attention to the biochemistry of emotions. That which causes you to *feel* something is more likely to be remembered, even if that feeling is nothing more than a little ***humor***, ***surprise***, or ***interest.***

We used a personalized, ***conversational style***, because your brain is tuned to pay more attention when it believes you're in a conversation than if it thinks you're passively listening to a presentation. Your brain does this even when you're *reading*.

We included ***activities***, because your brain is tuned to learn and remember more when you ***do*** things than when you *read* about things. And we made the exercises challenging-yet-doable, because that's what most people prefer.

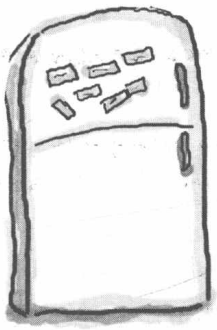
We used ***multiple learning styles***, because *you* might prefer step-by-step procedures, while someone else wants to understand the big picture first, and someone else just wants to see an example. But regardless of your own learning preference, *everyone* benefits from seeing the same content represented in multiple ways.

We include content for ***both sides of your brain***, because the more of your brain you engage, the more likely you are to learn and remember, and the longer you can stay focused. Since working one side of the brain often means giving the other side a chance to rest, you can be more productive at learning for a longer period of time.

And we included ***stories*** and exercises that present ***more than one point of view,*** because your brain is tuned to learn more deeply when it's forced to make evaluations and judgments.

We included ***challenges***, with exercises, and by asking ***questions*** that don't always have a straight answer, because your brain is tuned to learn and remember when it has to *work* at something. Think about it—you can't get your *body* in shape just by *watching* people at the gym. But we did our best to make sure that when you're working hard, it's on the *right* things. That ***you're not spending one extra dendrite*** processing a hard-to-understand example, or parsing difficult, jargon-laden, or overly terse text.

We used ***people***. In stories, examples, pictures, etc., because, well, *you're* a person. And your brain pays more attention to *people* than it does to *things*.



Cut this out and stick it on your refrigerator.

Here's what YOU can do to bend your brain into submission

So, we did our part. The rest is up to you. These tips are a starting point; listen to your brain and figure out what works for you and what doesn't. Try new things.

1. **Slow down. The more you understand, the less you have to memorize.**
Don't just *read*. Stop and think. When the book asks you a question, don't just skip to the answer. Imagine that someone really *is* asking the question. The more deeply you force your brain to think, the better chance you have of learning and remembering.

2. **Do the exercises. Write your own notes.**
We put them in, but if we did them for you, that would be like having someone else do your workouts for you. And don't just *look* at the exercises. **Use a pencil.** There's plenty of evidence that physical activity *while* learning can increase the learning.

3. **Read "There Are No Dumb Questions."**
That means all of them. They're not optional sidebars, ***they're part of the core content!*** Don't skip them.

4. **Make this the last thing you read before bed. Or at least the last challenging thing.**
Part of the learning (especially the transfer to long-term memory) happens *after* you put the book down. Your brain needs time on its own, to do more processing. If you put in something new during that processing time, some of what you just learned will be lost.

5. **Talk about it. Out loud.**
Speaking activates a different part of the brain. If you're trying to understand something, or increase your chance of remembering it later, say it out loud. Better still, try to explain it out loud to someone else. You'll learn more quickly, and you might uncover ideas you hadn't known were there when you were reading about it.

6. **Drink water. Lots of it.**
Your brain works best in a nice bath of fluid. Dehydration (which can happen before you ever feel thirsty) decreases cognitive function.

7. **Listen to your brain.**
Pay attention to whether your brain is getting overloaded. If you find yourself starting to skim the surface or forget what you just read, it's time for a break. Once you go past a certain point, you won't learn faster by trying to shove more in, and you might even hurt the process.

8. **Feel something.**
Your brain needs to know that this *matters*. Get involved with the stories. Make up your own captions for the photos. Groaning over a bad joke is *still* better than feeling nothing at all.

9. **Write a lot of code!**
There's only one way to learn to develop Ruby apps: **write a lot of code**. And that's what you're going to do throughout this book. Coding is a skill, and the only way to get good at it is to practice. We're going to give you a lot of practice: every chapter has exercises that pose a problem for you to solve. Don't just skip over them—a lot of the learning happens when you solve the exercises. We included a solution to each exercise—don't be afraid to **peek at the solution** if you get stuck! (It's easy to get snagged on something small.) But try to solve the problem before you look at the solution. And definitely get it working before you move on to the next part of the book.

Read me

This is a learning experience, not a reference book. We deliberately stripped out everything that might get in the way of learning whatever it is we're working on at that point in the book. And the first time through, you need to begin at the beginning, because the book makes assumptions about what you've already seen and learned.

It helps if you've done a *little* programming in some other language.

Most developers discover Ruby *after* they've learned some other programming language. (They often come seeking refuge from that other language.) We touch on the basics enough that a complete beginner can get by, but we don't go into great detail on what a variable is, or how an `if` statement works. You'll have an easier time if you've done at least a *little* of this before.

We don't cover every single class, library, and method ever created.

Ruby comes with a *lot* of classes and methods built in. Sure, they're all interesting, but we couldn't cover them all if this book was *twice* as long. Our focus is on the core classes and methods that *matter* to you, the beginner. We make sure you have a deep understanding of them, and confidence that you know how and when to use them. In any case, once you're done with *Head First Ruby*, you'll be able to pick up any reference book and get up to speed quickly on the classes and methods we left out.

The activities are NOT optional.

The exercises and activities are not add-ons; they're part of the core content of the book. Some of them are to help with memory, some are for understanding, and some will help you apply what you've learned. ***Don't skip the exercises.***

The redundancy is intentional and important.

One distinct difference in a Head First book is that we want you to *really* get it. And we want you to finish the book remembering what you've learned. Most reference books don't have retention and recall as a goal, but this book is about *learning*, so you'll see some of the same concepts come up more than once.

The code examples are as lean as possible.

It's frustrating to wade through 200 lines of code looking for the two lines you need to understand. Most examples in this book are shown in the smallest possible context, so that the part you're trying to learn is clear and simple. So don't expect the code to be robust, or even complete. That's *your* assignment after you finish the book. The book examples are written specifically for *learning*, and aren't always fully functional.

We've placed all the example files on the Web so you can download them. You'll find them at *http://headfirstruby.com/*.

Acknowledgments

Series Founders:

Huge thanks need to go first to the Head First founders, **Kathy Sierra** and **Bert Bates**. I loved the series when I encountered it a decade ago, but never imagined I might be writing for it. Thank you for creating this amazing style of teaching!

Bert deserves double credit, for offering extensive feedback on the early drafts, when I hadn't quite mastered the Head First Way. Because of you, ***Head First Ruby*** is a better book, and I am a better author!

At O'Reilly:

My eternal gratitude to my editor, **Meghan Blanchette**. She has an unwavering commitment to make books the best they can be, resolve that sustained me through those grueling second (and third) revisions. Thanks to **Mike Loukides** for helping me get on O'Reilly's radar in the first place, and to **Courtney Nash** for getting the project set up.

Thanks also to everyone else at O'Reilly who made this happen, particularly **Rachel Monaghan**, **Melanie Yarbrough**, and the rest of the production team.

Technical Reviewers:

Everyone makes mistakes, but luckily I have tech reviewers **Avdi Grimm**, **Sonda Sengupta**, **Edward Yue Shung Wong**, and **Olivier Lacan** to catch all of mine. You will never know how many problems they found, because I swiftly destroyed all the evidence. But their help and feedback were definitely necessary and are forever appreciated!

And More Thanks:

Ryan Benedetti, for helping get the project unstuck when it was mired in the early chapters. **Deborah Robinson** for typesetting help. **Janet McGavren** and **John McGavren** for proofreading. **Lenny McGavren** for photography. Readers of the Early Release, especially **Ed Fresco** and **John Larkin**, for catching typos and other glitches. Members of the **Ruby Rogues Parley** forum for feedback on code samples.

Perhaps most importantly, thanks to **Christine**, **Courtney**, **Bryan**, **Lenny**, and **Jeremy** for their patience and support. It's good to be back!

Safari® Books Online

Safari Books Online (*www.safaribooksonline.com*) is an on-demand digital library that delivers expert content in both book and video form from the world's leading authors in technology and business. Technology professionals, software developers, web designers, and business and creative professionals use Safari Books Online as their primary resource for research, problem solving, learning, and certification training.

Safari Books Online offers a range of product mixes and pricing programs for organizations, government agencies, and individuals. Subscribers have access to thousands of books, training videos, and prepublication manuscripts in one fully searchable database from publishers like O'Reilly Media, Prentice Hall Professional, Addison-Wesley Professional, Microsoft Press, Sams, Que, Peachpit Press, Focal Press, Cisco Press, John Wiley & Sons, Syngress, Morgan Kaufmann, IBM Redbooks, Packt, Adobe Press, FT Press, Apress, Manning, New Riders, McGraw-Hill, Jones & Bartlett, Course Technology, and dozens more. For more information about Safari Books Online, please visit us online.

Table of Contents (Summary)

Table of Contents (the real thing)

Intro

Your brain on Ruby. Here *you* are trying to *learn* something, while here your *brain* is, doing you a favor by making sure the learning doesn't *stick*. Your brain's thinking, "Better leave room for more important things, like which wild animals to avoid and whether naked snowboarding is a bad idea." So how *do* you trick your brain into thinking that your life depends on knowing how to program in Ruby?

more with less

1 Code the Way You Want

You're wondering what this crazy Ruby language is all about, and if it's right for you. Let us ask you this: ***Do you like being productive?*** Do you feel like all those extra compilers and libraries and class files and keystrokes in your other language bring you closer to a **finished product**, **admiring coworkers**, and **happy customers**? Would you like a language that **takes care of the details** for you? If you sometimes wish you could stop maintaining boilerplate code and *get to work on your problem*, then Ruby is for you. Ruby lets you **get more done with less code.**