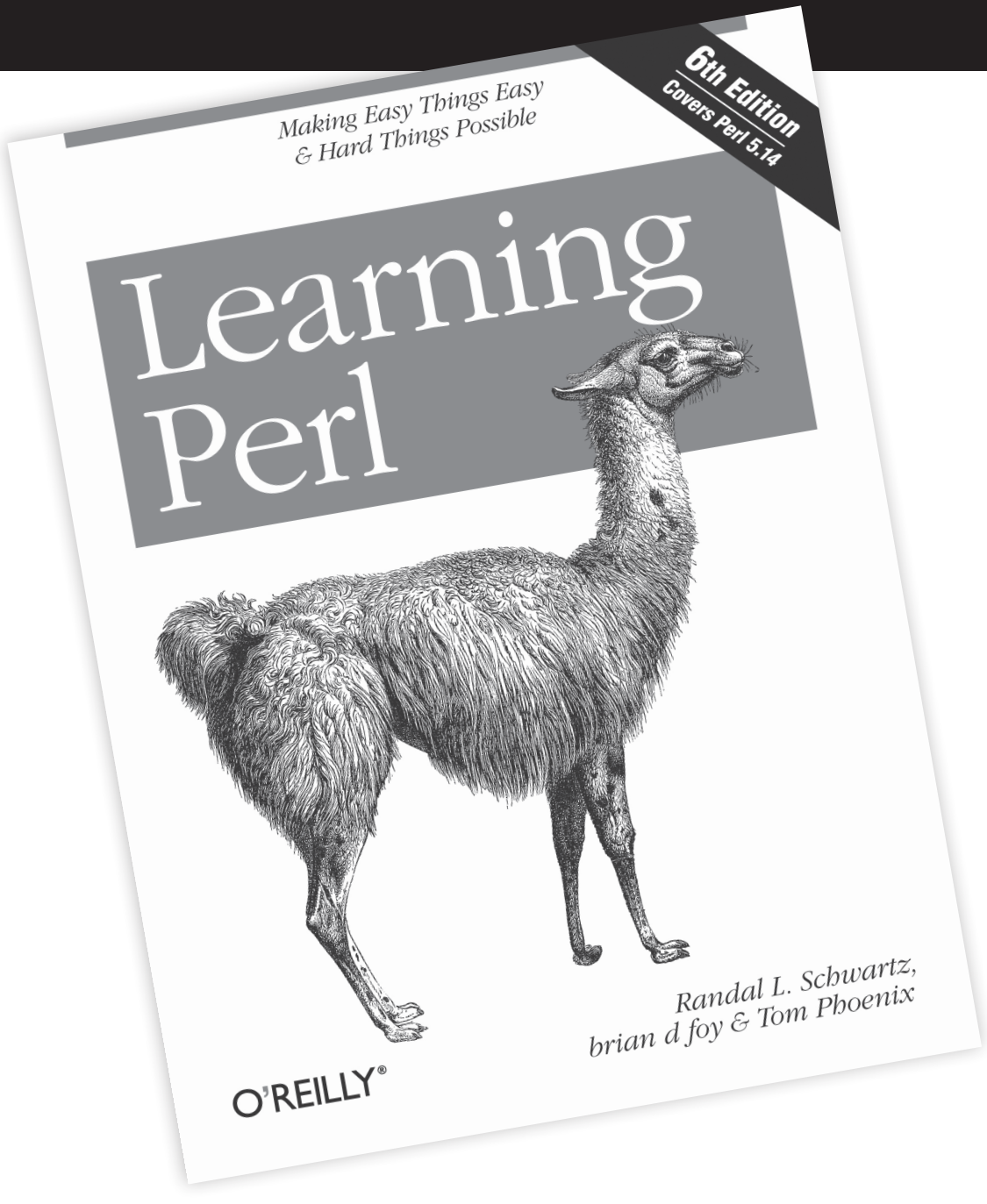
Student Workbook



O'REILLY®

brian d foy

SECOND EDITION

Student Workbook for Learning Perl

brian d foy

O'REILLY®
Beijing • Cambridge • Farnham • Köln • Sebastopol • Tokyo

Student Workbook for Learning Perl, Second Edition
by brian d foy

Printed in the United States of America.

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

O'Reilly books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (*http://my.safaribooksonline.com*). For more information, contact our corporate/institutional sales department: (800) 998-9938 or *corporate@oreilly.com*.

Editor: Shawn Wallace
Production Editor: Melanie Yarbrough

Cover Designer: Karen Montgomery
Interior Designer: David Futato
Illustrator: Robert Romano

January 2012: Second Edition.

Revision History for the Second Edition:
2012-01-25 First release
See *http://oreilly.com/catalog/errata.csp?isbn=9781449328061* for release details.

ISBN: 978-1-449-32806-1

[LSI]

1327515583

Table of Contents

Preface

This workbook is an additional set of exercises to supplement those already in *Learning Perl, Sixth Edition*. The chapter and page references apply only to that edition. Additionally, unless I denote otherwise, *Learning Perl* without qualification means the sixth edition. That book already contains some exercises, and I try to cover the topics those exercises didn't.

I've been teaching Perl since 1998, and my beginner classes use *Learning Perl* as the course text. Along the way, I've posed a lot of additional problems to students so they could test their understanding of the topics I covered and some of the additional information I added in the lecture. In doing this, I pose two sorts of problems: one of simple knowledge where knowing the right fact or trick makes the problem easy, and the other somewhat clever where a Perl implementation of a particular technique solves the problem. I've tried to include both of those sorts of problems in this book.

You should find some exercises are easy, some require that you find nuggets of information you may have missed on a first reading of *Learning Perl*, and others require you know something about algorithms or programming that aren't simply a question of Perl knowledge and might require you to do some research on your own. As you learn about how to learn about Perl, whether through its documentation or online resources, I pose these challenging problems expecting that you'll use everything available to you. There's a lot more to programming than just the syntax.

This isn't a classroom and you aren't being graded, unless it is a classroom and you *are* being graded. I won't think you are cheating if you use Google, and you shouldn't feel that you have to come up with everything just by staring at a blank document. If someone is grading you, just show them this paragraph and tell them I say it's okay. If you're the instructor, fear not: no one reads the preface anyway, especially if it's not on the test.

You should be able to complete most of the exercises with Perl 5.8, and I expect that you might even be able to use Perl 5.004, a quite ancient version indeed. Some of the exercises require a minimum version of Perl because I want you to use a particular feature introduced in that version, and I'll note the minimum version in the exercise when it's appropriate. If you don't note a minimum version assume that it's at least

Perl 5.8. I'd prefer that you use a supported version of Perl, which at this writing is at least Perl 5.14. However, I know that some of you don't get to make that choice.

Perl's motto, for good or bad, is "There's More Than One Way To Do It", so don't take my solutions as gospel. Remember that the corollary to that is "But most of them are wrong", so don't go out of your way to avoid techniques that I show. I show you a way to do it, and often I give you more than one way to do it. However, if you accomplished the task and your solution doesn't look remotely similar to mine, that's okay (probably).

Some of my answers have extra material that show techniques you'll read about in later chapters. I don't expect you to use those techniques, but I show them because they are the techniques and features you'll use in everyday programming once you finish the book. They are just a taste of things to come.

PART I

Exercises

CHAPTER 1
Introduction

1.1. The *perldoc* command is the key to all of the Perl documentation, and *perl* comes with literally thousands of pages of printed documentation (should you print them all out, which would take quite a long time). Take yourself on a tour of the *perldoc* command starting with the *perl* documentation. Next, try the *perlfunc* documentation, which lists all of the built-in Perl functions. This information may also be available in other formats on your system, and on some systems, you may need to download a separate package to get the Perl documentation (which should be a crime, but is becoming more common):

```
$ perldoc perl
$ perldoc perlfunc
```

(Answer to Exercise 1.1 on page 45)

1.2. The *perlfunc* documentation is much too long to scroll through every time you want to read about a Perl function, so *perldoc* has a handy switch to look up functions directly. Can you figure out what it is? (Answer to Exercise 1.2 on page 45)

1.3. The *perldoc* command has another switch, `-q`, which also looks up answers in the *perlfaq* documents. Give it a search term and it looks for answers (or questions) that contain that word. Try it with a few words to see what comes up. If you can't think of one, try **`comma`**. (Answer to Exercise 1.3 on page 45)

1.4. You saw the backticks (`) operator in the second example in this chapter in *Learning Perl* (page 17), although we do not cover it until Chapter 16. You can use any command line you like in backticks to save its output in your Perl program. Take the backtick operator for a spin by inserting your own command in it. (Answer to Exercise 1.4 on page 46)

1.5. The *perldoc* command, starting with version 3.15, has a switch that allows you to look up Perl special variables in the *perlvar* documentation. Find out what it is and use it to look up the variable `$_`. You may need to use special quoting so the shell does not think it should interpret the special characters. Also, before you start, ensure you have a recent enough version of *perldoc*. (Answer to Exercise 1.5 on page 46)

CHAPTER 2
Scalar Data

2.1. In Perl versions 5.6 onward, you can also turn on warnings with a pragma instead of the *–w* command-line switch:

```
use warnings;
```

You may already have a program that outputs warnings, but if you don't, write one and add the warnings pragma to it. If you can't come up with one, try these programs, each of which outputs a different sort of warning. Which warning do you get for each?

Program 1:

```
print;
```

Program 2:

```
3 + 4;
```

Program 3:

```
print $n + 1;
```

(Answer to Exercise 2.1 on page 49)

2.2. We mention the *perldiag* on page 29 of *Learning Perl*. This pragma gives you longer versions of warnings you get from my program. Perl has a shortcut into that documentation through the `diagnostics` pragma. Change the programs from the Exercise 2.1 to look in *perldiag* instead by replacing the warnings line with this one:

```
use diagnostics;
```

What difference do you see? (Answer to Exercise 2.2 on page 49)

2.3. Write a program that asks the user for an integer then reports if that number is odd or even. Turn on warnings and try this program with non-numerical input. What warning do you get? (Answer to Exercise 2.3 on page 50)

2.4. In Exercise 2.3, you read some input from the user, and immediately used `chomp` on the input to remove the trailing newline. Would the program still work if you took out the `chomp`? Would you get a warning? (Answer to Exercise 2.4 on page 51)

2.5. Remembering that Perl operations of the same precedence use associativity to decide what happens first, what is the answer to these operations? Which operation happens first? Does it matter?

```
2 ** 3 ** 4
2 / 3 * 4
2 + 3 * 4 ** 5 - 6
```

(Answer to Exercise 2.5 on page 51)

2.6. Write a program that asks the user for two numbers and reports which one is larger. What happens if they are the same? What happens if you don't enter digits, but something like **`thirty-seven`**? (Answer to Exercise 2.6 on page 51)

2.7. Repeatedly prompt the user to enter numbers. If the user enters a number, add that number to the sum of all the previously entered numbers. If the user ends input (with Control D on Unix-like systems and Mac OS X or Control Z on Windows), stop the program and report the sum. What happens if the user enters nothing but a newline? (Answer to Exercise 2.7 on page 52)

2.8. Write a program that prompts the user for a code point and prints the character for that code point. If you can't think of any interesting code points, try U+26F3, U+267A, U+2126, and U+03B6. Which characters are those code points?

If you need help setting up your terminal to handle Unicode, see Appendix C in *Learning Perl*.

(Answer to Exercise 2.8 on page 53)

CHAPTER 3
Lists and Arrays

3.1. Write a program to read in a list of strings then report the second-to-last element. There are many ways to do this, but two of them are easy. Use an array to store the lines. (Answer to Exercise 3.1 on page 55)

3.2. In Exercise 3.1, you used array indices to get to the second-to-last line. Do the same thing but don't use an array this time. (Answer to Exercise 3.2 on page 56)

3.3. Print the list of numbers from 1 to 10 and separate them with commas by interpolating an array in a double quoted string. (Answer to Exercise 3.3 on page 56)

3.4. Using a `foreach` loop, go through the numbers from 1 to 10 and print their squares and cubes. Use the range operator to create the list of numbers. (Answer to Exercise 3.4 on page 56)

3.5. Starting with the array `@numbers` that contains a list of numbers, use a `while` loop and `shift` to print the squares and cubes of the numbers in the list. (Answer to Exercise 3.5 on page 57)

3.6. The range operator, `..`, works with more than just numbers. Use the range operator to create a list of only the lowercase letters then a list of only the uppercase letters. Can you create a list of only the upper and lowercase letters together, and can you do it with only one range operator? (Answer to Exercise 3.6 on page 57)