ZED SHAW'S HARD WAY SERIES

Learn MORE PYTHON 3 the HARD WAY

The Next Step for New Python Programmers

ZED A. SHAW

FREE SAMPLE CHAPTER

SHARE WITH OTHERS

LEARN MORE PYTHON 3 THE HARD WAY

LEARN MORE PYTHON 3 THE HARD WAY

The Next Step for New Python Programmers

Zed A. Shaw

Addison-Wesley

Boston • Columbus • Indianapolis • New York • San Francisco • Amsterdam • Cape Town
Dubai • London • Madrid • Milan • Munich • Paris • Montreal • Toronto • Delhi • Mexico City
São Paulo • Sydney • Hong Kong • Seoul • Singapore • Taipei • Tokyo

Visit us on the Web: informit.com/aw

Library of Congress Control Number: 2017946529

ISBN-13: 978-0-13-412348-6
ISBN-10: 0-13-412348-4

1 17

Contents

This page intentionally left blank

Preface

Process, creativity, and quality. Burn these three words into your mind while you read this book. Process. Creativity. Quality. This book may be full of exercises that teach important topics every programmer should know, but the real knowledge you'll gain from the book is these three words. My goal in writing this book on programming is to teach you what I've known to be the three most important constants in software. Without process you'll flounder around wondering how to get started and have problems maintaining progress on long projects. Without creativity you'll be unable to solve the problems you'll encounter every day as a programmer. Without quality you'll have no idea if anything you're doing is any good.

Teaching you these three concepts is easy. I could simply write three blog posts and say, "There ya go, now you know what those three words mean." That isn't going to make you a better programmer and definitely not a person who can work on their own as a developer for the next 10 or 20 years. Simply knowing *about* process doesn't mean you can actually apply it in real practice. Reading a blog post about creativity doesn't help you find out how *you* are creative with code. To really understand these complex topics you'll want to internalize them, and the best way to do that is to apply them to simple projects.

As you work through the exercises in this book I will tell you which of the three you'll be working on. This is a change from my other books where I try to be sneaky and have you learn concepts without your realization. I'm going to be explicit this time because it's important that you keep the concept firmly in your mind so you can practice it throughout the exercise. You will then evaluate how well your attempt at applying the practice worked and what you can do to improve the next time. A key component of this book is the ability to reflect on your own capabilities objectively and improve yourself. You do this best by being focused on one technique or practice at a time while accomplishing some other goal.

In addition to process, creativity, and quality you'll also learn what I consider six important topics a modern programmer needs to function. These may change in the future, but they've been essential for decades now, so unless there's a drastic shift in technology they'll still apply. Even something like SQL in Part VI is still relevant because it teaches you how to structure data so that it doesn't logically fall apart later. Your secondary educational goals are the following:

1. Getting Started: You learn quick hacks to start a project.
2. Data Structures: I don't teach every single data structure, but I get you started down the path to learning them more completely.
3. Algorithms: Data structures are fairly pointless without a way to process them.
4. Parsing Text: The foundation of computer science is parsing, and knowing how to do that helps you learn programming languages as they become popular.

5. Data Modeling: I use SQL to teach you the basics of modeling stored data in a logical way.
6. Unix Tools: Command line tools are used throughout the book as projects for you to copy, and you then also learn advanced Unix command line tools.

At each part of the book you'll focus on one or two of the three practices at a time until finally, in Part VII, you'll apply them all as you build a simple website. The final projects aren't sexy. You won't learn how to create your next startup, but they are nice little projects that will help you apply what you know while learning Django.

It's All Personal

Many other books are designed to teach you these three concepts in the context of a team. When these books teach you about process it's all about how you work with another person on a project to maintain code. When they teach creativity it's all about how you go to meetings with your team to ask customers questions. Sadly most of these "professional" books don't really teach quality. This is all fine, but there's two problems with these team-style books for most beginners:

1. You don't have a team, so you can't practice what they're teaching. The team-oriented books are designed for junior programmers who already have jobs and need to work on the team they just joined. Until that happens to you, any team-oriented book is fairly useless to you.
2. What's the point of learning how to work on a team if your own personal process, creativity, and quality is a total mess? Despite what the fans of "team players" say, the vast majority of programming tasks are done solo, and your evaluation of your skills is usually done solo. If you work on a team, but your code is always low quality and you constantly have to ask team members for help, you get a low review from your boss. For all their talk of how awesome teams are, they never blame the team when a junior programmer can't work alone. They blame the junior programmer.

This book is *not* about being a good worker drone at Mega Enterprise, LLC. This book is about helping *you* improve *your personal skills* so that when you get a job you can work alone. If you improve your personal process then it makes sense that you'll be a stronger contributor on a team. It also means you can start and develop your own ideas, which is where the vast majority of projects start.

Using the Included Videos

Learn More Python 3 the Hard Way has an extensive set of videos demonstrating how the code works, debugging, and, most importantly, solutions to the challenges. The videos are the perfect place to demonstrate many common errors by breaking the Python code on purpose and showing you how to fix it. I also walk through the code using debugging and interrogation tricks and techniques. The videos

are where I show you how to “stop staring and ask” the code what’s wrong. You can watch these videos online at informit.com/title/9780134123486.

Register your copy of *Learn More Python 3 the Hard Way* on the InformIT site for convenient access to updates and corrections as they become available. To start the registration process, go to informit.com/register and log in or create an account. Enter the product ISBN (9780134123486) and answer the simple proof-of-purchase question. Then look on the Registered Products tab for an Access Bonus Content link next to this product, and follow that link to access the bonus materials.

EXERCISE 6

find

Hopefully you are discovering the various ways you sabotage yourself even before you begin to work. Maybe it's not that dramatic, but you should at least be identifying things you can improve in your environment that are making it difficult for you to start working. These little exercises are a good way for you to focus on the beginning since they are not that important and fit into a time scale that you can analyze. If these projects were hours long, you'd get bored reviewing what you've done and making improvements. A short 45-minute project is something you can take notes about (or record) and review very quickly.

This is a pattern I use throughout my studies. I'll identify something that I need to improve on, such as how I get started, or how I handle a tool. Then I'll devise an exercise that simply focuses on that. When I was learning to paint I struggled with going outside to paint trees. I sat down and looked at the problems, and the first thing I identified was I simply dragged too much stuff with me. I also kept all my things in random places around my apartment. I purchased a specific bag just for my painting supplies and kept that bag ready to go. When I wanted to paint outside I grabbed this bag and walked to one of a few places, rather than planning elaborate painting hikes. I practiced just grabbing my bag, walking to one of two places, setting up, doing a painting, then walking home until the process was smooth as silk. After that I watched Bob Ross to figure out how to paint trees because that guy can crank out some trees.

This is what you should be doing. One place many people waste time and effort is in their work area. Do you have a dedicated place to work that never changes? I ditched my laptop and now just use a desktop machine so that I can have a consistent place to do my work. This also saved my back and neck from hauling around that chunk of metal and gave me a bigger screen to work with, all improving my ability to work. In this exercise, I want you to focus on your work area and make sure that it's ready to go before you begin:

1. Do you have enough light? Do you need less light?
2. How's your chair? Do you need a better keyboard?
3. What other tools are getting in the way? Are you trying to do Unix-like things on a Windows machine? Trying to do Mac things on Linux? Don't go buy a new computer, but consider it for your next big purchase if you find there's just too much friction for what you want to do.
4. How's your desk? Do you even have one? Do you hack in cafés all day with terrible chairs and too much coffee?
5. How about music? Do you listen to music with words? I find that if I listen to music without words it's easier for me to focus on the voice in my head that helps me write or code.
6. Do you work in an open plan office and your coworkers are annoying? Go buy yourself a pair of big over-the-ear headphones. When you wear them it's obvious you're not paying attention,

so people will leave you alone and they'll feel it's less rude than if you're plugged in and they can't see. This will also block out distractions and help you focus.

Spend this exercise thinking about topics like this and trying to simplify and enhance your environment. One thing, though: Don't go buying crazy contraptions and spending tons of money. Just identify problems, and then try to find ways to fix them.

Exercise Challenge

In this challenge you are implementing a basic version of the `find` tool for finding files. You run `find` like this:

```
find . –name "*.txt" –print
```

That will search the current directory for every file ending in `.txt` and print it out. `find` has an insane number of command line arguments, so you are not expected to implement them all in one 45-minute session. The general format of find is the following:

1. The directory to start searching in: `.` or `/usr/local/`
2. A filter argument like `-name` or `-type d` (files of type directory)
3. An action to do with each found file: `-print`

You can do useful things like execute a command on every found file. If you want to delete every Ruby file in your home directory you can do this:

```
find . –name "*.rb" –exec rm {} \;
```

Please don't run this without realizing it will delete all the files that end in `.rb`. The `-exec` argument takes a command, replaces any instance of `{}` with the name of the file, and then stops reading the command when it hits a `;` (semicolon). We use `\;` in the preceding command because `bash` and many other shells use `;` as part of their language, so we have to escape it.

This exercise will really test your ability to use either `argparse` or `sys.argv`. I recommend you run `man find` to get a list of arguments, and then try using `find` to figure out exactly what arguments you'll implement. You only have 45 minutes, so you probably can't get too many, but `-name` and `-type` are both essential as well as `-print` and `-exec`. The `-exec` argument will be a challenge though, so save it for last.

When you implement this, try to find libraries that can do the work for you. You'll definitely want to look at the `subprocess` module and also the `glob` module. You will definitely want to look at `os` more carefully as well.

Study Drills

1. How much of `find` did you get implemented?
2. What are the libraries you found to improve this implementation?
3. Did you count finding libraries as part of your 45 minutes? You could say that research before you start hacking doesn't count, and I'd be alright with that. If you want the extra challenge, then include your research in the 45 minutes.

Further Study

How much of `find` can you implement in more 45 minute hacks? Maybe make this your hacking warmup challenge for the next week to see what you can get done. Remember that you should be trying to slap together the best ugly hack you can. Don't worry, I won't tell the Agile people you are just having fun.

This page intentionally left blank

EXERCISE 40

Reading with SQL

Out of the CRUD matrix you only know create. You can create tables and you can create rows in those tables. I'll now show you how to read, or in the case of SQL, SELECT:

ex5.sql

```
1   SELECT * FROM person;
2
3   SELECT name, age FROM pet;
4
5   SELECT name, age FROM pet WHERE dead = 0;
6
7   SELECT * FROM person WHERE first_name != "Zed";
```

Here's what each of these lines does:

Line 1 This says "select all columns from person and return all rows." The format for SELECT is `SELECT what FROM tables(s) WHERE (tests)`, and the WHERE clause is optional. The * (asterisk) character is what says you want all columns.

Line 3 In this one I'm only asking for two columns, `name` and `age`, from the `pet` table. It will return all rows.

Line 5 Now I'm looking for the same columns from the `pet` table, but I'm asking for *only* the rows where `dead = 0`. This gives me all the pets that are alive.

Line 7 Finally, I'm selecting all columns from `person` just like in the first line, but now I'm saying only if they do *not* equal "Zed." That WHERE clause is what determines which rows to return or not.

Select across Many Tables

Hopefully you're getting your head around selecting data out of tables. Always remember this: *SQL ONLY KNOWS TABLES. SQL LOVES TABLES. SQL ONLY RETURNS TABLES. TABLES. TABLES. TABLES. TABLES!* I repeat this in this rather crazy manner so that you will start to realize that what you know in programming isn't going to help. In programming you deal in graphs, and in SQL you deal in tables. They're related concepts, but the mental model is different.

Here's an example of where it becomes different. Imagine you want to know what pets Zed owns. You need to write a SELECT that looks in `person` and then "somehow" finds my pets. To do that you have to query the `person_pet` table to get the `id` columns you need. Here's how I'd do it:

ex6.sql

```
1    SELECT pet.id, pet.name, pet.age, pet.dead
2        FROM pet, person_pet, person
3        WHERE
4        pet.id = person_pet.pet_id AND
5        person_pet.person_id = person.id AND
6        person.first_name = "Zed";
```

Now, this looks huge, but I'll break it down so you can see it's simply crafting a new table based on data in the three tables and the WHERE clause:

Line 1 I only want some columns from `pet`, so I specify them in the select. In the last exercise you used * to say "every column" but that's going to be a bad idea here. Instead, you want to be explicit and say what column from each table you want, and you do that by using `table.column` as in `pet.name`.

Line 2 To connect `pet` to `person` I need to go through the `person_pet` relation table. In SQL that means I need to list all three tables after the FROM.

Line 3 Start the WHERE clause.

Line 4 First, I connect `pet` to `person_pet` by the related id columns `pet.id` and `person_pet.id`.

Line 5 *AND* I need to connect `person` to `person_pet` in the same way. Now the database can search for only the rows where the id columns all match up, and those are the ones that are connected.

Line 6 *AND* I finally ask for only the pets that I own by adding a `person.first_name` test for my first name.

Exercise Challenge

1. Write a query that finds all pets older than 10 years.
2. Write a query to find all people younger than you. Do one that's older.
3. Write a query that uses more than one test in the WHERE clause using the AND to write it. For example, `WHERE first_name = "Zed" AND age > 30`.
4. Do another query that searches for rows using three columns and uses both AND and OR operators.
5. This may be a mind-blowing, weird way to look at data if you already know a language like Python or Ruby. Take the time to model the same relationships using classes and objects, then map it to this setup.

6. Do a query that finds your pets you've added thus far.
7. Change the queries to use your `person.id` instead of the `person.name` like I've been doing.
8. Go through the output from your run, and make sure you know what table is produced for which SQL commands and how they produced that output.

Further Study

Continue to deep dive into SQLite3 by reading the documentation for the `SELECT` command at https://sqlite.org/lang_select.html and also read the documentation for the `EXPLAIN QUERY PLAN` feature at https://sqlite.org/eqp.html. If you ever wonder why SQLite3 did something, `EXPLAIN` is your answer.

This page intentionally left blank

Index