

Z E D S H A W ' S H A R D W A Y S E R I E S



Learn **PYTHON 3** the **HARD WAY**

A Very Simple Introduction
to the Terrifyingly Beautiful World of
Computers and Code

Z E D A . S H A W

About This E-Book

EPUB is an open, industry-standard format for e-books. However, support for EPUB and its many features varies across reading devices and applications. Use your device or app settings to customize the presentation to your liking. Settings that you can customize often include font, font size, single or double column, landscape or portrait mode, and figures that you can click or tap to enlarge. For additional information about the settings and features on your reading device or app, visit the device manufacturer's Web site.

Many titles include programming code or configuration examples. To optimize the presentation of these elements, view the e-book in single-column, landscape mode and adjust the font size to the smallest setting. In addition to presenting code and configurations in the reflowable text format, we have included images of the code that mimic the presentation found in the print book; therefore, where the reflowable format may compromise the presentation of the code listing, you will see a “Click here to view code image” link. Click the link to view the print-fidelity code image. To return to the previous page viewed, click the Back button on your device or app.

Learn Python 3 The Hard Way

**A Very Simple Introduction to the Terrifyingly
Beautiful World of Computers and Code 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

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 author 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 intlcs@pearson.com.

Visit us on the Web: informit.com/aw

Library of Congress Control Number: 2017940290

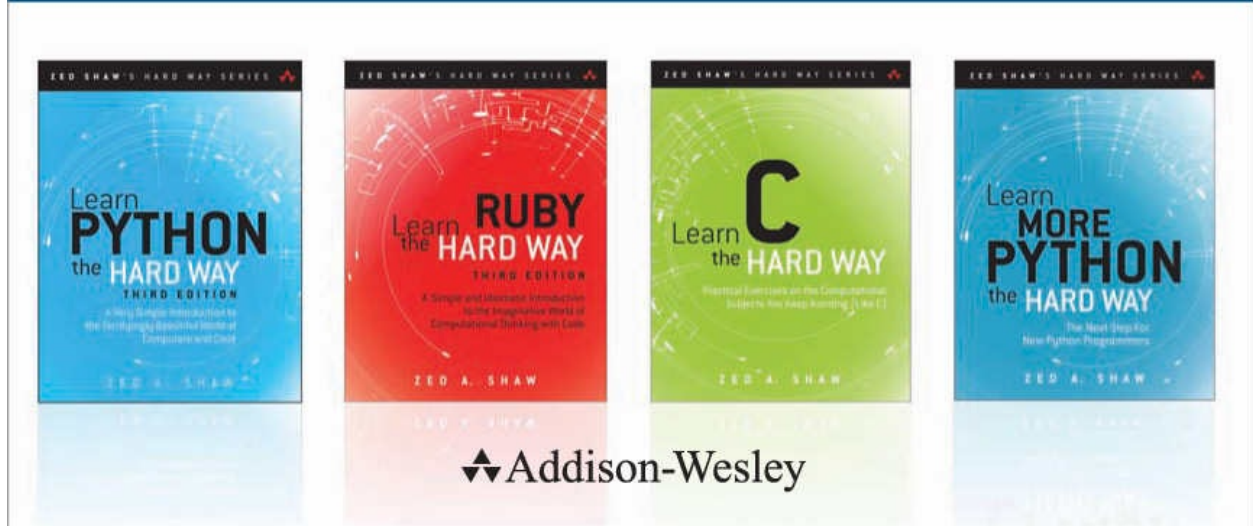
Copyright © 2017 Zed A. Shaw

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. For information regarding permissions, request forms and the appropriate contacts within the Pearson Education Global Rights & Permissions Department, please visit www.pearsoned.com/permissions/.

ISBN-13: 978-0-13-469288-3

ISBN-10: 0-13-469288-8

Zed Shaw's Hard Way Series



Visit informit.com/hardway for a complete list of available publications.

Zed Shaw's **Hard Way Series** emphasizes instruction and *making* things as the best way to get started in many computer science topics. Each book in the series is designed around short, understandable exercises that take you through a course of instruction that creates working software. All exercises are thoroughly tested to verify they work with real students, thus increasing your chance of success. The accompanying video walks you through the code in each exercise. Zed adds a bit of humor and inside jokes to make you laugh while you're learning.



Make sure to connect with us!
informit.com/socialconnect



Contents

[Preface](#)

[Improvements in the Python 3 Edition](#)

[The Hard Way Is Easier](#)

[Reading and Writing](#)

[Attention to Detail](#)

[Spotting Differences](#)

[Ask, Don't Stare](#)

[Do Not Copy-Paste](#)

[Using the Included Videos](#)

[A Note on Practice and Persistence](#)

[Acknowledgments](#)

[Exercise 0 The Setup](#)

[macOS](#)

[macOS: What You Should See](#)

[Windows](#)

[Windows: What You Should See](#)

[Linux](#)

[Linux: What You Should See](#)

[Finding Things on the Internet](#)

[Warnings for Beginners](#)

[Alternative Text Editors](#)

[Exercise 1 A Good First Program](#)

[What You Should See](#)

[Study Drills](#)

[Common Student Questions](#)

[Exercise 2 Comments and Pound Characters](#)

[What You Should See](#)

[Study Drills](#)

[Common Student Questions](#)

[Exercise 3 Numbers and Math](#)

[What You Should See](#)

[Study Drills](#)

[Common Student Questions](#)

[Exercise 4 Variables and Names](#)

[What You Should See](#)

[Study Drills](#)

[Common Student Questions](#)

[Exercise 5 More Variables and Printing](#)

[What You Should See](#)

[Study Drills](#)

[Common Student Questions](#)

[Exercise 6 Strings and Text](#)

[What You Should See](#)

[Study Drills](#)

[Break It](#)

[Common Student Questions](#)

[Exercise 7 More Printing](#)

[What You Should See](#)

[Study Drills](#)

[Break It](#)

[Common Student Questions](#)

[Exercise 8 Printing, Printing](#)

[What You Should See](#)

[Study Drills](#)

[Common Student Questions](#)

Exercise 9 Printing, Printing, Printing

[What You Should See](#)

[Study Drills](#)

[Common Student Questions](#)

Exercise 10 What Was That?

[What You Should See](#)

[Escape Sequences](#)

[Study Drills](#)

[Common Student Questions](#)

Exercise 11 Asking Questions

[What You Should See](#)

[Study Drills](#)

[Common Student Questions](#)

Exercise 12 Prompting People

[What You Should See](#)

[Study Drills](#)

[Common Student Questions](#)

Exercise 13 Parameters, Unpacking, Variables

[Hold Up! Features Have Another Name](#)

[What You Should See](#)

[Study Drills](#)

[Common Student Questions](#)

Exercise 14 Prompting and Passing

[What You Should See](#)

[Study Drills](#)

[Common Student Questions](#)

Exercise 15 Reading Files

[What You Should See](#)

[Study Drills](#)

[Common Student Questions](#)

[Exercise 16 Reading and Writing Files](#)

[What You Should See](#)

[Study Drills](#)

[Common Student Questions](#)

[Exercise 17 More Files](#)

[What You Should See](#)

[Study Drills](#)

[Common Student Questions](#)

[Exercise 18 Names, Variables, Code, Functions](#)

[What You Should See](#)

[Study Drills](#)

[Common Student Questions](#)

[Exercise 19 Functions and Variables](#)

[What You Should See](#)

[Study Drills](#)

[Common Student Questions](#)

[Exercise 20 Functions and Files](#)

[What You Should See](#)

[Study Drills](#)

[Common Student Questions](#)

[Exercise 21 Functions Can Return Something](#)

[What You Should See](#)

[Study Drills](#)

[Common Student Questions](#)

[Exercise 22 What Do You Know So Far?](#)

[What You Are Learning](#)

[Exercise 23 Strings, Bytes, and Character Encodings](#)

[Initial Research](#)

[Switches, Conventions, and Encodings](#)

[Disecting the Output](#)

[Disecting the Code](#)

[Encodings Deep Dive](#)

[Breaking It](#)

[Exercise 24 More Practice](#)

[What You Should See](#)

[Study Drills](#)

[Common Student Questions](#)

[Exercise 25 Even More Practice](#)

[What You Should See](#)

[Study Drills](#)

[Common Student Questions](#)

[Exercise 26 Congratulations, Take a Test!](#)

[Common Student Questions](#)

[Exercise 27 Memorizing Logic](#)

[The Truth Terms](#)

[The Truth Tables](#)

[Common Student Questions](#)

[Exercise 28 Boolean Practice](#)

[What You Should See](#)

[Study Drills](#)

[Common Student Questions](#)

[Exercise 29 What If](#)

[What You Should See](#)

[Study Drills](#)

[Common Student Questions](#)

Exercise 30 Else and If

[What You Should See](#)

[Study Drills](#)

[Common Student Questions](#)

Exercise 31 Making Decisions

[What You Should See](#)

[Study Drills](#)

[Common Student Questions](#)

Exercise 32 Loops and Lists

[What You Should See](#)

[Study Drills](#)

[Common Student Questions](#)

Exercise 33 While Loops

[What You Should See](#)

[Study Drills](#)

[Common Student Questions](#)

Exercise 34 Accessing Elements of Lists

[Study Drills](#)

Exercise 35 Branches and Functions

[What You Should See](#)

[Study Drills](#)

[Common Student Questions](#)

Exercise 36 Designing and Debugging

[Rules for if - statements](#)

[Rules for Loops](#)

[Tips for Debugging](#)

[Homework](#)

Exercise 37 Symbol Review

[Keywords](#)

[Data Types](#)

[String Escape Sequences](#)

[Old Style String Formats](#)

[Operators](#)

[Reading Code](#)

[Study Drills](#)

[Common Student Questions](#)

[Exercise 38 Doing Things to Lists](#)

[What You Should See](#)

[What Lists Can Do](#)

[When to Use Lists](#)

[Study Drills](#)

[Common Student Questions](#)

[Exercise 39 Dictionaries, Oh Lovely Dictionaries](#)

[A Dictionary Example](#)

[What You Should See](#)

[What Dictionaries Can Do](#)

[Study Drills](#)

[Common Student Questions](#)

[Exercise 40 Modules, Classes, and Objects](#)

[Modules Are Like Dictionaries](#)

[Classes Are Like Modules](#)

[Objects Are Like Import](#)

[Getting Things from Things](#)

[A First Class Example](#)

[What You Should See](#)

[Study Drills](#)

[Common Student Questions](#)

Exercise 41 Learning to Speak Object-Oriented

[Word Drills](#)

[Phrase Drills](#)

[Combined Drills](#)

[A Reading Test](#)

[Practice English to Code](#)

[Reading More Code](#)

[Common Student Questions](#)

Exercise 42 Is-A, Has-A, Objects, and Classes

[How This Looks in Code](#)

[About `class Name\(object\)`](#)

[Study Drills](#)

[Common Student Questions](#)

Exercise 43 Basic Object-Oriented Analysis and Design

[The Analysis of a Simple Game Engine](#)

[Write or Draw About the Problem](#)

[Extract Key Concepts and Research Them](#)

[Create a Class Hierarchy and Object Map for the Concepts](#)

[Code the Classes and a Test to Run Them](#)

[Repeat and Refine](#)

[Top Down versus Bottom Up](#)

[The Code for “Gothons from Planet Percal #25”](#)

[What You Should See](#)

[Study Drills](#)

[Common Student Questions](#)

Exercise 44 Inheritance versus Composition

[What Is Inheritance?](#)

[Implicit Inheritance](#)

[Override Explicitly](#)

[Alter Before or After](#)

[All Three Combined](#)
[The Reason for super\(\)](#)
[Using super\(\) with __init__](#)
[Composition](#)
[When to Use Inheritance or Composition](#)
[Study Drills](#)
[Common Student Questions](#)

Exercise 45 You Make a Game

[Evaluating Your Game](#)
[Function Style](#)
[Class Style](#)
[Code Style](#)
[Good Comments](#)
[Evaluate Your Game](#)

Exercise 46 A Project Skeleton

[macOS/Linux Setup](#)
[Windows 10 Setup](#)
[Creating the Skeleton Project Directory](#)
[Final Directory Structure](#)
[Testing Your Setup](#)
[Using the Skeleton](#)
[Required Quiz](#)
[Common Student Questions](#)

Exercise 47 Automated Testing

[Writing a Test Case](#)
[Testing Guidelines](#)
[What You Should See](#)
[Study Drills](#)
[Common Student Questions](#)

Exercise 48 Advanced User Input

[Our Game Lexicon](#)

[Breaking Up a Sentence](#)

[Lexicon Tuples](#)

[Scanning Input](#)

[Exceptions and Numbers](#)

[A Test First Challenge](#)

[What You Should Test](#)

[Study Drills](#)

[Common Student Questions](#)

Exercise 49 Making Sentences

[Match and Peek](#)

[The Sentence Grammar](#)

[A Word on Exceptions](#)

[The Parser Code](#)

[Playing with the Parser](#)

[What You Should Test](#)

[Study Drills](#)

[Common Student Questions](#)

Exercise 50 Your First Website

[Installing flask](#)

[Make a Simple “Hello World” Project](#)

[What’s Going On?](#)

[Fixing Errors](#)

[Create Basic Templates](#)

[Study Drills](#)

[Common Student Questions](#)

Exercise 51 Getting Input from a Browser

[How the Web Works](#)

[How Forms Work](#)

[Creating HTML Forms](#)

[Creating a Layout Template](#)

[Writing Automated Tests for Forms](#)

[Study Drills](#)

[Breaking It](#)

[Exercise 52 The Start of Your Web Game](#)

[Refactoring the Exercise 43 Game](#)

[Creating an Engine](#)

[Your Final Exam](#)

[Common Student Questions](#)

[Next Steps](#)

[How to Learn Any Programming Language](#)

[Advice from an Old Programmer](#)

[Appendix Command Line Crash Course](#)

[Introduction: Shut Up and Shell](#)

[How to Use This Appendix](#)

[You Will Be Memorizing Things](#)

[The Setup](#)

[Do This](#)

[You Learned This](#)

[Do More](#)

[Paths, Folders, Directories \(pwd\)](#)

[Do This](#)

[You Learned This](#)

[Do More](#)

[If You Get Lost](#)

[Do This](#)

[You Learned This](#)

[Make a Directory \(mkdir\)](#)

[Do This](#)

[You Learned This](#)

[Do More](#)

[Change Directory \(cd\)](#)

[Do This](#)

[You Learned This](#)

[Do More](#)

[List Directory \(ls\)](#)

[Do This](#)

[You Learned This](#)

[Do More](#)

[Remove Directory \(rmdir\)](#)

[Do This](#)

[You Learned This](#)

[Do More](#)

[Moving Around \(pushd, popd\)](#)

[Do This](#)

[You Learned This](#)

[Do More](#)

[Making Empty Files \(touch/New-Item\)](#)

[Do This](#)

[You Learned This](#)

[Do More](#)

[Copy a File \(cp\)](#)

[Do This](#)

[You Learned This](#)

[Do More](#)

[Moving a File \(mv\)](#)

[Do This](#)

[You Learned This](#)

[Do More](#)

[View a File \(less/more\)](#)

[Do This](#)

[You Learned This](#)

[Do More](#)

[Stream a File \(cat\)](#)

[Do This](#)

[You Learned This](#)

[Do More](#)

[Removing a File \(rm\)](#)

[Do This](#)

[You Learned This](#)

[Do More](#)

[Exiting Your Terminal \(exit\)](#)

[Do This](#)

[You Learned This](#)

[Do More](#)

[Command Line Next Steps](#)

[Unix Bash References](#)

[PowerShell References](#)

[**Index**](#)

Preface

This simple book is meant to get you started in programming. The title says it's the hard way to learn to write code, but it's actually not. It's only the "hard" way because it uses a technique called *instruction*. Instruction is where I tell you to do a sequence of controlled exercises designed to build a skill through repetition. This technique works very well with beginners who know nothing and need to acquire basic skills before they can understand more complex topics. It's used in everything from martial arts to music to even basic math and reading skills.

This book instructs you in Python by slowly building and establishing skills through techniques such as practice and memorization, then applying them to increasingly difficult problems. By the end of the book you will have the tools needed to begin learning more complex programming topics. I like to tell people that my book gives you your "programming black belt." What this means is that you know the basics well enough to now start learning programming.

If you work hard, take your time, and build these skills, you will learn to code.

Improvements in the Python 3 Edition

Learn Python 3 the Hard Way now uses Python 3.6. I've standardized on this version of Python because it has a new, improved string formatting system that is easier to use than the previous 4 (or 3, I forget, there were many) versions. There are a few problems with Python 3.6 for beginners, but I help you navigate these issues in the book. A particularly hairy problem is that Python 3.6 has very poor error messages in some key areas that I help you understand.

I have also improved the videos based on my experiences over the last five years teaching people Python. You can watch these videos online at informit.com/title/9780134692883. In the past the videos simply let you watch me do the exercise. The Python 3 edition videos also show you how to break—and then fix—every exercise. This skill is called "debugging." It teaches you how to fix problems you run into but also how Python runs the programs you're creating. The goal of this new methodology is to build a mental model of how Python runs your code so you can more easily figure out why it's broken. You'll also learn many useful tricks for debugging broken software.

Last, the Python 3 edition fully supports Microsoft Windows 10 from beginning to end. The previous edition focused mostly on the Unix-style systems such as macOS and Linux, with Windows being more of an afterthought. At the time I started writing the Python 3 edition Microsoft had started to take open source

started writing the book, many Python developers had started to take open source tools and developers seriously, and it was difficult to ignore them as a serious Python development platform. The videos feature Microsoft Windows using Python in various scenarios and also show macOS and Linux for full compatibility. I tell you about any gotchas on each platform, cover installation instructions, and provide any other tips I can give you.

The Hard Way Is Easier

With the help of this book, you will do the incredibly simple things that all programmers do to learn a programming language:

1. Go through each exercise.
2. Type in each file *exactly*.
3. Make it run.

That's it. This will be *very* difficult at first, but stick with it. If you go through this book and do each exercise for one or two hours a night, you will have a good foundation for moving on to another book about Python to continue your studies. This book won't turn you into a programmer overnight, but it will get you started on the path to learning how to code.

This book's job is to teach you the three most essential skills that a beginning programmer needs to know: reading and writing, attention to detail, and spotting differences.

Reading and Writing

If you have a problem typing, you will have a problem learning to code, especially if you have a problem typing the fairly odd characters in source code. Without this simple skill you will be unable to learn even the most basic things about how software works.

Typing the code samples and getting them to run will help you learn the names of the symbols, get familiar with typing them, and get you reading the language.

Attention to Detail

The one skill that separates good programmers from bad programmers is attention to detail. In fact, it's what separates the good from the bad in any profession. You must pay attention to the tiniest details of your work or you will miss important elements of what you create. In programming, this is how you end up with bugs and difficult-to-use systems.

By going through this book, and copying each example *exactly*, you will be training your brain to focus on the details of what you are doing, as you are

doing it.

Spotting Differences

A very important skill (that most programmers develop over time) is the ability to visually notice differences between things. An experienced programmer can take two pieces of code that are slightly different and immediately start pointing out the differences. Programmers have invented tools to make this even easier, but we won't be using any of these. You first have to train your brain the hard way, then use the tools.

While you do these exercises, typing each one in, you will make mistakes. It's inevitable; even seasoned programmers would make a few. Your job is to compare what you have written to what's required and fix all the differences. By doing so, you will train yourself to notice mistakes, bugs, and other problems.

Ask, Don't Stare

If you write code, you will write bugs. A "bug" means a defect, error, or problem with the code you've written. The legends say that this comes from an actual moth that flew into one of the first computers causing it to malfunction. Fixing it required "debugging" the computer. In the world of software, there are a *lot* of bugs. So many.

Like that first moth, your bugs will be hidden somewhere in the code, and you have to go find them. You can't just sit at your computer screen staring at the words you've written hoping that the answer jumps out at you. There is no more additional information you can get doing that, and you need additional information. You need to get up and go find the moth.

To do that you have to interrogate your code and ask it what is going on or look at the problem from a different view. In this book I frequently tell you to "stop staring and ask." I show you how to make your code tell you everything it can about what's going on and how to turn this into possible solutions. I also show you how to see your code in different ways, so you can get more information and insight.

Do Not Copy-Paste

You must *type* each of these exercises in, manually. If you copy-paste, you might as well not even do them. The point of these exercises is to train your hands, your brain, and your mind in how to read, write, and see code. If you copy-paste, you are cheating yourself out of the effectiveness of the lessons.

Using the Included Videos

Learn Python 3 the Hard Way has an extensive set of videos demonstrating how the code works and, most importantly, how to *break* it. 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/9780134692883.

A Note on Practice and Persistence

While you are studying programming, I’m studying how to play guitar. I practice it every day for at least two hours a day. I play scales, chords, and arpeggios for an hour and then learn music theory, ear training, songs, and anything else I can. Some days I study guitar and music for eight hours because I feel like it and it’s fun. To me repetitive practice is natural and just how to learn something. I know that to get good at anything I have to practice every day, even if I suck that day (which is often) or it’s difficult. Keep trying, and eventually it’ll be easier and fun.

Between the time that I wrote *Learn Python the Hard Way* and *Learn Ruby the Hard Way* I discovered drawing and painting. I fell in love with making visual art at the age of 39 and have been spending every day studying it in much the same way that I studied guitar, music, and programming. I collected books of instructional material, did what the books said, painted every day, and focused on enjoying the process of learning. I am by no means an “artist,” or even that good, but I can now say that I can draw and paint. The same method I’m teaching you in this book applied to my adventures in art. If you break the problem down into small exercises and lessons, and do them every day, you can learn to do almost anything. If you focus on slowly improving and enjoying the learning process, then you will benefit no matter how good you are at it.

As you study this book, and continue with programming, remember that anything worth doing is difficult at first. Maybe you are the kind of person who is afraid of failure, so you give up at the first sign of difficulty. Maybe you never learned self-discipline, so you can’t do anything that’s “boring.” Maybe you were told that you are “gifted,” so you never attempt anything that might make you seem stupid or not a prodigy. Maybe you are competitive and unfairly compare yourself to someone like me who’s been programming for more than 20 years.

Whatever your reason for wanting to quit, *keep at it*. Force yourself. If you run into a Study Drill you can’t do, or a lesson you just do not understand, then skip

it and come back to it later. Just keep going because with programming there's this very odd thing that happens. At first, you will not understand anything. It'll be weird, just like with learning any human language. You will struggle with words and not know what symbols are what, and it'll all be very confusing. Then, one day, *BANG*—your brain will snap and you will suddenly “get it.” If you keep doing the exercises and keep trying to understand them, you will get it. You might not be a master coder, but you will at least understand how programming works.

If you give up, you won't ever reach this point. You will hit the first confusing thing (which is everything at first) and then stop. If you keep trying, keep typing it in, keep trying to understand it and reading about it, you will eventually get it. If you go through this whole book, and you still do not understand how to code, at least you gave it a shot. You can say you tried your best and a little more, and it didn't work out, but at least you tried. You can be proud of that.

Register your copy of *Learn 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 (9780134692883) 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.

Acknowledgments

I would like to thank Angela for helping me with the first two versions of this book. Without her I probably wouldn't have bothered to finish it at all. She did the copyediting of the first draft and supported me immensely while I wrote it.

I'd also like to thank Greg Newman for doing the original cover art, Brian Shumate for early website designs, and all of the people who read this book and took the time to send me feedback and corrections.

Thank you.

Exercise 0. The Setup

This exercise has no code. It is simply the exercise you complete to get your computer to run Python. You should follow these instructions as exactly as possible. If you have problems following the written instructions, then watch the included videos for your platform.

Warning!

If you do not know how to use PowerShell on Windows, Terminal on macOS or bash on Linux then you need to go learn that first. You should do the exercises in the appendix first before continuing with these exercises.

macOS

Do the following tasks to complete this exercise:

1. Go to <https://www.python.org/downloads/release/python-360/> and download the version titled “Mac OS X 64-bit/32-bit installer.” Install it like you would any other software.
2. Go to <https://atom.io> with your browser, get the Atom text editor, and install it. If Atom does not suit you, then see the *Alternative Text Editors* section at the end of this exercise.
3. Put Atom (your text editor) in your dock, so you can reach it easily.
4. Find your Terminal program. Search for it. You will find it.
5. Put your Terminal in your dock as well.
6. Run your Terminal program. It won’t look like much.
7. In your Terminal program, run `python3.6`. You run things in Terminal by just typing the name and hitting RETURN.
8. Type `quit()`, Enter, and get out of `python3.6`.
9. You should be back at a prompt similar to what you had before you typed `python`. If not, find out why.
10. Learn how to make a directory in the Terminal.
11. Learn how to change into a directory in the Terminal.
12. Use your editor to create a file in this directory. Make the file, Save or

Save As . . . , and pick this directory.

13. Go back to Terminal using just the keyboard to switch windows.

14. Back in Terminal, list the directory with `ls` to see your newly created file.

macOS: What You Should See

Here's me doing this on my macOS computer in Terminal. Your computer might be different but should be similar to this.

[Click here to view code image](#)

```
$ python3.6
Python 3.6.0 (default, Feb 2 2017, 12:48:29)
[GCC 4.2.1 Compatible Apple LLVM 7.0.2 (clang-700.1.81)] on darwin
Type "help", "copyright", "credits" or "license" for more
information.
>>>
~ $ mkdir lpthw
~ $ cd lpthw
lpthw $ ls
# ... Use your text editor here to edit test.txt....
lpthw $ ls
test.txt
lpthw $
```

Windows

1. Go to <https://atom.io> with your browser, get the AtOm text editor, and install it. You do not need to be the administrator to do this.
2. Make sure you can get to AtOm easily by putting it on your desktop and/or in Quick Launch. Both options are available during setup. If you cannot run AtOm because your computer is not fast enough, then see the *Alternative Text Editors* section at the end of this exercise.
3. Run PowerShel1 from the Start menu. Search for it, and just press Enter to run it.
4. Make a shortcut to it on your desktop and/or Quick Launch for your convenience.
5. Run your PowerShel1 program (which I will call Terminal later). It won't look like much.
6. Download Python 3.6 from <https://www.python.org/downloads/release/python-360/> and install it. *Be sure to check the box that says to add Python 3.6 to your path.*

7. In your PowerShell (Terminal) program, run `python`. You run things in Terminal by just typing the name and pressing Enter. If you type `python` and it does not run, then you have to reinstall Python and make sure you check the box for “Add python to the PATH.” It’s very small so look carefully.
8. Type `quit()`, and press Enter to exit python.
9. You should be back at a prompt similar to what you had before you typed `python`. If not, find out why.
10. Learn how to make a directory in the PowerShell (Terminal).
11. Learn how to change into a directory in the PowerShell (Terminal).
12. Use your editor to create a file in this directory. Make the file, **Save** or **Save As . . .**, and pick this directory.
13. Go back to PowerShell (Terminal) using just the keyboard to switch windows. Look it up if you can’t figure it out.
14. Back in PowerShell (Terminal), list the directory to see your newly created file.

From now on, when I say “Terminal” or “shell” I mean PowerShell, and that’s what you should use. When I run `python3.6` you can just type `python`.

Windows: What You Should See

[Click here to view code image](#)

```
> python
>>> quit()
> mkdir lpthw
> cd lpthw
... Here you would use your text editor to make test.txt in lpthw ...
>
> dir
Volume in drive C is
Volume Serial Number is 085C-7E02

Directory of C:\Documents and Settings\you\lpthw

04.05.2010  23:32    <DIR>          .
04.05.2010  23:32    <DIR>          ..
04.05.2010  23:32                6 test.txt
                1 File(s)                6 bytes
                2 Dir(s)  14 804 623 360 bytes free
```

>

It is still correct if you see different information than mine, but yours should be similar.

Linux

Linux is a varied operating system with many different ways to install software. I'm assuming if you are running Linux then you know how to install packages, so here are your instructions:

1. Use your package manager to install Python 3.6, and if you can't, then download source from <https://www.python.org/downloads/release/python-360/> and build from source.
2. Use your Linux package manager, and install the Atom text editor. If Atom does not suit you, then see the *Alternative Text Editors* section at the end of this exercise.
3. Make sure you can get to Atom easily by putting it in your window manager's menu.
4. Find your Terminal program. It could be called GNOME Terminal, Konsole, or xterm.
5. Put your Terminal in your dock as well.
6. Run your Terminal program. It won't look like much.
7. In your Terminal program, run `python3.6`. You run things in Terminal by just typing the command name and pressing Enter. If you can't run `python3.6`, try running just `python`.
8. Type `quit()` and press Enter to exit python.
9. You should be back at a prompt similar to what you had before you typed `python`. If not, find out why.
10. Learn how to make a directory in Terminal.
11. Learn how to change into a directory in Terminal.
12. Use your editor to create a file in this directory. Typically, you will make the file, Save or Save As . . . , and pick this directory.
13. Go back to Terminal using just the keyboard to switch windows. Look it up if you can't figure it out.
14. Back in Terminal, list the directory to see your newly created file.

Linux: What You Should See

[Click here to view code image](#)

```
$ python
>>> quit()
$ mkdir lpthw
$ cd lpthw
# ... Use your text editor here to edit test.txt ...
$ ls
test.txt
$
```

It is still correct if you see different information than mine, but yours should be similar.

Finding Things on the Internet

A major part of this book is learning to research programming topics online. I'll tell you to "search for this on the internet," and your job is to use a search engine to find the answer. The reason I have you search instead of just giving you the answer is because I want you to be an independent learner who does not need my book when you're done with it. If you can find the answers to your questions online, then you are one step closer to not needing me, and that is my goal.

Thanks to search engines such as Google you can easily find anything I tell you to find. If I say, "search online for the python list functions," then you simply do this:

1. Go to <http://google.com>.
2. Type: python3 list functions.
3. Read the websites listed to find the best answer.

Warnings for Beginners

You are done with this exercise. This exercise might be hard for you depending on your familiarity with your computer. If it is difficult, take the time to read and study and get through it, because until you can do these very basic things you will find it difficult to get much programming done.

If someone tells you to stop at a specific exercise in this book or to skip certain ones, you should ignore that person. Anyone trying to hide knowledge from you, or worse, make you get it from them instead of through your own efforts, is trying to make you depend on them for your skills. Don't listen to them, and do the exercises anyway so that you learn how to educate yourself.

A programmer will eventually tell you to use macOS or Linux. If the programmer likes fonts and typography, they'll tell you to get a macOS

computer. If he likes control and has a huge beard, he will (or ze will if you prefer non-gendered pronouns for humans with beards) tell you to install Linux. Again, use whatever computer you have right now that works. All you need is an editor, a Terminal, and Python.

Finally, the purpose of this setup helps you do three things very reliably while you work on the exercises:

1. *Write* exercises using the text editor.
2. *Run* the exercises you wrote.
3. *Fix* them when they are broken.
4. Repeat.

Anything else will only confuse you, so stick to the plan.

Alternative Text Editors

Text editors are very important to a programmer, but as a beginner you only need a simple *programmer's text editor*. These are different from software for writing stories and books because they work with the unique needs of computer code. I recommend AtOm in this book because it is free and works nearly everywhere. However, AtOm may not run well on your computer, so here are some alternatives to try:

Editor Name	Works On	Where To Get It
Visual Studio Code	Windows, macOS, Linux	https://code.visualstudio.com
Notepad++	Windows	https://notepad-plus-plus.org
gEdit	Linux, macOS, Windows	https://github.com/GNOME/gedit
Textmate	macOS	https://github.com/textmate/textmate
SciTE	Windows, Linux	http://www.scintilla.org/SciTE.html
jEdit	Linux, macOS, Windows	http://www.jedit.org

These are ranked in order of most likely to work. Keep in mind that these projects may be abandoned, dead, or not work anymore on your computer. If you try one and it doesn't work, try another one. I've also listed the "Works On" in order of most likely to work, so if you're on Windows then look at the editors where Windows is listed first in the "Works On" column.

If you already know how to use Vim or Emacs then feel free to use them. If you have never used Vim or Emacs then avoid them. Programmers may try to convince you to use Vim or Emacs, but this will only derail you. Your focus is learning Python, not learning Vim or Emacs. If you try to use Vim and don't know how to quit, then type `:q!` or `ZZ`. If someone told you to use Vim, and

they didn't even tell you this, then now you know why you shouldn't listen to them.

Do not use an Integrated Development Environment (IDE) while you go through this book. Relying on an IDE means that you can't work with new programming languages until some company decides to sell you an IDE for that language. This means you can't use that new language until the language is large enough to justify a lucrative customer base. If you are confident you can work with only a programmer's text editor (like Vim, Emacs, Atom, etc.) then you don't have to wait for a third party. IDEs are nice in some situations (such as working with a giant existing code base) but being addicted to them will limit your future.

You should also not use IDLE. It has serious limitations in how it works and isn't a very good piece of software. All you need is a simple text editor, a shell, and Python.

Exercise 1. A Good First Program

Warning!

If you skipped [Exercise 0](#), then you are not doing this book right. Are you trying to use IDLE or an IDE? I said not to use one in [Exercise 0](#), so you should not use one. If you skipped [Exercise 0](#) please go back to it and read it.

You should have spent a good amount of time in [Exercise 0](#) learning how to install a text editor, run the text editor, run the Terminal, and work with both of them. If you haven't done that, then do not go on. You will not have a good time. This is the only time I'll start an exercise with a warning that you should not skip or get ahead of yourself.

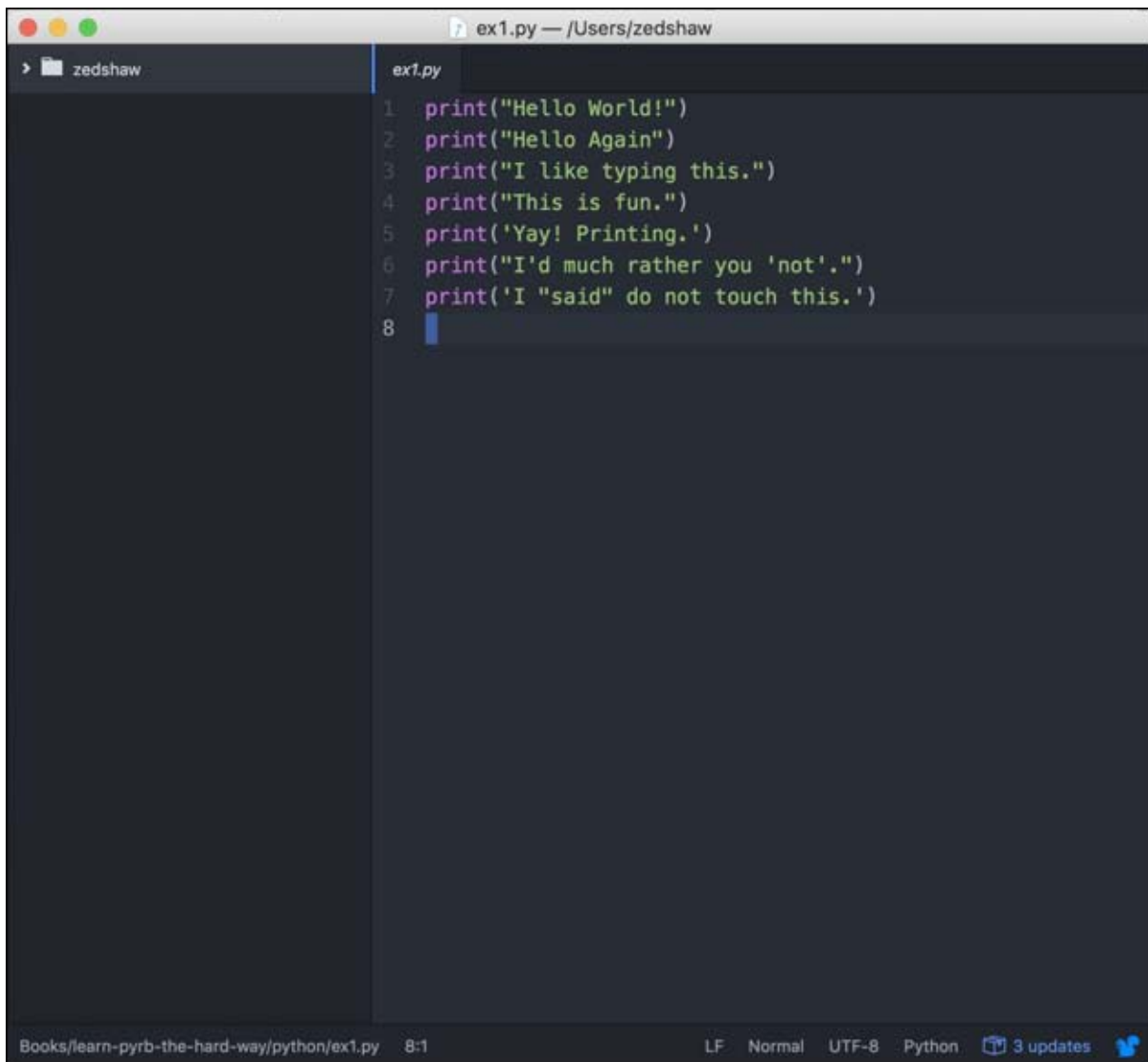
Type the following text into a single file named `ex1.py`. Python works best with files ending in `.py`.

[Click here to view code image](#)

`ex1.py`

```
1  print("Hello World!")
2  print("Hello Again")
3  print("I like typing this.")
4  print("This is fun.")
5  print('Yay! Printing.')
6  print("I'd much rather you 'not'.")
7  print('I "said" do not touch this.')
```

Your AtOm text editor should look something like this on all platforms:



```
ex1.py
1 print("Hello World!")
2 print("Hello Again")
3 print("I like typing this.")
4 print("This is fun.")
5 print('Yay! Printing.')
6 print("I'd much rather you 'not'.")
7 print('I "said" do not touch this.')
8
```

Don't worry if your editor doesn't look exactly the same; it should be close though. You may have a slightly different window header, maybe slightly different colors, and the left side of your Atom window won't say "zedshaw" but will instead show the directory you used for saving your files. All of those differences are fine.

When you create this file, keep in mind these points:

1. I did not type the line numbers on the left. Those are printed in the book so I can talk about specific lines by saying, "See line 5...." You do not type line numbers into Python scripts.
2. I have the `print` at the beginning of the line, and it looks exactly the same as what I have in `ex1.py`. Exactly means exactly, not kind of sort of the same. Every single character has to match for it to work. Color doesn't

matter, only the characters you type.

In macOS Terminal or (maybe) Linux *run* the file by typing:

```
python3.6 ex1.py
```

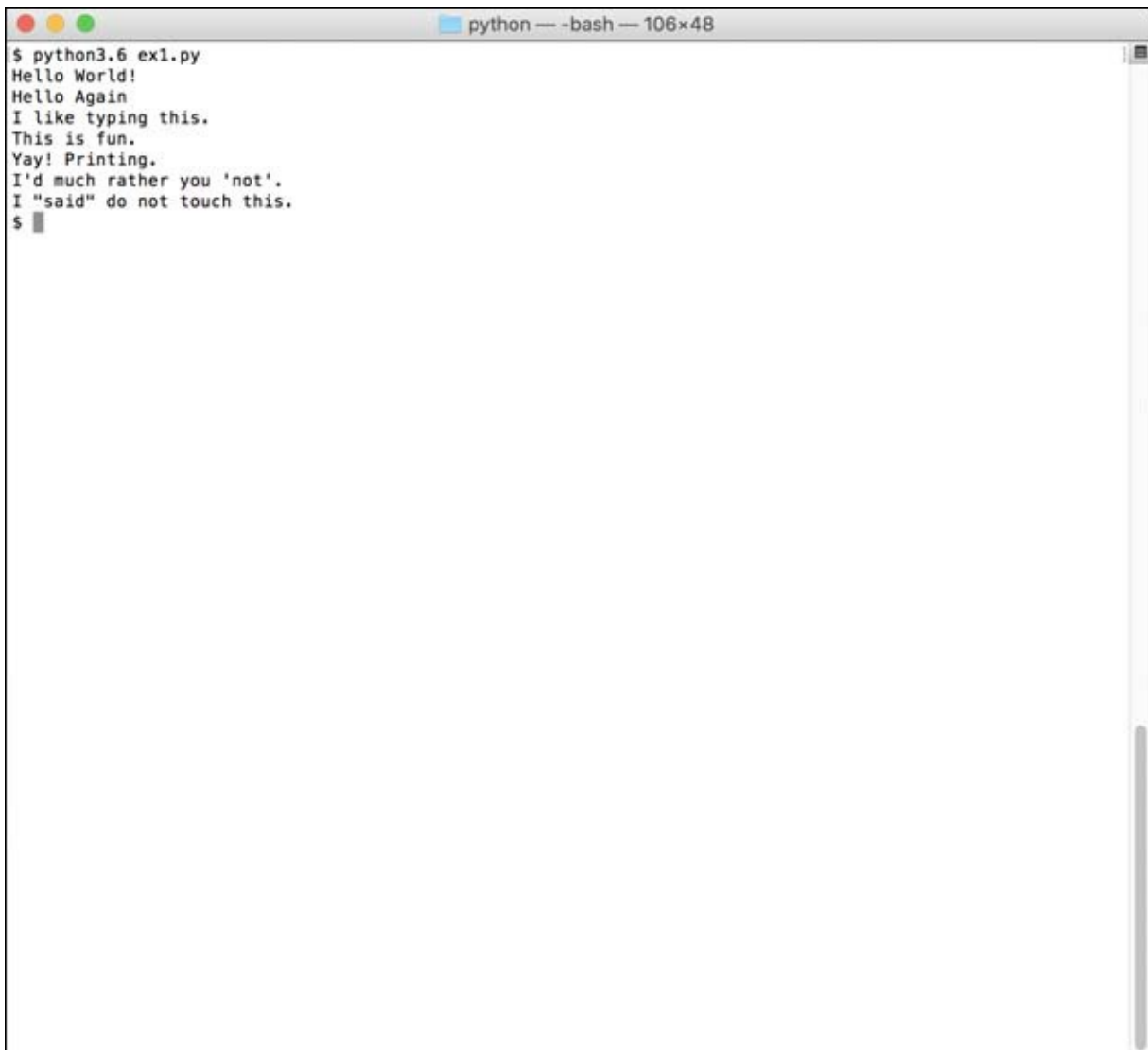
On Windows, remember you *always* type `python` instead of `python3.6`, like this:

```
python ex1.py
```

If you did it right then you should see the same output as I in the *What You Should See* section of this exercise. If not, you have done something wrong. No, the computer is not wrong.

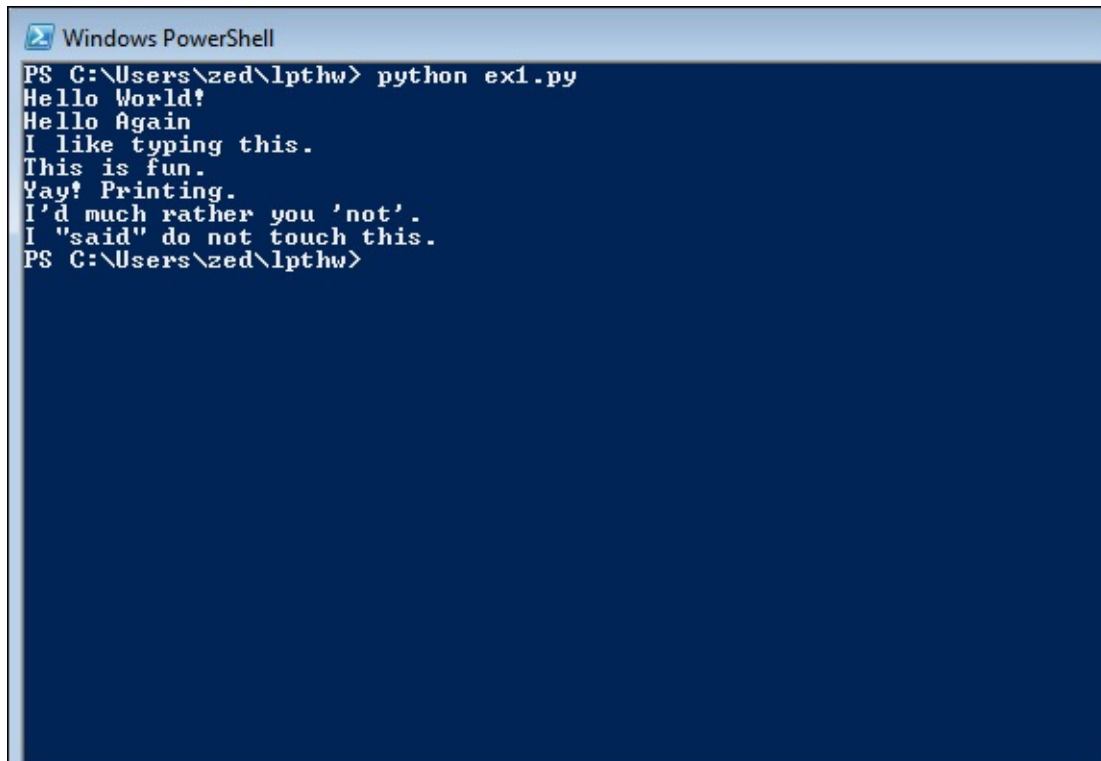
What You Should See

On macOS in the Terminal you should see this:

A terminal window with a title bar that reads "python — -bash — 106x48". The window contains the output of a Python script. The text is as follows:

```
$ python3.6 ex1.py
Hello World!
Hello Again
I like typing this.
This is fun.
Yay! Printing.
I'd much rather you 'not'.
I "said" do not touch this.
$
```

On Windows in PowerShell you should see this:

A screenshot of a Windows PowerShell window. The title bar says "Windows PowerShell". The command prompt shows the user running `python ex1.py` from the directory `C:\Users\zed\lpthw`. The output of the script is displayed line by line: `Hello World!`, `Hello Again`, `I like typing this.`, `This is fun.`, `Yay! Printing.`, `I'd much rather you 'not'.`, and `I "said" do not touch this.`. The prompt then returns to `PS C:\Users\zed\lpthw>`.

You may see different names before the `python3.6 ex1.py` command, but the important part is that you type the command and see the output is the same as mine.

If you have an error it will look like this:

[Click here to view code image](#)

```
$ python3.6 python/ex1.py
File "python/ex1.py", line 3
    print("I like typing this.
          ^
SyntaxError: EOL while scanning string literal
```

It's important that you can read these error messages because you will be making many of these mistakes. Even I make many of these mistakes. Let's look at this line by line.

1. We ran our command in the Terminal to run the `ex1.py` script.
2. Python tells us that the file `ex1.py` has an error on line 3 of `ex1.py`.
3. It prints this line of code for us to see it.
4. Then it puts a ^ (caret) character to point at where the problem is. Notice the missing " (double-quote) character?
5. Finally, it prints out a "SyntaxError" and tells us something about what might be the error. Usually these are very cryptic, but if you copy that text

into a search engine, you will find someone else who's had that error, and you can probably figure out how to fix it.

Study Drills

The Study Drills contain things you should *try* to do. If you can't, skip it and come back later.

For this exercise, try these things:

1. Make your script print another line.
2. Make your script print only one of the lines.
3. Put a # (octothorpe) character at the beginning of a line. What did it do? Try to find out what this character does.

From now on, I won't explain how each exercise works unless an exercise is different.

Warning!

An “octothorpe” is also called a “pound,” “hash,” “mesh,” or any number of other names. Pick the one that makes you chill out.

Common Student Questions

These are *actual* questions that real students have asked when doing this exercise.

Can I use IDLE? No, you should use Terminal on macOS and PowerShell on Windows, just like I have here. If you don't know how to use those, then you can go read the appendix.

How do you get colors in your editor? Save your file first as a .py file, such as ex1.py. Then you'll have color when you type.

I get `SyntaxError: invalid syntax` when I run ex1.py. You are probably trying to run Python, then trying to type Python again. Close your Terminal, start it again, and right away type only `python3.6 ex1.py`.

I get `can't open file 'ex1.py': [Errno 2] No such file or directory`. You need to be in the same directory as the file you created. Make sure you use the `cd` command to go there first. For example, if you saved your file in `lpthw/ex1.py`, then you would do `cd lpthw/` before trying to run `python3.6`

ex1.py. If you don't know what any of that means, then go through the appendix.

My file doesn't run; I just get the prompt back with no output. You most likely took the code in my ex1.py file literally and thought that `print("Hello World!")` meant to type only "Hello World!" into the file, without the `print`. Your file has to be *exactly* like mine.