Learn PYTHON the HARD WAY

THIRD EDITION

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

ZED SHAW

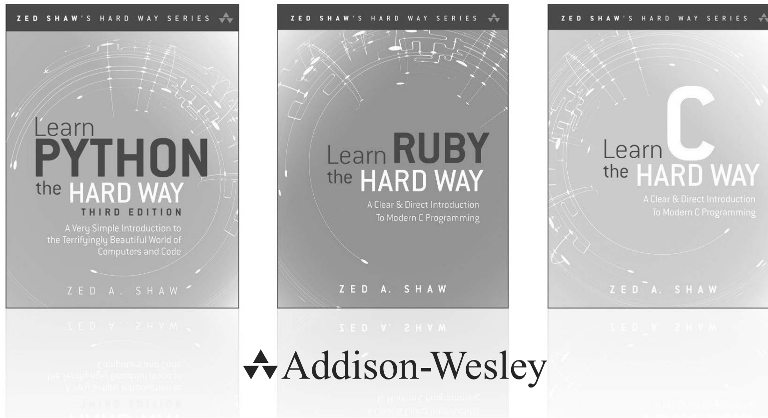
FREE SAMPLE CHAPTER

SHARE WITH OTHERS

LEARN PYTHON THE HARD WAY

Third Edition

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

informIT.com the trusted technology learning source | Addison-Wesley | Safari Books Online

ALWAYS LEARNING PEARSON

LEARN PYTHON THE HARD WAY

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

Third Edition

Zed A. Shaw

Addison-Wesley

Upper Saddle River, NJ • Boston • Indianapolis • San Francisco
New York • Toronto • Montreal • London • Munich • Paris • Madrid
Capetown • Sydney • Tokyo • Singapore • Mexico City

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and the publisher was aware of a trademark claim, the designations have been printed with initial capital letters or in all capitals.

The 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.

The publisher offers excellent discounts on this book when ordered in quantity for bulk purchases or special sales, which may include electronic versions and/or custom covers and content particular to your business, training goals, marketing focus, and branding interests. For more information, please contact:

U.S. Corporate and Government Sales
(800) 382-3419
corpsales@pearsontechgroup.com

For sales outside the United States, please contact:

International Sales
international@pearson.com

Visit us on the Web: informit.com/aw

Library of Congress Cataloging-in-Publication Data
Shaw, Zed.
Learn Python the hard way : a very simple introduction to the terrifyingly beautiful world of computers and code / Zed A. Shaw.—Third edition.
pages cm
Includes index.
ISBN 978-0-321-88491-6 (paperback : alkaline paper)
1. Python (Computer program language) 2. Python (Computer program language)—Problems, exercises, etc. 3. Computer programming—Problems, exercises, etc. I. Title.
QA76.73.P98S53 2014
005.13'3—dc23

2013029738

Copyright © 2014 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. To obtain permission to use material from this work, please submit a written request to Pearson Education, Inc., Permissions Department, One Lake Street, Upper Saddle River, New Jersey 07458, or you may fax your request to (201) 236-3290.

ISBN-13: 978-0-321-88491-6
ISBN-10: 0-321-88491-4

Text printed in the United States on recycled paper at RR Donnelley in Crawfordsville, Indiana.
Fourth printing, March 2014

Contents

Preface 1
Acknowledgments 1
The Hard Way Is Easier 1
Reading and Writing 2
Attention to Detail 2
Spotting Differences 2
Do Not Copy-Paste 2
Using the Included Videos 3
A Word of Advice for "Visual Learners" 3
A Note on Practice and Persistence 3
A Warning for the Smarties 4
Exercise 0 The Setup 6
Mac OSX 6
OSX: What You Should See 7
Windows 7
Windows: What You Should See 8
Linux 9
Linux: What You Should See 10
Warnings for Beginners 10
Exercise 1 A Good First Program 12
What You Should See 14
Study Drills 15
Common Student Questions 16
Exercise 2 Comments and Pound Characters 18
What You Should See 18
Study Drills 18
Common Student Questions 19
Exercise 3 Numbers and Math 20
What You Should See 21
Study Drills 21
Common Student Questions 22

Exercise 4 Variables and Names 24
What You Should See 25
Study Drills 25
Common Student Questions 25
Exercise 5 More Variables and Printing 28
What You Should See 28
Study Drills 29
Common Student Questions 29
Exercise 6 Strings and Text 30
What You Should See 31
Study Drills 31
Common Student Questions 31
Exercise 7 More Printing 32
What You Should See 32
Study Drills 32
Common Student Questions 33
Exercise 8 Printing, Printing 34
What You Should See 34
Study Drills 34
Common Student Questions 34
Exercise 9 Printing, Printing, Printing 36
What You Should See 36
Study Drills 36
Common Student Questions 37
Exercise 10 What Was That? 38
What You Should See 39
Escape Sequences 39
Study Drills 40
Common Student Questions 40
Exercise 11 Asking Questions 42
What You Should See 42
Study Drills 43
Common Student Questions 43

Exercise 12 Prompting People 44
What You Should See 44
Study Drills 44
Common Student Questions 45

Exercise 13 Parameters, Unpacking, Variables 46
Hold Up! Features Have Another Name 46
What You Should See 47
Study Drills 48
Common Student Questions 48

Exercise 14 Prompting and Passing 50
What You Should See 50
Study Drills 51
Common Student Questions 51

Exercise 15 Reading Files 54
What You Should See 55
Study Drills 55
Common Student Questions 56

Exercise 16 Reading and Writing Files 58
What You Should See 59
Study Drills 59
Common Student Questions 60

Exercise 17 More Files 62
What You Should See 63
Study Drills 63
Common Student Questions 63

Exercise 18 Names, Variables, Code, Functions 66
What You Should See 67
Study Drills 68
Common Student Questions 68

Exercise 19 Functions and Variables 70
What You Should See 71
Study Drills 71
Common Student Questions 71

Exercise 20 Functions and Files 74
What You Should See 75
Study Drills 75
Common Student Questions 75
Exercise 21 Functions Can Return Something 78
What You Should See 79
Study Drills 79
Common Student Questions 80
Exercise 22 What Do You Know So Far? 81
What You Are Learning 81
Exercise 23 Read Some Code 82
Exercise 24 More Practice 84
What You Should See 85
Study Drills 85
Common Student Questions 85
Exercise 25 Even More Practice 86
What You Should See 87
Study Drills 88
Common Student Questions 89
Exercise 26 Congratulations, Take a Test! 90
Common Student Questions 90
Exercise 27 Memorizing Logic 92
The Truth Terms 92
The Truth Tables 93
Common Student Questions 94
Exercise 28 Boolean Practice 96
What You Should See 98
Study Drills 98
Common Student Questions 98
Exercise 29 What If 100
What You Should See 100
Study Drills 101
Common Student Questions 101

Exercise 30 Else and If. 102
What You Should See 103
Study Drills 103
Common Student Questions 103
Exercise 31 Making Decisions. 104
What You Should See 105
Study Drills 105
Common Student Questions 105
Exercise 32 Loops and Lists. 106
What You Should See 107
Study Drills 108
Common Student Questions 108
Exercise 33 While-Loops 110
What You Should See 111
Study Drills 111
Common Student Questions 112
Exercise 34 Accessing Elements of Lists 114
Study Drills 115
Exercise 35 Branches and Functions. 116
What You Should See 117
Study Drills 118
Common Student Questions 118
Exercise 36 Designing and Debugging 120
Rules for If-Statements. 120
Rules for Loops 120
Tips for Debugging. 121
Homework. 121
Exercise 37 Symbol Review. 122
Keywords. 122
Data Types 123
String Escape Sequences. 124
String Formats. 124
Operators 125

Reading Code 126
Study Drills 127
Common Student Questions 127

Exercise 38 Doing Things to Lists 128
What You Should See 129
Study Drills 130
Common Student Questions 130

Exercise 39 Dictionaries, Oh Lovely Dictionaries 132
What You Should See 134
Study Drills 135
Common Student Questions 135

Exercise 40 Modules, Classes, and Objects 138
Modules Are Like Dictionaries 138
Classes Are Like Modules 139
Objects Are Like Mini-Imports 140
Getting Things from Things 141
A First-Class Example 141
What You Should See 142
Study Drills 142
Common Student Questions 143

Exercise 41 Learning to Speak Object Oriented 144
Word Drills 144
Phrase Drills 144
Combined Drills 145
A Reading Test 145
Practice English to Code 147
Reading More Code 148
Common Student Questions 148

Exercise 42 Is-A, Has-A, Objects, and Classes 150
How This Looks in Code 151
About class Name(object) 153
Study Drills 153
Common Student Questions 154

Exercise 43 Basic Object-Oriented Analysis and Design 156
The Analysis of a Simple Game Engine 157
Write or Draw about the Problem 157
Extract Key Concepts and Research Them 158
Create a Class Hierarchy and Object Map for the Concepts 158
Code the Classes and a Test to Run Them 159
Repeat and Refine 161
Top Down vs. Bottom Up 161
The Code for "Gothons from Planet Percal #25" 162
What You Should See 167
Study Drills 168
Common Student Questions 168
Exercise 44 Inheritance vs. Composition 170
What is Inheritance? 170
Implicit Inheritance 171
Override Explicitly 172
Alter Before or After 172
All Three Combined 174
The Reason for super() 175
Using super() with __init__ 175
Composition 176
When to Use Inheritance or Composition 177
Study Drills 177
Common Student Questions 178
Exercise 45 You Make a Game 180
Evaluating Your Game 180
Function Style 181
Class Style 181
Code Style 182
Good Comments 182
Evaluate Your Game 183
Exercise 46 A Project Skeleton 184
Installing Python Packages 184
Creating the Skeleton Project Directory 185

Final Directory Structure 186
Testing Your Setup 187
Using the Skeleton 188
Required Quiz 188
Common Student Questions 189

Exercise 47 Automated Testing 190
Writing a Test Case 190
Testing Guidelines. 192
What You Should See 192
Study Drills 193
Common Student Questions 193

Exercise 48 Advanced User Input. 194
Our Game Lexicon 194
Breaking Up a Sentence 195
Lexicon Tuples 195
Scanning Input. 195
Exceptions and Numbers. 196
What You Should Test 196
Design Hints 198
Study Drills 198
Common Student Questions 198

Exercise 49 Making Sentences 200
Match and Peek 200
The Sentence Grammar 201
A Word on Exceptions 203
What You Should Test 204
Study Drills 204
Common Student Questions 204

Exercise 50 Your First Website 206
Installing lpthw.web. 206
Make a Simple "Hello World" Project. 207
What's Going On?. 208
Fixing Errors 209

Create Basic Templates 209
Study Drills 211
Common Student Questions 211

Exercise 51 Getting Input from a Browser 214
How the Web Works. 214
How Forms Work 216
Creating HTML Forms 218
Creating a Layout Template 220
Writing Automated Tests for Forms 221
Study Drills 223
Common Student Questions 224

Exercise 52 The Start of Your Web Game 226
Refactoring the Exercise 43 Game 226
Sessions and Tracking Users 231
Creating an Engine 232
Your Final Exam 235
Common Student Questions 236

Next Steps 237
How to Learn Any Programming Language 238

Advice from an Old Programmer 241

Appendix Command Line Crash Course 243
Introduction: Shut Up and Shell 243
How to Use This Appendix 243
You Will Be Memorizing Things 244
Exercise 1: The Setup 245
Do This 245
You Learned This 246
Do More 246
Exercise 2: Paths, Folders, Directories (pwd) 248
Do This 248
You Learned This 249
Do More 249
Exercise 3: If You Get Lost 250

- Do This . . . 250
- You Learned This . . . 250

Exercise 4: Make a Directory (mkdir) . . . 250
- Do This . . . 250
- You Learned This . . . 252
- Do More . . . 252

Exercise 5: Change Directory (cd) . . . 252
- Do This . . . 252
- You Learned This . . . 255
- Do More . . . 255

Exercise 6: List Directory (ls) . . . 256
- Do This . . . 256
- You Learned This . . . 259
- Do More . . . 260

Exercise 7: Remove Directory (rmdir) . . . 260
- Do This . . . 260
- You Learned This . . . 262
- Do More . . . 262

Exercise 8: Move Around (pushd, popd) . . . 262
- Do This . . . 263
- You Learned This . . . 264
- Do More . . . 265

Exercise 9: Make Empty Files (Touch, New-Item) . . . 265
- Do This . . . 265
- You Learned This . . . 266
- Do More . . . 266

Exercise 10: Copy a File (cp) . . . 266
- Do This . . . 266
- You Learned This . . . 268
- Do More . . . 269

Exercise 11: Move a File (mv) . . . 269
- Do This . . . 269
- You Learned This . . . 271
- Do More . . . 271

Exercise 12: View a File (less, MORE) 271
Do This 271
You Learned This 272
Do More 272
Exercise 13: Stream a File (cat) 272
Do This 272
You Learned This 273
Do More 273
Exercise 14: Remove a File (rm) 273
Do This 273
You Learned This 275
Do More 275
Exercise 15: Exit Your Terminal (exit) 275
Do This 275
You Learned This 276
Do More 276
Command Line Next Steps 276
Unix Bash References 276
PowerShell References 277

Index 279

This page intentionally left blank

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 like 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.

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 copy editing of the first draft and supported me immensely while I wrote it.

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

Thank 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 sample *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. You might not really learn "programming" from this book, but you will learn the foundation skills you need to start learning the language.

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

It seems stupidly obvious, but 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 you familiar with typing them, and get you reading the language.

Attention to Detail

The one skill that separates bad programmers from good programmers is attention to detail. In fact, it's what separates the good from the bad in any profession. Without paying attention to the tiniest details of your work, you will miss key 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—which 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 you can use the tools.

While you do these exercises, typing each one in, you will make mistakes. It's inevitable; even seasoned programmers 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.

Do Not Copy-Paste

You must *type* each of these exercises in, manually. If you copy and paste, you might as well just 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

Included in the third edition of *Learn Python The Hard Way* is more than five hours of instructional videos. There is one video for each exercise where I either demonstrate the exercise or give you tips for completing the exercise. The best way to use the videos is to attempt or complete the exercises without them first, then use the videos to review what you learned or if you are stuck. This will slowly wean you off of using videos to learn programming and will build your skills at understanding code directly. Stick with it, and over time you won't need these videos, or any videos, to learn programming. You'll be able to just read for the information you need.

A Word of Advice for "Visual Learners"

Your belief that you are "only" a visual learner is potentially holding you back in your educational goals. The idea that a person could only possibly learn from one of their senses is preposterous. It is possible to learn to use all of your senses when tackling a complex subject such as programming. If you've locked yourself in the visual- or kinetic-learner prison your whole life, then this book is a great way to break out of it and build up your analytic skills.

By first attempting each exercise from the book, you will build your skills at analytic thinking, skills which most likely you already have but have simply forgotten how to use effectively. However, when you get stuck, grab the video for that exercise and use your ability to process visual information to help. In fact, finding ways to apply all of your sense to a given difficult problem will give you new insights into that problem no matter how strange it may seem at first.

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 at least 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 is just how to learn something. I know that to get good at anything you 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.

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 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, trying to understand it and reading about it, you will eventually get it.

But 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.

A Warning for the Smarties

Sometimes people who already know a programming language will read this book and feel I'm insulting them. There is nothing in this book that is intended to be interpreted as condescending, insulting, or belittling. I simply know more about programming than my *intended* readers. If you think you are smarter than me, then you will feel talked down to and there's nothing I can do about that because you are not my *intended* reader.

If you are reading this book and flipping out at every third sentence because you feel I'm insulting your intelligence, then I have three points of advice for you:

1. Stop reading my book. I didn't write it for you. I wrote it for people who don't already know everything.
2. Empty before you fill. You will have a hard time learning from someone with more knowledge if you already know everything.
3. Go learn Lisp. I hear people who know everything really like Lisp.

For everyone else who's here to learn, just read everything as if I'm smiling and I have a mischievous little twinkle in my eye.

EXERCISE 1

A Good First Program

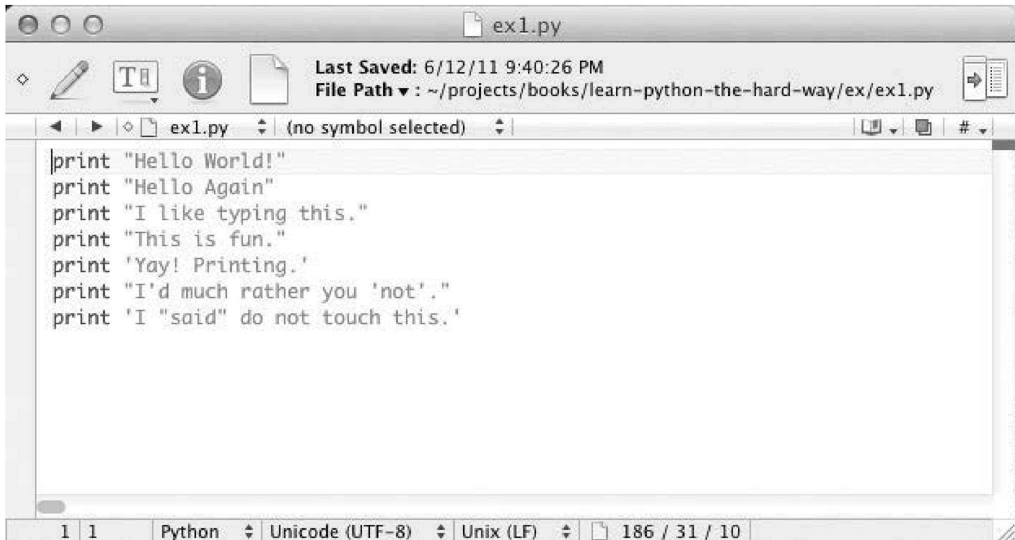
Remember, you should have spent a good amount of time in Exercise 0, learning how to install a text editor, run the text editor, run the Terminal, and work with both of them. If you haven't done that, then do not go on. You will not have a good time. This is the only time I'll start an exercise with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

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

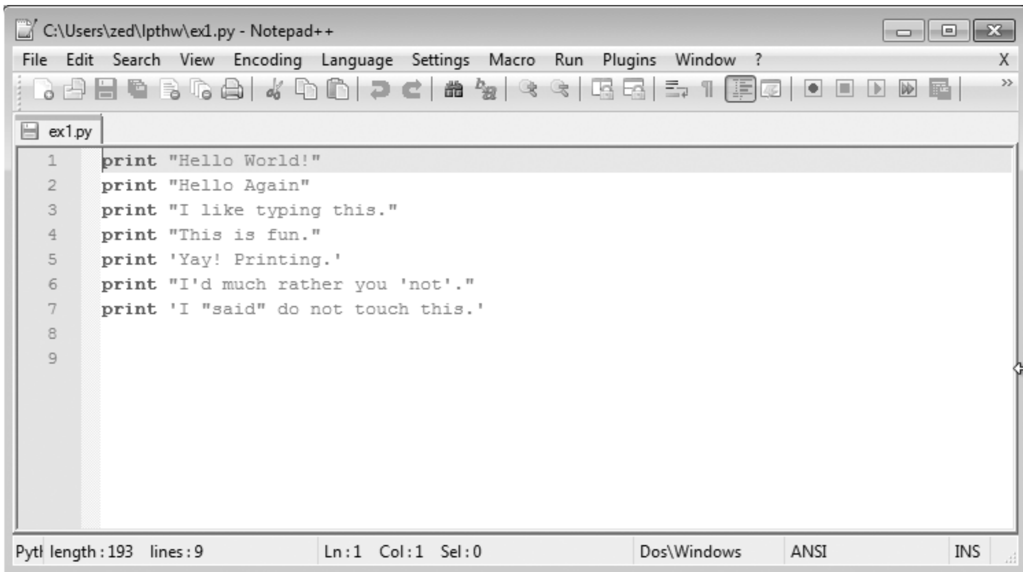
ex1.py

```
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.'
```

If you are on Mac OSX, then this is what your text editor might look like if you use TextWrangler:



If you are on Windows using Notepad++, then this is what it would look like:



Don't worry if your editor doesn't look exactly the same; the key points are as follows:

1. Notice 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 those into Python scripts.

2. Notice I have the `print` at the beginning of the line and how it looks exactly the same as what I have above. Exactly means exactly, not kind of sort of the same. Every single character has to match for it to work. But the colors are all different. Color doesn't matter; only the characters you type.

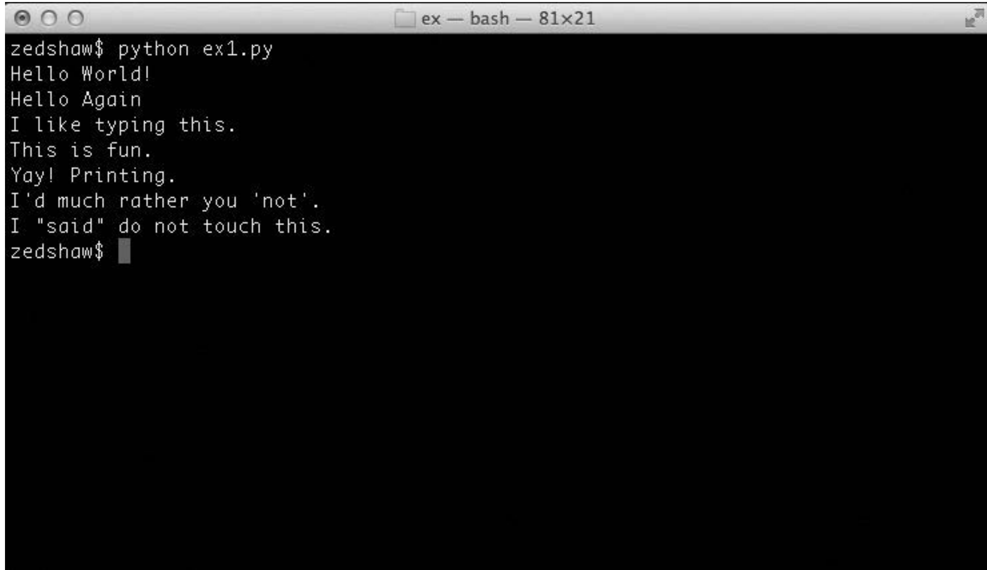
Then in Terminal, *run* the file by typing:

```
python ex1.py
```

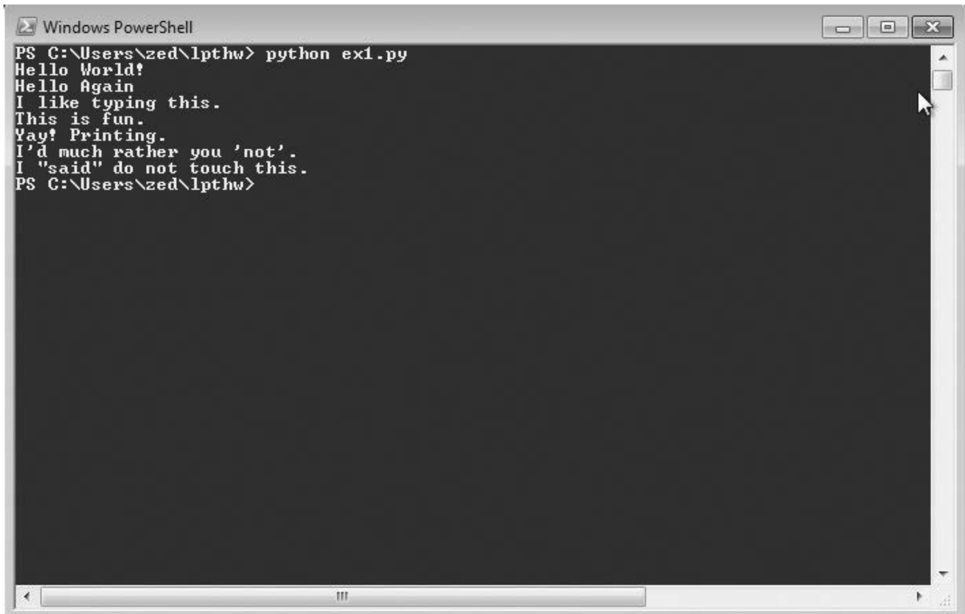
If you did it right, then you should see the same output I have below. If not, you have done something wrong. No, the computer is not wrong.

What You Should See

On Max OSX in the Terminal, you should see this:



On Windows in PowerShell, you should see this:



You may see different names, the name of your computer or other things, before the `python ex1.py`, 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:

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

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

1. Here we ran our command in the Terminal to run the `ex1.py` script.
2. Python then tells us that the file `ex1.py` has an error on line 3.
3. It then prints this line for us.
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.

WARNING! If you are from another country and you get errors about ASCII encodings, then put this at the top of your Python scripts:

```
# -*- coding: utf-8 -*-
```

It will fix them so that you can use Unicode UTF-8 in your scripts without a problem.

Study Drills

Each exercise also contains 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.

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

Common Student Questions

These are *actual* questions by real students in the comments section of the book when it was online. You may run into some of these, so I've collected and answered them for you.

Can I use IDLE?
No, you should use Terminal on OSX and PowerShell on Windows, just like I have here. If you don't know how to use those, then you can go read the Command Line Crash Course in 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 `python 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 `python ex1.py`. If you don't know what any of that means, then go through the Command Line Crash Course (CLI-CC) mentioned in the first question.

How do I get my country's language characters into my file?
Make sure you type this at the top of your file: `# -*- coding: utf-8 -*-`.

My file doesn't run; I just get the prompt back with no output.
You most likely took the previous code literally and thought that `print "Hello World!"` meant to literally print just `"Hello World!"` into the file, without the `print`. Your file has to be *exactly* like mine in the previous code and all the screenshots; I have `print "Hello World!"` and `print` before every line. Make sure your code is like mine and it should work.

Comments and Pound Characters

Comments are very important in your programs. They are used to tell you what something does in English, and they also are used to disable parts of your program if you need to remove them temporarily. Here's how you use comments in Python:

ex2.py

```
1    # A comment, this is so you can read your program later.
2    # Anything after the # is ignored by python.
3
4    print "I could have code like this." # and the comment after is ignored
5
6    # You can also use a comment to "disable" or comment out a piece of code:
7    # print "This won't run."
8
9    print "This will run."
```

From now on, I'm going to write code like this. It is important for you to understand that everything does not have to be literal. Your screen and program may visually look different, but what's important is the text you type into the file you're writing in your text editor. In fact, I could work with any text editor and the results would be the same.

What You Should See

Exercise 2 Session

```
$ python ex2.py
I could have code like this.
This will run.
```

Again, I'm not going to show you screenshots of all the Terminals possible. You should understand that the above is not a literal translation of what your output should look like visually, but the text between the first `$ Python ...` and last $ lines will be what you focus on.

Study Drills

1. Find out if you were right about what the # character does and make sure you know what it's called (octothorpe or pound character).

2. Take your `ex2.py` file and review each line going backward. Start at the last line, and check each word in reverse against what you should have typed.

3. Did you find more mistakes? Fix them.
4. Read what you typed previously out loud, including saying each character by its name. Did you find more mistakes? Fix them.

Common Student Questions

Are you sure # is called the pound character?
I call it the octothorpe and that is the only name that no country uses and that works in every country. Every country thinks its way to call this one character is both the most important way to do it and also the only way it's done. To me this is simply arrogance and, really, y'all should just chill out and focus on more important things like learning to code.

If # is for comments, then how come `# -*- coding: utf-8 -*-` works?
Python still ignores that as code, but it's used as a kind of "hack" or workaround for problems with setting and detecting the format of a file. You also find a similar kind of comment for editor settings.

Why does the # in `print "Hi # there."` not get ignored?
The # in that code is inside a string, so it will be put into the string until the ending " character is hit. These pound characters are just considered characters and aren't considered comments.

How do I comment out multiple lines?
Put a # in front of each one.

I can't figure out how to type a # character on my country's keyboard?
Some countries use the Alt key and combinations of those to print characters foreign to their language. You'll have to look online in a search engine to see how to type it.

Why do I have to read code backward?
It's a trick to make your brain not attach meaning to each part of the code, and doing that makes you process each piece exactly. This catches errors and is a handy error-checking technique.

This page intentionally left blank

Index

Symbols and Numbers

* (asterisk), 67

*args (asterisk args), 66–68

\ (backslash), 38–40

\\ (double backslash), 38, 252

[(left bracket), 106

] (right bracket), 106

^ (caret), 15

: (colon), 67–68

 code blocks and, 102, 110

 as slice syntax, 148

, (comma), 32–33

 in lists, 30, 68, 106

 print function and, 42, 76, 195

. (dot or period), 55, 88, 128, 138–139

.DS_Store file, 261

.py files, 16

 __init__.py file, 185–189, 193, 198, 221, 224, 236

 passing variables to, 46–48

 setup.py file, 82, 185–189

 study drills for, 18, 21, 24

" (double-quote), 15, 38, 31, 33, 34–35, 38–40, 51

""" (triple-double-quote), 36–37

= (single-equal), 25–26, 78–79

== (double-equal), 25, 93–94, 96–100, 125

/ (forward slash), 40

 file paths, 252

 operator, 20–22, 125

// (double forward slash), 125–126

% (modulus/percent), 20, 22. *See also* formats *under* strings

%d string format, 29, 31, 124, 127

%i string format, 22, 124, 127

%r string format, 29–31, 34–35, 37, 40, 44

%s string format, 26, 29, 31, 34–35, 40, 43

(octothorpe), 15–16, 18–19. *See also* commenting

( ) (parentheses), 22, 67, 97

' (single-quote), 30–31, 33–35, 38–41, 43

''' (triple-single-quote), 40–41

_ (underscore), 24–25, 68, 181

__ (double underscore), 56

__init__.py file, 185–189, 193, 198, 221, 224, 236

__template__(), 224

A

ActiveState Python, 7–8

addresses, web. *See* URL (uniform resource locator)

Adventure, 50–51, 157

algorithms, 175, 232

Apache, 223

append() function, 107–108, 128

arguments, 46, 62, 70–72, 79, 181

 *args (asterisk args), 66–68

 argv (variables), 46–48, 50–51, 66–68

 command line, 48, 51, 247–248

 def command, 75

 errors involving, 31, 224

 lists and, 128–129

 none, 224, 264

 raw_input() vs., 48

arrays. *See* lists

ASCII characters, 15, 28, 35, 39

assert_equal() function, 191–192

assert_raises() function, 204

assert_response() function, 221–223

attributes, 143–144, 150, 154. *See also* classes

B

base64 library, 231–232
Bash (Linux), 6, 89, 246, 276
`bin` folder, 189
  in a game engine, 232–235
  in a "Hello World" project, 207–212
  in a HTML form project, 216–224
  in a skeleton project directory, 185–189
boolean logic, 31, 92–98
  boolean algebra vs., 94
  common questions about, 94, 98
  `elif` (`else-if` statements) and, 102–103
  `exit(0)` and, 118
  `if-statements` and, 106
  nested decisions, 104–105
  practice, 96–99
  testing and, 120
  Truth Tables, 93
  `while-loops` and, 110

C

C (programming language), 237
"camel case," 181
cascading style sheets (CSS), 211, 216, 223. *See also* layout templates *under* HTML (hypertext markup language)
`cat` command, 63, 247, 272–273
`cd` command, 246–247, 252–256
  errors involving, 16, 208, 212
  pwd command and, 250
character encodings. *See* ASCII; Unicode (UTF-8)
`class` index, 207–209, 219
`class Name(object)` construct, 153, 182
classes, 123, 129–130, 138–148
  attributes and, 144, 148. *See also* attributes
  class hierarchy, 156–159, 175, 177
  class index, 209–210, 219
  coding, 159–161
  composition and, 144, 176–177
  `def` and, 144
  functions and, 138–146, 148, 156–158, 181
  good style, 153, 181
  inheritance and, 144, 148, 153–154, 170–175. *See also* inheritance
  instances and, 144
  modules vs., 139–140
  objects and, 144–145, 150–153, 178
  parameters and, 144–145
  parent/child, 154, 156, 170–173. *See also* inheritance
  `self` and, 144, 154
  testing, 159–161
`close()` function, 56, 58, 62–64
coding
  fixing. *See* debugging
  good style, 181–182
  "hard," 54
  reusable, 177
command line interface (CLI)
  arguments, 48, 51, 247–248
  commands to learn on your own, 276
  crash course (CLI-CC), 243–277
  errors involving, 45, 48, 51, 56, 189
  graphical user interfaces (GUIs) vs., 255
  IDLE vs. 35
  next steps, 276–277
  passing arguments, 47–48
  setup, 244–245
commenting, 18–19, 24–25. *See also* # (octothorpe)
  documentation comments, 88
  good practices in, 21, 25, 72, 118, 127, 182–183
composition, 144, 170, 176–177
config dictionary, 189
connection (web), 208, 215–216
`cp` command, 247, 266–269, 272

CSS (cascading style sheets), 211, 216, 223. *See also* layout templates *under* HTML (hypertext markup language)
C3 algorithm, 175

D

data types, 123
debugging, 34, 120–122
  "debuggers," 121
  log messages, 208
  string formatting for, 31, 34–35, 37, 40, 45
def keyword, 66–68, 75, 123, 140, 224
Dexy, 237
dictionaries. *See* dicts (dictionaries)
dicts (dictionaries), 132–136
  lists vs., 135
  modules as, 138–140
Dijkstra, 115
directories, 246–247. *See also* bin folder
  changing into (cd), 8, 250, 252–256
  command line interface (CLI) commands, 246–265
  errors involving, 16, 198, 208, 212, 224, 236
  Linux, 9–10, 185
  listing the contents of (ls), 256–260
  Mac OSX, 7–8, 185
  making (mkdir), 8, 250–252
  moving around (pushd, popd), 262–265
  print working (pwd), 248–230, 255–256
  project skeleton, 184–189
  removing (rmdir), 260–262
  testing, 221
  Windows, 8–9, 185–186
distribute package, 184, 193
Django, 211, 237

E

elif (else-if statements), 102–105, 120
else statements, 102–103, 120
emacs text editor, 10
"end" statements, 33
errors. *See also* exception handling
  arguments, 31, 224
  ^ (caret), 15
  cd command, 16, 208, 212
  def keyword, 224
  directories, 16, 198, 208, 212, 224, 236
  if-statements, 120, 198
  import command, 198, 234
  ImportError, 89, 189, 198, 212, 224, 236
  int(), 196
  lpthw.web, 16, 209, 212
  modules, 89, 189, 198, 208, 212, 224
  NameError, 25, 52
  nosetests, 189, 193, 236
  objects, 29
  parameters, 47–48
  PowerShell, 8, 16, 56, 264
  PYTHONPATH, 193, 198, 224, 236
  raising, 201
  strings, 31, 64, 37, 40, 43
  SyntaxError, 15–16, 45, 51, 89
  Terminal program, 16, 56, 89
  TypeError, 29, 31
  ValueError, 47–48, 51, 196
escape sequences, 38–41, 124
except keyword, 196
exception handling, 196, 198, 203–204, 196. *See also* errors
exists command, 62, 67
exit command, 275–276
exit() function, 118

F

features. *See* modules
files
  common questions about, 56, 60, 63–64
  copying (cp), 62–64, 266–269
  file mode modifiers, 60
  functions and, 74–76

files (*continued*)
  making empty (touch, new-item), 265–266
  moving (mv), 269–271
  paths, 8, 252
  reading, 54–56, 58–60. *See also* read() function; readline() function
  removing (rm), 273–275
  running Python files, 13
  streaming (cat), 63, 247, 272–273
  study drills for, 55–56, 59–60, 63
  viewing (less, MORE), 271–272
  writing to, 58–60, 62
File Transport Protocol (FTP), 215
"finite state machine," 168
"fixing stuff." *See* refactoring
floating point numbers, 21, 25–26, 29, 80
float(raw_input()), 80
flow charts, 127
folders. *See* directories
for-loops, 106–108, 110
  rules for, 120
  while-loops vs., 112
freecode.com, 83
FTP (File Transport Protocol), 215
"functional programming," 130
functions, 55–56, 66–68, 126–127
  branches and, 116–118
  calling, 71–72, 128
  checklist for, 68
  classes and, 138–146, 148, 156–158, 181
  common questions about, 68, 71–72, 75–76, 80, 118
  composition and, 176–177
  creating, 66
  files and, 74–76
  good style, 68, 181
  if-statements and, 102, 104, 106, 120
  inheritance and, 171–173, 175
  lists and, 128–130
  loops and, 106, 110–112
  match and peek, 200–201
  modules and, 138–143
  returning values from, 78–80
  study drills for, 68, 71, 75, 80–81, 118
  testing and, 196, 204, 221–223
  variables and, 70–72, 85, 86–89

G

game design, 120–122, 180–183
  common questions about, 168, 236
  evaluating, 180–183
  game engines, 157–161, 232–235
  input. *See* user input
  skeleton code for, 162–167
  study drills for, 168, 235
  web-based games, 226–230
gedit text editor, 9–11, 271
GET method, 207–211, 214, 219, 223
github.com, 83
gitorious.org, 83
global keyword, 122, 182
GNOME Terminal, 9
graphical user interface (GUI), 255–256, 269. *See also* command line interface (CLI)

H

"hard coding," 54
has-a relationships, 144–145, 150–154, 176
hashes (#). *See* # (octothorpe)
has-many relationships, 153
"Hello World," 207–209
help() function, 88
HTML (hypertext markup language)
  basic templates, 209–212
  cascading style sheets (CSS) and, 216
  forms, 214–219, 221–224
  layout templates, 220–221
  web-based game engines, 232–235
http (Hyper-Text Transport Protocol), 208, 215, 223, 231

I
IDLE, 16
if-else statements, 198, 229
if-statements, 100–106, 120, 122
  loops and, 106, 110
  rules for, 120
import command
  errors involving, 198, 234. *See also* ImportError
  files, 62–63, 86–90
  modules, 138–141
  packages, 46–47, 56
  practice, 145–146, 180, 193, 198
ImportError, 89, 189, 198, 212, 224, 236
"increment by" operator, 101
infinite loops, 118
inheritance, 144–145, 170–178
  altering before or after, 172–175
  composition vs. 176–177
  explicitly overriding, 172, 174–175
  implicit, 171, 174–175
  multiple, 154, 175. *See also* super() function
input(), 42–43. *See also* user input
instances, 144–146, 150–151
  attributes and, 143–144, 150, 154
  inheritance and, 170–172, 209
  in practice, 209
  self and, 143
int(), 43, 48, 72, 118, 196. *See also* raw_input()
int(raw_input()), 43, 80
is-a relationships, 144–145, 150–154, 176

J
JavaScript, 178, 216, 238
join() function, 129–131

K
keywords, 122–123, 198
  def keyword, 66–68, 75, 123, 140, 224
  except, 196
  global, 122, 182
  raise, 123, 203
  self, 140–146, 148, 154, 173
  try, 196
Kivy, 237
Konsole, 9

L
launchpad.net, 83
learning
  ignoring other programmers, 10, 115, 180, 182, 207, 243, 246
  index cards for, 68, 92, 122, 244, 246–248, 252, 276
  overthinking, 48, 269
  practicing the hard way, 1–4, 11
  reading code, 82–83, 126–127
  reviewing, 81, 84–89, 122–127
  self-learning, 122–127, 276
  testing your knowledge, 90–91, 188
len() function, 62, 64
less command, 271–272
lexicons, 194–198, 200–204
Linux, 6
  command line commands, 245–247, 271
  installing packages on, 9–10, 184–185, 206
  setting up Python on, 9–11
  Terminal, 245–246
lists
  accessing elements of, 114–116
  arguments and, 128–129
  arrays vs., 108
  colons in, 148
  commas in, 30, 68, 106
  common questions about, 108, 130–131
  dicts (dictionaries) vs., 135
  functions and, 128–130
  indexing into, 132

lists (*continued*)
  loops and, 106–109
  manipulating, 128–131
  ordering, 114
  slice syntax, 148
  study drills for, 108, 115, 130
  tuples, 195–196, 200–202
  2-dimensional (2D), 108
localhost, 208, 211, 215, 217, 219
logic. *See* boolean logic
look up tables. *See* `dicts` (dictionaries)
loops
  `for-loops`, 106–110, 112, 120
  functions and 106, 110–112
  `if-statements` and, 106, 110
  infinite, 118
  lists and, 106–109
  rules for, 120
  tuples and, 200
  `while-loops`, 110–112, 126, 128, 130, 161
`lpthw.web`
  dynamic web pages and, 207–211, 231
  errors involving, 16, 209, 212
  HTML forms and, 221–223
  installing, 206–207
`ls` command, 187, 246–247, 256–260

M

Mac OSX
  command line commands, 245–247
  `.DS_Store` file, 261
  installing packages on, 6–7, 184–185, 206
  setting up Python on, 6–7, 10
match and peek, 200–201
`match()` function, 201
math, 20–22, 125–126. *See also* numbers; operators
  %d string format, 29, 31, 124, 127
  `%i` string format, 22, 124, 127
meshes (#). *See* # (octothorpe)
method resolution order (MRO), 175
`mkdir` command, 185–186, 246–247, 250–252, 265
modules, 46–48, 138–143
  classes and, 139–140
  composition and, 176–177
  `dicts` (dictionaries) and, 138–139
  errors involving, 89, 189, 198, 208, 212, 224
  functions and, 138–143
  in practice, 87–89, 194–198, 202
  installing new, 184–185, 188–189
  variables from, 182
`MORE` command, 271–272
`mv` command, 247, 269–271

N

`NAME` module, 188–189
`NameError`, 25, 52
Natural Language Tool Kit, 237
"nested" decisions, 104–105
`new-item` command, 265–266
new line character, 37, 38
Nginx, 223
`nose` package, 184, 204
`nosetests`, 187–189, 191–193, 222, 226, 236
`Notepad++` text editor, 7–8, 10–11, 13
numbers, 20–22. *See also* math
  as a data type, 123
  `dicts` (dictionaries) and, 132–133
  exceptions and, 196
  floating point, 21, 25–26, 29, 80
  indexing into a list with, 132
  ordinal vs. cardinal, 114–115
  ranges of, 105, 108
  rounding down, 22, 29
  user input of, 43

O

object-oriented programming (OOP), 130, 138, 142, 144–148

analysis and design, 154–168
as "functional programming," 130
inheritance in, 170, 177
top-down vs. bottom-up design processes, 161–162
objects, 138–143, 144–145
classes and, 144–145, 150–153, 178
creating, 140
errors involving, 29
as "mini-imports," 140–141
object maps, 156, 158–159
rendering, 211
self and, 144
open() function, 54, 56, 60, 64
operators, 22, 98, 125–126
"increment by," 101
order of operations, 22
space around, 26

P

packages
import command, 46–47, 56
installing, 9–10, 184–185, 206
lpthw.web. *See* lpthw.web
nose. *See* nose package
pip, 184, 188, 193, 206, 212
sys package, 56. *See also* argv
virtualenv, 184
Panda3D, 237
Pandas, 237
parameters, 46–48, 148, 211–212
argv, 67
classes and, 144–145
errors involving, 47–48
file handling and, 58–59
passing information as, 218–219, 222–223
raw_input(), 54–55, 118
syntax, 204
parent/child classes, 154, 156, 170–173. *See also* inheritance
passing information
using parameters, 218–219, 222–223
variables, 46–48
peek() function, 200–202
pickle library, 231–232
pip package, 184, 188, 193, 206, 212
pop() function, 86, 89, 181
popd command, 246–247, 262–265
POST method, 218–223
pound sign (#). *See* # (octothorpe)
PowerShell
errors involving, 8, 16, 56, 264
references for, 277
setting up, 6, 7–9, 13, 245–246
print function, 24, 28–29, 32–37
commas in, 42, 76, 195
common questions about, 33–36
study drills for, 32–34, 36
programmers
%r string format, 29–31, 34–35, 37, 40–44
advice from a veteran, 241–242
ignoring other, 10, 115, 180, 182, 207, 243, 246
resources for, 237–238
specific use of numbers, 114–115
programming
"functional," 130. *See also* object-oriented (OOP) programming
other languages, 238–239. *See also* C (programming language); Django; JavaScript; Ruby
project design, 120–122. *See also* game design
common questions about, 189
creating a skeleton project directory, 185–188
installing packages, 184–185
object-oriented programming (OOP) analysis and, 154–168
testing your setup, 187–188
top-down vs. bottom-up, 161–162

pushd command, 246–247, 262–265
pwd command, 246–247, 248–256
pydoc command, 44–45, 54, 56, 74
PyGame, 237
Python
 ActiveState, 7–8
 first program in, 12–16
 packages. *See* packages
 setting up, 6–11. *See also specific operating systems*
 versions to use, 9, 35
PYTHONPATH, 193, 198, 224, 233–234, 236

Q

quit() command, 8–9, 56

R

raise keyword, 123, 203
raising exceptions, 196, 201–204
range() function, 105, 107–108, 112, 130
raw_input() function, 42–45, 48, 72, 80, 118, 195. *See also* int()
read() function, 54–56, 64
readline() function, 74–76, 89
refactoring, 226–230. *See also* debugging
relationships, 144–145, 150–154, 176. *See also* composition; inheritance
Ren'Py, 237
render.index() function, 210–211, 217, 219
rm command, 246, 261, 273–275
rm -rf /, 246
rmdir command, 246–247, 260–262
round() function, 29
Ruby, 108, 238

S

SciKit-Learn, 237
SciPy, 237
ScraPy, 237
seek() function, 74–76
self keyword, 140–146, 148, 154, 173
sentences, 195, 200–204
servers, 208, 214–216, 219, 222–223
sessions (users), 231–232
setup.py file, 82, 185–189, 206
SimpleCV, 237
skip() function, 201–203
sourceforge.net, 83
strings, 30–31, 38–41
 as arguments, 62
 character encoding. *See* ASCII; Unicode (UTF-8)
 errors involving, 31, 64, 37, 40, 43
 escape sequences, 38–41, 124
 formats, 28–31, 34, 37, 40, 51, 124–125, 127
 string literal, 15, 64
sudo command, 206, 247, 276
super() function, 154, 173–175
SyntaxError, 15–16, 45, 51, 64, 89
sys package, 56. *See also* argv
system PATH, 185, 189

T

temp directory, 250–260, 271–272, 275
Terminal program
 errors involving, 16, 56, 89
 exiting (exit), 275–277
 IDLE vs., 16
 input and, 43, 44–45
 Linux, 9–10, 245
 OSX, 6–7, 14, 245
 Windows. *See* PowerShell
testing
 automated, 190–193
 guidelines for, 192
 HTML forms, 221–224
 test_ functions, 192
 writing test cases, 190–192
text editors, 6–7, 9–12, 18, 271–272
TextWrangler text editor, 6–7, 10–12

thttpd, 223
touch command, 265–266
truncate() function, 58, 60
Truth Tables, The, 93
try-expect construct, 198
try keyword, 196
tuples, 195–196, 200–202
TypeError, 29, 31

U

"underscore format," 181
Unicode (UTF-8), 15–16, 19, 28, 39, 43
Unix, 248–249, 252, 260. *See also* Linux
- Bash references, 276
- cat command, 273
- pushd command, 264
- rmdir command, 266
- skeleton project directory for, 187
- touch command, 266

URL (uniform resource locator), 208–209, 215, 217–219, 222–223
urllib library, 145–146
user input, 42–43
- advanced, 194–198
- browser, 214–219
- common questions, 43, 45, 51–52, 198, 204
- exceptions, 196, 203
- input() function, 42–43
- numbers, 196
- prompting for, 44, 50–52
- scanner, 195–198
- study drills for, 43, 45–46, 51, 198, 204
- tracking sessions, 231–232

UserWarning, 193

V

ValueError, 47–48, 51, 196
variables, 24–26, 28–29
- arguments and, 46–48, 50–51, 66–68
- common questions about, 25–26, 29, 71–72
- declarations, 224
- functions and, 70–72, 85, 86–89
- global, 72
- modules and, 182
- naming, 29, 30
- passing to Python files, 46–48
- representation of, 34. *See also* %r string format
- study drills for, 25, 29, 71

vim text editor, 10
virtualenv package, 184

W

web.py files, 207, 212, 277
websites, 206–224
- HTML forms, 214–224
- HTML templates, 209–211, 220–221
- web requests, 214–216

while-loops, 110–112, 120, 126, 128, 130, 161
Windows
- command line interface (CLI), 245–248
- directories in, 8–9, 185–186
- installing packages on, 7–9, 184–185, 206
- PowerShell. *See* PowerShell
- setting up Python on, 7–9

write() function, 58–59

X

xterm, 9

Y

"yak shaving," 184

Z

Zork, 50–51, 157
zsh (Z shell), 246