Learn to Program with Python 3

A Step-by-Step Guide to Programming

Second Edition

Irv Kalb

Apress®

Learn to Program with Python 3

A Step-by-Step Guide to Programming

Second Edition

Irv Kalb

Apress®

Learn to Program with Python 3

Irv Kalb
Mountain View, California, USA

ISBN-13 (pbk): 978-1-4842-3878-3 ISBN-13 (electronic): 978-1-4842-3879-0
https://doi.org/10.1007/978-1-4842-3879-0

Library of Congress Control Number: 2018954633

Managing Director, Apress Media LLC: Welmoed Spahr
Acquisitions Editor: Todd Green
Development Editor: James Markham
Coordinating Editor: Jill Balzano

Cover designed by eStudioCalamar

Cover image designed by Freepik (www.freepik.com)

Distributed to the book trade worldwide by Springer Science+Business Media New York, 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax (201) 348-4505, e-mail orders-ny@springer-sbm.com, or visit www.springeronline.com. Apress Media, LLC is a California LLC and the sole member (owner) is Springer Science + Business Media Finance Inc (SSBM Finance Inc). SSBM Finance Inc is a **Delaware** corporation.

For information on translations, please e-mail rights@apress.com, or visit www.apress.com/rights-permissions.

Apress titles may be purchased in bulk for academic, corporate, or promotional use. eBook versions and licenses are also available for most titles. For more information, reference our Print and eBook Bulk Sales web page at www.apress.com/bulk-sales.

Any source code or other supplementary material referenced by the author in this book is available to readers on GitHub via the book's product page, located at www.apress.com/9781484238783. For more detailed information, please visit www.apress.com/source-code.

Printed on acid-free paper

This book is dedicated to the memory of my mother, Lorraine Kalb.

*I started learning about programming when I was 16 years old,
at Columbia High School in Maplewood, New Jersey.
We were extremely fortunate to have a very early computer,
an IBM 1130, that students could use.*

I remember learning the basics of the Fortran programming language and writing a simple program that would add two numbers together and print the result. I was thrilled when I finally got my program to work correctly. It was a rewarding feeling to be able to get this huge, complicated machine to do exactly what I wanted it to do.

I clearly remember explaining to my mother that I wrote this program that got the computer to add 9 and 5 and come up with an answer of 14. She said that she didn't need a computer to do that. I tried to explain to her that getting the answer of 14 was not the important part. What was important was that I had written a program that would add any two numbers and print the result. She still didn't get it, but she was happy for me and very supportive.

Hopefully, through my explanations in this book, you will get it.

Table of Contents

About the Author xiii

About the Technical Reviewer xv

Acknowledgments xvii

Chapter 1: Getting Started 1

- What Is Python? 2
- Installing Python 2
- IDLE and the Python Shell 3
- Hello World 4
- Creating, Saving, and Running a Python File 6
- IDLE on Multiple Platforms 8
- Summary 9

Chapter 2: Variables and Assignment Statements 11

- A Sample Python Program 12
- The Building Blocks of Programming 14
- Four Types of Data 15
  - Integers 15
  - Floats 15
  - Strings 16
  - Booleans 16
  - Examples of Data 17
  - Form with Underlying Data 18
- Variables 19
- Assignment Statements 24

Variable Names 27
  Naming Convention 28
  Keywords 29
  Case Sensitivity 31
  More Complicated Assignment Statements 32
Print Statements 33
Simple Math 35
Order of Operations 38
First Python Programs 39
Shorthand Naming Convention 41
Adding Comments 43
  Full-Line Comment 43
  Add a Comment After a Line of Code 44
  Multiline Comment 44
Whitespace 45
Errors 46
  Syntax Error 46
  Exception Error 48
  Logic Error 49
Summary 49

Chapter 3: Built-in Functions 51

Overview of Built-in Functions 51
Function Call 52
Arguments 52
Results 53
Built-in type Function 53
Getting Input from the User 55
Conversion Functions 56
  int Function 57
  float Function 57
  str Function 57

First Real Programs 58
Concatenation 61
Another Programming Exercise 62
Using Function Calls Inside Assignment Statements 63
Summary 65

Chapter 4: User-Defined Functions 67

A Recipe as an Analogy for Building Software 68
Ingredients 68
Directions 68
Definition of a Function 71
Building Our First Function 72
Calling a User-Defined Function 73
Receiving Data in a User-Defined Function: Parameters 76
Building User-Defined Functions with Parameters 78
Building a Simple Function That Does Addition 81
Building a Function to Calculate an Average 81
Returning a Value from a Function: The return Statement 82
Returning No Value: None 84
Returning More Than One Value 85
Specific and General Variable Names in Calls and Functions 86
Temperature Conversion Functions 88
Placement of Functions in a Python File 89
Never Write Multiple Copies of the Same Code 90
Constants 91
Scope 93
Global Variables and Local Variables with the Same Names 97
Finding Errors in Functions: Traceback 98
Summary 101

Chapter 5: if, else, and elif Statements 103
Flowcharting 104
The if Statement 107
Comparison Operators 109
Examples of if Statements 109
Nested if Statement 111
The else Statement 111
Using if/else Inside a Function 114
The elif Statement 115
Using Many elif Statements 118
A Grading Program 120
A Small Sample Program: Absolute Value 120
Programming Challenges 123
Negative, Positive, Zero 123
isSquare 125
isEven 128
isRectangle 130
Conditional Logic 132
The Logical not Operator 132
The Logical and Operator 133
The Logical or Operator 135
Precedence of Comparison and Logical Operators 136
Booleans in if Statements 136
Program to Calculate Shipping 137
Summary 141
Chapter 6: Loops 143
User's View of the Game 144
Loops 145
The while Statement 147
First Loop in a Real Program 150

Increment and Decrement 151
Running a Program Multiple Times 152
Python's Built-in Packages 154
Generating a Random Number 155
Simulation of Flipping a Coin 157
Other Examples of Using Random Numbers 158
Creating an Infinite Loop 160
A New Style of Building a Loop: while True, and break 160
Asking If the User Wants to Repeat: the Empty String 163
Pseudocode 164
Building the Guess the Number Program 164
Playing a Game Multiple Times 171
Error Detection with try/except 173
The continue Statement 175
Full Game 176
Building Error-Checking Utility Functions 178
Coding Challenge 179
Summary 181

Chapter 7: Lists 183

Collections of Data 184
Lists 185
Elements 185
Python Syntax for a List 186
Empty List 187
Position of an Element in a List: Index 187
Accessing an Element in a List 189
Using a Variable or Expression as an Index in a List 190
Changing a Value in a List 192
Using Negative Indices 192
Building a Simple Mad Libs Game 193

Adding a List to Our Mad Libs Game 195
Determining the Number of Elements in a List: The len Function 196
Programming Challenge 1 198
Using a List Argument with a Function 200
Accessing All Elements of a List: Iteration 203
for Statements and for Loops 204
Programming Challenge 2 206
Generating a Range of Numbers 207
Programming Challenge 3 208
Scientific Simulations 209
List Manipulation 214
List Manipulation Example: an Inventory Example 216
Pizza Toppings Example 217
Summary 223

Chapter 8: Strings 225

len Function Applied to Strings 226
Indexing Characters in a String 226
Accessing Characters in a String 227
Iterating Through Characters in a String 228
Creating a Substring: A Slice 230
Programming Challenge 1: Creating a Slice 232
Additional Slicing Syntax 235
Slicing as Applied to a List 236
Strings Are Not Changeable 236
Programming Challenge 2: Searching a String 237
Built-in String Operations 238
Examples of String Operations 240
Programming Challenge 3: Directory Style 241
Summary 243

Chapter 9: File Input/Output 245
Saving Files on a Computer 246
Defining a Path to a File 247
Reading from and Writing to a File 249
File Handle 250
The Python os Package 251
Building Reusable File I/O Functions 252
Example Using Our File I/O Functions 254
Importing Our Own Modules 255
Saving Data to a File and Reading It Back 257
Building an Adding Game 260
Programming Challenge 1 260
Programming Challenge 2 261
Writing/Reading One Piece of Data to and from a File 263
Writing/Reading Multiple Pieces of Data to and from a File 266
The join Function 266
The split Function 267
Final Version of the Adding Game 268
Writing and Reading a Line at a Time with a File 271
Example: Multiple Choice Test 275
A Compiled Version of a Module 281
Summary 282

Chapter 10: Internet Data 283
Request/Response Model 283
Getting a Stock Price 285
Pretending to Be a Browser 286
API 288
Requests with Values 288
API Key 289
Example Program to Get Stock Price Information Using an API 291

Example Program to Get Weather Information 294
URL Encoding 297
Summary 299

Chapter 11: Data Structures 301
Tuples 302
Lists of Lists 305
Representing a Grid or a Spreadsheet 306
Representing the World of an Adventure Game 307
Reading a Comma-Separated Value (.csv) File 311
Dictionary 316
Using the in Operator on a Dictionary 319
Programming Challenge 320
A Python Dictionary to Represent a Programming Dictionary 322
Iterating Through a Dictionary 323
Combining Lists and Dictionaries 325
JSON: JavaScript Object Notation 328
Example Program to Get Weather Data 331
XML Data 334
Accessing Repeating Groupings in JSON and XML 338
Summary 341

Chapter 12: Where to Go from Here 343
Python Language Documentation 343
Python Standard Library 344
Python External Packages 345
Python Development Environments 346
Places to Find Answers to Questions 347
Projects and Practice, Practice, Practice 347
Summary 348

Index 349

About the Author

Irv Kalb is an adjunct professor at UCSC (University of California, Santa Cruz) Extension Silicon Valley and Cogswell Polytechnical College. He has been teaching software development classes since 2010.

Irv has worked as a software developer, manager of software developers, and manager of software development projects. He has been an independent consultant for many years with his own company, Furry Pants Productions, where he has concentrated on educational software. Prior to that, he worked as an employee for a number of high-tech companies. He has BS and MS degrees in computer science.

Recently, he has been a mentor to a number of local competitive robotics teams.

His previous publications include numerous technical articles, two children's edutainment CD-ROMs (about Darby the Dalmatian), an online e-book on object-oriented programming in the Lingo programming language, and the first book on Ultimate Frisbee, *Ultimate: Fundamentals of the Sport* (Revolutionary Publications, 1983).

He was highly involved in the early development of the sport of Ultimate Frisbee.

About the Technical Reviewer

Mark Furman, MBA is a systems engineer, author, teacher, and entrepreneur. For the last 16 years he has worked in the information technology field with a focus on Linux-based systems and programming in Python. He's worked for a range of companies including Host Gator, Interland, Suntrust Bank, AT&T, and Winn-Dixie. Currently he has been focusing his career on the maker movement and has launched Tech Forge (techforge.org), which focuses on helping people start a makerspace and help sustain current spaces. He holds an MBA degree from Ohio University. You can follow him on Twitter `@mfurman`.

Acknowledgments

I would like to thank the following people, without whom this book would not have been possible:

My wonderful wife, Doreen, who is the glue that keeps our family together.

Our two sons, Jamie and Robbie, who keep us on our toes.

Our two cats, Chester and Cody (whom we think of as people).

Mariah Armstrong, who created all the graphics in this book. I am not an artist (I don't even play one on TV). Mariah was able to take my "chicken scratches" and turn them into very clear and understandable pieces of art.

Chris Sasso and Ravi Chityala for their technical reviews and helpful suggestions.

Luke Kwan, Catherine Chanse, and Christina Ri at the Art Institute of California-Silicon Valley.

Andy Hou at the UCSC-Silicon Valley Extension.

Jerome Solomon at Cogswell Polytechnical College, who first suggested that I consider getting into Python.

Jill Balzano, Jim Markham, Mark Furman, and Todd Green at Apress for all the work they did reviewing, editing, and expertly answering all my questions.

All the students who have been in my classes over many years at the Art Institute California-Silicon Valley, Cogswell Polytechnical College, and the UCSC Silicon Valley Extension. Their feedback, suggestions, smiles, frowns, light-bulb moments, frustrations, and knowing head-nods were extremely helpful in shaping the content of this book.

Finally, Guido van Rossum, without whom Python would not exist.

CHAPTER 1

Getting Started

Congratulations! You have made a wise decision. No, not the decision to buy this book, although I think that will turn out to be a wise decision also. I mean you have made a wise decision to learn the basics of computer programming using the Python language.

In this book, I teach you the fundamentals of writing computer software. I assume that you have never written any software before, so I start completely from scratch. The only requirements are that you possess a basic knowledge of algebra and a good sense of logic. As the book progresses, each chapter builds upon the information learned in the previous chapter(s). The overall goal is to give you a solid introduction to the way that computer code and data interact to form well-written programs. I introduce the key elements of software, including variables, functions, `if/else` statements, loops, lists, and strings. I offer many real-world examples that should help explain the uses of each of these elements. I also give definitions to help you with the new vocabulary that I introduce.

This book is not intended to be comprehensive. Rather, it is an introduction that gives you a solid foundation in programming. The approach is highly interactive, asking you to create small programs along the way as a chance to practice what has been explained in each chapter. By the end of the book, you should be comfortable writing small to medium-sized programs in Python.

This first chapter covers the following topics:

- Introducing Python
- Getting Python installed on your computer
- Using IDLE and the Python Shell
- Writing your first program: Hello World
- Creating, saving, and running Python files
- Working with IDLE on multiple platforms

I. Kalb, *Learn to Program with Python 3*, https://doi.org/10.1007/978-1-4842-3879-0_1

What Is Python?

Python is a general-purpose programming language. That means it was designed and developed to write software for a wide variety of disciplines. Python has been used to write applications to solve problems in biology, chemistry, financial analysis, numerical analysis, robotics, and many other fields. It is also widely used as a *scripting language* for use by computer administrators, who use it to capture and replay sequences of computer commands. It is different from a language like HTML (HyperText Markup Language), which was designed for the single purpose of allowing people to specify the layout of a web page.

Once you learn the basic concepts of a programming language like Python, you find that you can pick up a new computer languages very quickly. No matter what the language (and there are many) the underlying concepts are very similar. The key things that you learn about—variables, assignment statements, `if` statements, `while` loops, function calls—are all concepts that are easily transferable to any other programming language.

Installing Python

Python was created in the 1990s by Guido van Rossum. He is affectionately known as Python's Benevolent Dictator for Life. The language has two current versions: 2.7 and 3.6. Version 2.7 is still widely used, but its "end of life" has recently been announced. Therefore, this version of the book will use the newer *Python 3*, as it is known. With respect to the contents of this book, there are only a few differences between the versions of the language. Where appropriate, I point out how something presented in Python 3 was handled in Python 2.

Python is maintained as an *open source* project by a group called the Python Software Foundation. Because it is open source, Python is free. There is no single company that owns and/or sells the software. You can get everything you need to write and run all the Python programs in this book by simply downloading Python from the Internet. I'll explain how you can get it and install it.

The center of the Python universe is at `www.python.org`.

Bring up the browser of your choice and go to that address. The site changes over time, but the essential functionality should remain the same. On the main page, there should be a Downloads button or rollover. Once you're in the Downloads area, you should be able to select Windows, Mac, or Other Platforms (which includes Linux). After choosing your operating system, you should get an opportunity to choose between versions 3.x.y (whatever is the current subversion of Python 3) and version 2.x.y (whatever is the current subversion of Python 2). Choose version 3.x.y.

Clicking the button downloads an installer file. On a Mac, the downloaded file has a name like `python-3.6.4-macosx10.6pkg`. On a Windows computer, the file has a name like `python-3.6.4-msi`. On either platform, find the file that was downloaded and double-click it. That should start the installation process, which should be very simple.

IDLE and the Python Shell

There are many different *software development environments* (applications) that you can use to write code in Python. It may seem odd that you use a program to write a program, but that's what a software development environment is. Some of these environments are free; others can be costly. They differ in the tools they offer to help programmers be more efficient.

The environment we will use in this book is called IDLE. You might think that IDLE is an acronym, maybe Interactive DeveLopment Environment. When the name was chosen, it didn't mean anything. In fact, the name Python doesn't refer to the snake. Apparently, Guido van Rossum was a big fan of *Monty Python's Flying Circus*, a TV series by a well-known comedy group from Britain, and he named the language after them. One of the founding members of Monty Python was Eric Idle. The name IDLE is a reference to him.

IDLE is free. When you download and run the Python installer, it installs IDLE on your computer. Once installed, you can find IDLE on a Mac by opening the applications folder and locating the folder named Python 3.x. Once you open it, you should see the IDLE application. To open IDLE, double-click the icon. On Windows, IDLE is installed in the standard Program Files folder. If your version of Windows has a Start button, click the Start button and type **IDLE** in the type-in field. Otherwise, you might have to do a Control+R or Control+Q to bring up a dialog box where you can type **IDLE**. However you open IDLE, you should see a window with contents that look something like this:

```
Python 3.6.1 (v3.6.1:69c0db5050, Mar 21 2017, 01:21:04)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "copyright", "credits" or "license()" for more information.
>>>
```

This window is called the Python Shell. In fact, the title of the window should be Python 3.x.y Shell.

Hello World

There is a tradition that when programmers learn a new computer language, they try writing what is called the Hello World program. That is, just to make sure they can get something to work, they write a simple program that writes out "Hello World!"

Let's do that now with Python. The Python Shell (commonly just called the Shell) gives you a prompt that looks like three greater-than signs. This is called the *chevron prompt* or simply the *prompt*. When you see the prompt, it means the Shell is ready for you to type something. Throughout this book, I strongly encourage you to use the IDLE environment by trying out code as I explain it. At the prompt, enter the following:

```
>>> print('Hello World!')
```

Then press the Return key or Enter key. When you do, you should see this:

```
>>> print('Hello World!')
Hello World!
>>>
```

Congratulations! You have just written your first computer program. You told the computer to do something, and it did exactly what you told it to do. My work is done here. You're not quite ready to add *Python programmer* to your résumé and get a job as a professional computer programmer, but you are off to a good start!

Note If you don't like the font and/or size of the text used in the Shell, you can choose IDLE ➤ Preferences (Mac) or Configure IDLE (Windows) and easily change either or both.

One of the key advantages of the Python language is how readable it is. The program you just wrote is simply the word `print`, an open parenthesis, whatever you want to be printed (inside quotes), and a closing parenthesis. Anyone can understand the Hello World program written in Python. But to make this point very clear, let's see what you have to do to write the Hello World program in some other popular languages.

You've probably heard of the language called C, perhaps the most widely used programming language in the world. Here is what you have to write in C to get the same results:

```
#include <stdio.h>

int main(void)
{
  printf("Hello World!\n");
  return 0;
}
```

Notice all the brackets, parentheses, braces, and semicolons you need to have, along with how many lines you have to write?

There is another language called C++, which is a modification of the original C language to give it more power. Here's what the Hello World program looks like in C++:

```
#include "std_lib_facilities.h"

int main()
{
        cout << "Hello World!\n";
        return 0;
}
```

Not surprisingly, it also has many brackets, parentheses, braces, and semicolons.

Finally, here is the same Hello World program written in Java, yet another popular computer language:

```
public class HelloWorld {
   public static void main(String[] args) {
      System.out.println("Hello World!");
   }
}
```

Again, there are many brackets, parentheses, and semicolons, and many words with meanings that are not immediately obvious.

By comparison, notice how English-like, simple, and readable the Python version is. This readability and simplicity are big reasons why Python is growing in popularity, especially as a language used to teach programming to beginners.

Creating, Saving, and Running a Python File

So far, you have only seen a single line of Python code:

```
>>> print('Hello World!')
```

You typed it into the Shell and pressed Enter or Return to make it run. Typing one line at a time into the Shell is a great way to learn Python, and it is very handy for trying out things quickly. But soon I'll have you writing programs with tens, hundreds, and maybe thousands of lines of code. The Shell is not an appropriate place for writing large programs. Python, like every other computer language, allows you to put the code you write into a file and save it. Programs saved this way can be opened at any time and run without having to retype them. I'll explain how we do this in Python.

Just like any standard word processor or spreadsheet program, to create a new file in IDLE, you go to the File menu and select New File (denoted from here on as File ➤ New File). You can also use the keyboard shortcuts Control+N (Windows) or Command+N (Mac).

This opens a new, blank editing window, waiting for you to enter Python code. It behaves just like any text editing program you have ever used. You enter your Python code, line by line, similar to the way that you did it in the Shell. However, when you press Return or Enter at the end of a line, the line does not run—it does *not* produce immediate results as it did in the Shell. Instead, the cursor just moves down to allow you to enter another line. You can use all the standard text-editing features that you are used to: Cut, Copy, Paste, Find, Replace, and so on. You can move around the lines of code using the arrow keys or by clicking the mouse. When a program gets long enough, scrolling becomes enabled. You can select multiple lines using the standard click-and-drag or click to create a starting point and Shift-click to mark an ending point.

Let's build a simple program containing three `print` statements. Open a new file. Notice that when you open the file, it is named `Untitled` in the window title. Enter the following:

```
print('I am now entering Python code into a Python file.')
print('When I press Return or Enter here, nothing happens.')
print('This is the last line.')
```

When you type the word **print**, IDLE colorizes it (both here in the editing window and when you type it in the Shell). This is IDLE letting you know that this is a word that it recognizes. IDLE also turns all the words enclosed in quotes to green. This also is an acknowledgement from IDLE that it has an understanding of what you are trying to say.

Notice that when you started typing, the window title changed to `*Untitled*`. The asterisks around the name are there to show that the contents of the file have been changed, but the file has not been saved. That is, IDLE knows about the new content, but the content has not yet been written to the hard disk. To save the file, press the standard Control+S (Windows) or Command+S (Mac). Alternatively, you can click File ➤ Save. Because this is the first time the file is being saved, you see the standard Save dialog box. Feel free to navigate to a folder where you are able to find your Python files(s), or click the New Folder button to create a new folder. In the top of the box, where it says "Save As", enter a name for this file. Because we are just testing things out, you can name the file `Test`. However, Python filenames should always end with a `.py` extension. Therefore, you should enter the name **Test.py** in the Save As box.

Note If you save your Python file without a `.py` extension, IDLE will not recognize it as a Python file. If Python does not know that your file is a Python file, it will not colorize your code. This may not seem important now, but it will turn out to be very helpful when you start writing larger programs. So make it a habit right from the start to always ensure that your Python file names end with the `.py` extension.

Now that we have a saved Python file, we want to run, or *execute*, the *statements* in the file. To do that, click Run ➤ Run Module or press the F5 shortcut key. If everything went well, the program should print the following in the Shell:

```
I am now entering Python code into a Python file.
When I press Return or Enter here, nothing happens.
This is the last line.
```

Now let's quit IDLE by pressing Control+Q (Windows) or Command+Q (Mac) keys. Alternatively, you can click IDLE ➤ Exit (Windows) or IDLE ➤ Quit IDLE (Mac).

When you are ready to open IDLE again, you have choices. You can open IDLE by typing **IDLE** into the Start menu (Windows) or by double-clicking the IDLE icon (Mac). If you then want to open a previously saved Python file, you can click File ➤ Open and navigate to the file you want to open.

However, if you want to open IDLE and open a previously saved Python file, you can navigate to the saved Python file (for example, find the `Test.py` file that you just saved) and open IDLE by opening the file. On Windows, if you double-click the icon, a window typically opens and closes very fast. This runs the Python program, but does not keep the window open. Instead, to open the file and IDLE, right-click the file icon. From the context menu that appears, select the second item, Edit with IDLE.

On a Mac, you can simply double-click the file icon. If double-clicking the Python file opens a program other than IDLE, you can fix that with a one-time change. Quit whatever program opened. Select the Python file. Press Command+I (or click File ➤ Get Info), which opens a long dialog box. In the section labeled "Open with", select the IDLE application (`IDLE.app`). Finally, click the Change All button. Once you do that, you should be able to double-click any file whose name ends in `.py`, and it should open with IDLE.

Programming typically involves iterations of edits to one or more Python files. Each time you make changes and you want to test the new code, you must save the file and then run it. If you don't save the file before you try to run it, IDLE will prompt you by asking you to save the file. You'll quickly become familiar with the typical development cycle of edit your code, save the file (Command+S or Control+S), and run the program (F5).

IDLE on Multiple Platforms

One other very nice feature of Python and IDLE is that the environment is almost completely platform independent. That is, the IDLE environment looks almost identical on a Windows computer, Mac, or Linux system. The only differences are those associated with the particular operating system (such as the look of the window's title bar, the location of the menus, the look of the dialog boxes, and so on). These are very minor details. Overall, the platform you run on does not matter.

Perhaps even more importantly, the code you write is platform independent. If you create a Python file on one platform, you can move that file to another platform and it will open and run just fine. Many programmers use multiple systems to develop Python code. In fact, even though I typically develop most of my Python code on a Mac, I often bring these same files into classrooms, open them, teach with them on Windows systems.

Summary

In this chapter, you got up and running with Python. You should now have Python installed on your computer and have a good understanding of what the IDLE environment is. You built the standard Hello World program in the Shell, and then used the editor window to build, save, and run a simple multiline Python program (whose name ends in `.py`) made up of `print` statements. Finally, you learned that Python and the IDLE environment are platform independent.

CHAPTER 2

Variables and Assignment Statements

This chapter covers the following topics:

- A sample Python program
- Building blocks of programming
- Four types of data
- What a variable is
- Rules for naming variables
- Giving a variable a value with an assignment statement
- A good way to name variables
- Special Python keywords
- Case sensitivity
- More complicated assignment statements
- Print statements
- Basic math operators
- Order of operations and parentheses
- A few small sample programs
- Additional naming conventions
- How to add comments in a program
- Use of "whitespace"
- Errors in programs

I. Kalb, *Learn to Program with Python 3*, https://doi.org/10.1007/978-1-4842-3879-0_2

A Sample Python Program

Let's jump right in and see an example of what Python code looks like. You are probably familiar with a simple toy called the Magic 8-Ball, made by Mattel, Inc. To play with the toy, you ask it a yes-or-no question, turn the ball over, and the ball gives you one of a number of possible answers. Here is the output of a Python program that simulates the Magic 8-Ball:

```
Ask the Magic 8-Ball a question (Return or Enter to quit): Will this be a
great book?
Absolutely!

Ask the Magic 8-Ball a question (Return or Enter to quit): Will I learn to
program in Python?
Answer is foggy, ask again later.

Ask the Magic 8-Ball a question (Return or Enter to quit): Will I learn to
program in Python?
You may rely on it.

Ask the Magic 8-Ball a question (Return or Enter to quit): Will I be able
to play football in the NFL?
No way, dude!

Ask the Magic 8-Ball a question (Return or Enter to quit): Will I make a
million dollars?
Absolutely!

Ask the Magic 8-Ball a question (Return or Enter to quit): Does the Magic
8-Ball ever make mistakes?
No way, dude!

Ask the Magic 8-Ball a question (Return or Enter to quit):
```

Now, let's jump right in and take a look at the underlying code of this program. I'm showing you this just to give you a feeling for what Python code looks like. I am certainly not expecting you to understand much of this code. At this point, the details are unimportant. Here it is:

```
import random  # Allow the program to use random numbers

while True:
    print() # prints a blank line

    usersQuestion = input('Ask the Magic 8-Ball a question
    (press enter to quit): ')
    if usersQuestion == ":
        break   # we're done

    randomAnswer = random.randrange(0, 8)  # pick a random number

    if randomAnswer == 0:
        print('It is certain.')

    elif randomAnswer == 1:
        print('Absolutely!')

    elif randomAnswer == 2:
        print('You may rely on it.')

    elif randomAnswer == 3:
        print('Answer is foggy, ask again later.')

    elif randomAnswer == 4:
        print('Concentrate and ask again.')

    elif randomAnswer == 5:
        print('Unsure at this point, try again.')

    elif randomAnswer == 6:
        print('No way, dude!')

    elif randomAnswer == 7:
        print('No, no, no, no, no.')
```

Here's a very quick explanation: at the top, there is a line that allows the program to use random numbers. Then there is a line that says `while True`. This line creates something called a *loop*, which is a portion of a program that runs over and over again. In this case, it allows the user to ask a question and get an answer, and then enter another question and get another answer, and on and on.

Moving down, there is a line that causes `Ask the Magic 8-Ball a question` to be printed out and allows the user to type a question for the Magic 8-Ball to answer.

Skipping down a few lines, the program generates a random number between 0 and 7. After generating the random number, the program then checks to see if the value of the random number is 0. If so, it tells the user the answer: `It is certain.` Otherwise, if the value of the randomly chosen number is 1, it tells the user: `You may rely on it.`

The rest of the lines work similarly, checking the random number and giving different outputs.

After the program prints an answer, because the program is inside the loop, the program goes around again and tells the user to ask another question. And the process keeps going.

As I said, don't worry about the details of the program—just get a sense of how the program does what it does. But there are some things to notice. First, see how readable this code is. With only this brief introduction, you can probably get a feeling for the basic logical flow of how the program operates. Second, notice that the program asks the user for input, does some computation, and generates some output. These are the three main steps in almost all computer programs.

Let's get into programming 101. This may be extremely basic, but I want to start right at the beginning, create a solid foundation, and then build on that.

The Building Blocks of Programming

The two basic building blocks of programming are code and data. *Code* is a set of instructions that tell the computer what to perform and how it should perform. But I want to start our discussion with data.

Data refers to the quantities, characters, and/or symbols on which operations are performed with a computer. Anything you need the computer to remember is a piece of data. Simple examples of data include the number of students in class, grade point average, name, whether a switch is in an on or off position, and so on.

There are many different types of data, but this book deals mostly with four basic types, which I describe in the next section.

Four Types of Data

The four basic types of data are called *integer numbers*, *floating-point numbers*, *strings*, and *Booleans*. This section explains and provides examples of each of these types of data.

Integers

Integer numbers (or simply, *integers*) are counting numbers, like 1, 2, 3, but also include 0 and negative numbers. The following are examples of data that is expressed as integers:

- Number of people in a room
- Personal or team score in a game
- Course number
- Date in a month
- Temperature (in terms of number of degrees)

Floats

Floating-point numbers (or simply *floats*) are numbers that have a decimal point in them. The following are examples of data that is expressed as floating-point numbers:

- Grade point average
- Price of something
- Percentages
- Irrational numbers, like pi

Strings

Strings (also called *text*) are any sequences of characters. Examples of data that is expressed as strings include the following:

- Name
- Address
- Course name
- Title of a book, song, or movie
- Sentence
- Name of a file on a computer

Booleans

Booleans are a type of data that can only have one of two values: True or False. Booleans are named after the English mathematician George Boole, who created an entire field of logic based on these two-state data items. The following are some examples of data that can be expressed as Booleans:

- *The state of a light switch*: True for on, False for off
- *Inside or outside*: True for inside, False for outside
- *Whether someone is alive or not*: True for alive, False for dead
- *If someone is listening*: True for listening, False for not listening

It might seem that integer and floating-point data have overlaps. For example, there is an integer 0 and there is a floating-point 0.0. There is an integer 1 and a floating-point 1.0. Although these may appear to be the same thing to us humans, integers and floats are handled very differently inside the computer. Without getting too wrapped up in the details, it is easier for the computer to represent and operate with integers. But when we have a value with a decimal point, we need to use a floating-point number instead. Whenever we represent a value, we choose the appropriate numeric data type. As you will see, Python makes a clear distinction between these two types of data.

There are many other types of data in the computer world. For example, you are probably familiar with music being stored in MP3 format or video being stored in MP4. These are other representations of data. However, to make things simple and clear, I'll use just the four basic types of data in most of this book.