

LEARNING MADE EASY



2nd Edition

Beginning Programming with Python[®]

for
dummies[®]
A Wiley Brand



Use Python to create and
run your first application

—
Understand ways to
troubleshoot and fix errors

—
Learn to work with Anaconda
and use Magic Functions

John Paul Mueller



Beginning Programming with Python[®]

2nd Edition

by John Paul Mueller

for
dummies[®]
A Wiley Brand

Beginning Programming with Python® For Dummies®, 2nd Edition

Published by: **John Wiley & Sons, Inc.**, 111 River Street, Hoboken, NJ 07030-5774, www.wiley.com

Copyright © 2018 by John Wiley & Sons, Inc., Hoboken, New Jersey

Published simultaneously in Canada

No part of this publication may be reproduced, stored in a retrieval system or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, scanning or otherwise, except as permitted under Sections 107 or 108 of the 1976 United States Copyright Act, without the prior written permission of the Publisher. Requests to the Publisher for permission should be addressed to the Permissions Department, John Wiley & Sons, Inc., 111 River Street, Hoboken, NJ 07030, (201) 748-6011, fax (201) 748-6008, or online at <http://www.wiley.com/go/permissions>.

Trademarks: Wiley, For Dummies, the Dummies Man logo, Dummies.com, Making Everything Easier, and related trade dress are trademarks or registered trademarks of John Wiley & Sons, Inc. and may not be used without written permission. Python is a registered trademark of Python Software Foundation Corporation. All other trademarks are the property of their respective owners. John Wiley & Sons, Inc. is not associated with any product or vendor mentioned in this book.

LIMIT OF LIABILITY/DISCLAIMER OF WARRANTY: THE PUBLISHER AND THE AUTHOR MAKE NO REPRESENTATIONS OR WARRANTIES WITH RESPECT TO THE ACCURACY OR COMPLETENESS OF THE CONTENTS OF THIS WORK AND SPECIFICALLY DISCLAIM ALL WARRANTIES, INCLUDING WITHOUT LIMITATION WARRANTIES OF FITNESS FOR A PARTICULAR PURPOSE. NO WARRANTY MAY BE CREATED OR EXTENDED BY SALES OR PROMOTIONAL MATERIALS. THE ADVICE AND STRATEGIES CONTAINED HEREIN MAY NOT BE SUITABLE FOR EVERY SITUATION. THIS WORK IS SOLD WITH THE UNDERSTANDING THAT THE PUBLISHER IS NOT ENGAGED IN RENDERING LEGAL, ACCOUNTING, OR OTHER PROFESSIONAL SERVICES. IF PROFESSIONAL ASSISTANCE IS REQUIRED, THE SERVICES OF A COMPETENT PROFESSIONAL PERSON SHOULD BE SOUGHT. NEITHER THE PUBLISHER NOR THE AUTHOR SHALL BE LIABLE FOR DAMAGES ARISING HEREFROM. THE FACT THAT AN ORGANIZATION OR WEBSITE IS REFERRED TO IN THIS WORK AS A CITATION AND/OR A POTENTIAL SOURCE OF FURTHER INFORMATION DOES NOT MEAN THAT THE AUTHOR OR THE PUBLISHER ENDORSES THE INFORMATION THE ORGANIZATION OR WEBSITE MAY PROVIDE OR RECOMMENDATIONS IT MAY MAKE. FURTHER, READERS SHOULD BE AWARE THAT INTERNET WEBSITES LISTED IN THIS WORK MAY HAVE CHANGED OR DISAPPEARED BETWEEN WHEN THIS WORK WAS WRITTEN AND WHEN IT IS READ.

For general information on our other products and services, please contact our Customer Care Department within the U.S. at 877-762-2974, outside the U.S. at 317-572-3993, or fax 317-572-4002. For technical support, please visit <https://hub.wiley.com/community/support/dummies>.

Wiley publishes in a variety of print and electronic formats and by print-on-demand. Some material included with standard print versions of this book may not be included in e-books or in print-on-demand. If this book refers to media such as a CD or DVD that is not included in the version you purchased, you may download this material at <http://booksupport.wiley.com>. For more information about Wiley products, visit www.wiley.com.

Library of Congress Control Number: 2017964018

ISBN 978-1-119-45789-3; ISBN 978-1-119-45787-9 (ebk); ISBN 978-1-119-45790-9 (ebk)

Manufactured in the United States of America

10 9 8 7 6 5 4 3 2 1

Contents at a Glance

Introduction	1
Part 1: Getting Started with Python	5
CHAPTER 1: Talking to Your Computer	7
CHAPTER 2: Getting Your Own Copy of Python	21
CHAPTER 3: Interacting with Python	37
CHAPTER 4: Writing Your First Application	55
CHAPTER 5: Working with Anaconda	83
Part 2: Talking the Talk	101
CHAPTER 6: Storing and Modifying Information	103
CHAPTER 7: Managing Information	113
CHAPTER 8: Making Decisions	135
CHAPTER 9: Performing Repetitive Tasks	151
CHAPTER 10: Dealing with Errors	165
Part 3: Performing Common Tasks	195
CHAPTER 11: Interacting with Packages	197
CHAPTER 12: Working with Strings	225
CHAPTER 13: Managing Lists	243
CHAPTER 14: Collecting All Sorts of Data	261
CHAPTER 15: Creating and Using Classes	281
Part 4: Performing Advanced Tasks	303
CHAPTER 16: Storing Data in Files	305
CHAPTER 17: Sending an Email	323
Part 5: The Part of Tens	341
CHAPTER 18: Ten Amazing Programming Resources	343
CHAPTER 19: Ten Ways to Make a Living with Python	353
CHAPTER 20: Ten Tools That Enhance Your Python Experience	361
CHAPTER 21: Ten (Plus) Libraries You Need to Know About	371
Index	379

Table of Contents

INTRODUCTION	1
About This Book	1
Foolish Assumptions	2
Icons Used in This Book	3
Beyond the Book	3
Where to Go from Here	4
PART 1: GETTING STARTED WITH PYTHON	5
CHAPTER 1: Talking to Your Computer	7
Understanding Why You Want to Talk to Your Computer	8
Knowing that an Application is a Form of Communication	9
Thinking about procedures you use daily	9
Writing procedures down	10
Seeing applications as being like any other procedure	11
Understanding that computers take things literally	11
Defining What an Application Is	11
Understanding that computers use a special language	12
Helping humans speak to the computer	12
Understanding Why Python Is So Cool	14
Unearthing the reasons for using Python	14
Deciding how you can personally benefit from Python	15
Discovering which organizations use Python	16
Finding useful Python applications	17
Comparing Python to other languages	18
CHAPTER 2: Getting Your Own Copy of Python	21
Downloading the Version You Need	21
Installing Python	24
Working with Windows	25
Working with the Mac	27
Working with Linux	28
Accessing Python on Your Machine	31
Using Windows	32
Using the Mac	34
Using Linux	35
Testing Your Installation	35

CHAPTER 3: Interacting with Python	37
Opening the Command Line	38
Starting Python	38
Using the command line to your advantage	39
Using Python environment variables to your advantage	41
Typing a Command	43
Telling the computer what to do	43
Telling the computer you're done	44
Seeing the result	44
Using Help	46
Getting into help mode	46
Asking for help	47
Leaving help mode	49
Obtaining help directly	50
Closing the Command Line	51
CHAPTER 4: Writing Your First Application	55
Understanding Why IDEs Are Important	56
Creating better code	56
Debugging functionality	56
Defining why notebooks are useful	57
Obtaining Your Copy of Anaconda	58
Obtaining Analytics Anaconda	58
Installing Anaconda on Linux	59
Installing Anaconda on MacOS	60
Installing Anaconda on Windows	61
Downloading the Datasets and Example Code	64
Using Jupyter Notebook	64
Defining the code repository	65
Creating the Application	71
Understanding cells	71
Adding documentation cells	74
Other cell content	75
Understanding the Use of Indentation	75
Adding Comments	77
Understanding comments	78
Using comments to leave yourself reminders	79
Using comments to keep code from executing	80
Closing Jupyter Notebook	80

CHAPTER 5: Working with Anaconda	83
Downloading Your Code	84
Working with Checkpoints	85
Defining the uses of checkpoints	85
Saving a checkpoint	86
Restoring a checkpoint	86
Manipulating Cells	86
Adding various cell types	87
Splitting and merging cells	87
Moving cells around	88
Running cells	88
Toggling outputs	90
Changing Jupyter Notebook's Appearance	90
Finding commands using the Command Palette	91
Working with line numbers	92
Using the Cell Toolbar features	93
Interacting with the Kernel	94
Obtaining Help	95
Using the Magic Functions	97
Viewing the Running Processes	99
PART 2: TALKING THE TALK	101
CHAPTER 6: Storing and Modifying Information	103
Storing Information	104
Seeing variables as storage boxes	104
Using the right box to store the data	104
Defining the Essential Python Data Types	105
Putting information into variables	105
Understanding the numeric types	106
Understanding Boolean values	110
Understanding strings	110
Working with Dates and Times	111
CHAPTER 7: Managing Information	113
Controlling How Python Views Data	114
Making comparisons	114
Understanding how computers make comparisons	115
Working with Operators	115
Defining the operators	116
Understanding operator precedence	122

Creating and Using Functions	123
Viewing functions as code packages.	124
Understanding code reusability	124
Defining a function	125
Accessing functions.	126
Sending information to functions.	127
Returning information from functions.	131
Comparing function output.	132
Getting User Input.	132
CHAPTER 8: Making Decisions	135
Making Simple Decisions by Using the if Statement	136
Understanding the if statement	136
Using the if statement in an application	137
Choosing Alternatives by Using the if. .else Statement	141
Understanding the if. .else statement	141
Using the if. .else statement in an application	142
Using the if. .elif statement in an application	143
Using Nested Decision Statements.	146
Using multiple if or if. .else statements	146
Combining other types of decisions	148
CHAPTER 9: Performing Repetitive Tasks	151
Processing Data Using the for Statement	152
Understanding the for statement.	152
Creating a basic for loop.	153
Controlling execution with the break statement	153
Controlling execution with the continue statement.	156
Controlling execution with the pass clause.	157
Controlling execution with the else statement.	158
Processing Data by Using the while Statement	159
Understanding the while statement.	160
Using the while statement in an application	161
Nesting Loop Statements.	162
CHAPTER 10: Dealing with Errors	165
Knowing Why Python Doesn't Understand You	166
Considering the Sources of Errors	167
Classifying when errors occur	168
Distinguishing error types	169
Catching Exceptions	171
Basic exception handling	171
Handling more specific to less specific exceptions	183
Nested exception handling	185

Raising Exceptions	189
Raising exceptions during exceptional conditions	189
Passing error information to the caller	190
Creating and Using Custom Exceptions	191
Using the finally Clause	192
PART 3: PERFORMING COMMON TASKS	195
CHAPTER 11: Interacting with Packages	197
Creating Code Groupings	198
Understanding the package types	200
Considering the package cache	201
Importing Packages	202
Using the import statement	203
Using the from . . import statement	205
Finding Packages on Disk	207
Downloading Packages from Other Sources	209
Opening the Anaconda Prompt	210
Working with conda packages	210
Installing packages by using pip	215
Viewing the Package Content	216
Viewing Package Documentation	219
Opening the Pydoc application	219
Using the quick-access links	220
Typing a search term	221
Viewing the results	222
CHAPTER 12: Working with Strings	225
Understanding That Strings Are Different	226
Defining a character by using numbers	226
Using characters to create strings	227
Creating Strings with Special Characters	229
Selecting Individual Characters	231
Slicing and Dicing Strings	233
Locating a Value in a String	236
Formatting Strings	238
CHAPTER 13: Managing Lists	243
Organizing Information in an Application	244
Defining organization using lists	244
Understanding how computers view lists	245
Creating Lists	246
Accessing Lists	248
Looping through Lists	249

Modifying Lists	250
Searching Lists	254
Sorting Lists	255
Printing Lists	257
Working with the Counter Object	259
CHAPTER 14: Collecting All Sorts of Data	261
Understanding Collections	262
Working with Tuples	263
Working with Dictionaries	266
Creating and using a dictionary	267
Replacing the switch statement with a dictionary	270
Creating Stacks Using Lists	273
Working with queues	275
Working with deques	278
CHAPTER 15: Creating and Using Classes	281
Understanding the Class as a Packaging Method	282
Considering the Parts of a Class	284
Creating the class definition	284
Considering the built-in class attributes	285
Working with methods	286
Working with constructors	288
Working with variables	290
Using methods with variable argument lists	293
Overloading operators	294
Creating a Class	296
Defining the MyClass class	296
Saving a class to disk	297
Using the Class in an Application	298
Extending Classes to Make New Classes	299
Building the child class	299
Testing the class in an application	301
PART 4: PERFORMING ADVANCED TASKS	303
CHAPTER 16: Storing Data in Files	305
Understanding How Permanent Storage Works	306
Creating Content for Permanent Storage	308
Creating a File	311
Reading File Content	314
Updating File Content	317
Deleting a File	321

CHAPTER 17: Sending an Email	323
Understanding What Happens When You Send Email	324
Viewing email as you do a letter	325
Defining the parts of the envelope	326
Defining the parts of the letter	331
Creating the Email Message	335
Working with a text message	335
Working with an HTML message	337
Seeing the Email Output	338
PART 5: THE PART OF TENS	341
CHAPTER 18: Ten Amazing Programming Resources	343
Working with the Python Documentation Online	344
Using the LearnPython.org Tutorial	345
Performing Web Programming by Using Python	346
Getting Additional Libraries	346
Creating Applications Faster by Using an IDE	348
Checking Your Syntax with Greater Ease	348
Using XML to Your Advantage	349
Getting Past the Common Python Newbie Errors	350
Understanding Unicode	351
Making Your Python Application Fast	352
CHAPTER 19: Ten Ways to Make a Living with Python	353
Working in QA	354
Becoming the IT Staff for a Smaller Organization	355
Performing Specialty Scripting for Applications	355
Administering a Network	356
Teaching Programming Skills	357
Helping People Decide on Location	357
Performing Data Mining	358
Interacting with Embedded Systems	358
Carrying Out Scientific Tasks	359
Performing Real-Time Analysis of Data	359
CHAPTER 20: Ten Tools That Enhance Your Python Experience	361
Tracking Bugs with Roundup Issue Tracker	362
Creating a Virtual Environment by Using VirtualEnv	363
Installing Your Application by Using PyInstaller	364
Building Developer Documentation by Using pdoc	365
Developing Application Code by Using Komodo Edit	366

Debugging Your Application by Using pydbgr.	367
Entering an Interactive Environment by Using IPython.	368
Testing Python Applications by Using PyUnit.	368
Tidying Your Code by Using Isort.	369
Providing Version Control by Using Mercurial.	370
CHAPTER 21: Ten (Plus) Libraries You Need to Know About.	371
Developing a Secure Environment by Using PyCrypto.	372
Interacting with Databases by Using SQLAlchemy.	372
Seeing the World by Using Google Maps.	373
Adding a Graphical User Interface by Using TkInter.	373
Providing a Nice Tabular Data Presentation by Using PrettyTable.	374
Enhancing Your Application with Sound by Using PyAudio.	374
Manipulating Images by Using PyQtGraph.	375
Locating Your Information by Using IRLib.	376
Creating an Interoperable Java Environment by Using JPype.	377
Accessing Local Network Resources by Using Twisted Matrix.	378
Accessing Internet Resources by Using Libraries.	378
INDEX.	379

Introduction

Python is an example of a language that does everything right within the domain of things that it's designed to do. This isn't just me saying it, either: Programmers have voted by using Python enough that it's now the fifth-ranked language in the world (see <https://www.tiobe.com/tiobe-index/> for details). The amazing thing about Python is that you really can write an application on one platform and use it on every other platform that you need to support. In contrast to other programming languages that promised to provide platform independence, Python really does make that independence possible. In this case, the promise is as good as the result you get.

Python emphasizes code readability and a concise syntax that lets you write applications using fewer lines of code than other programming languages require. You can also use a coding style that meets your needs, given that Python supports the functional, imperative, object-oriented, and procedural coding styles (see Chapter 3 for details). In addition, because of the way Python works, you find it used in all sorts of fields that are filled with nonprogrammers. *Beginning Programming with Python for Dummies*, 2nd Edition is designed to help everyone, including nonprogrammers, get up and running with Python quickly.

Some people view Python as a scripted language, but it really is so much more. (Chapter 18 gives you just an inkling of the occupations that rely on Python to make things work.) However, Python it does lend itself to educational and other uses for which other programming languages can fall short. In fact, this book uses Jupyter Notebook for examples, which relies on the highly readable literate programming paradigm advanced by Stanford computer scientist Donald Knuth (see Chapter 4 for details). Your examples end up looking like highly readable reports that almost anyone can understand with ease.

About This Book

Beginning Programming with Python For Dummies, 2nd Edition is all about getting up and running with Python quickly. You want to learn the language fast so that you can become productive in using it to perform your real job, which could be anything. Unlike most books on the topic, this one starts you right at the beginning by showing you what makes Python different from other languages and how it can help you perform useful work in a job other than programming. As a result, you

gain an understanding of what you need to do from the start, using hands-on examples and spending a good deal of time performing actually useful tasks. You even get help with installing Python on your particular system.

When you have a good installation on whatever platform you're using, you start with the basics and work your way up. By the time you finish working through the examples in this book, you'll be writing simple programs and performing tasks such as sending an email using Python. No, you won't be an expert, but you will be able to use Python to meet specific needs in the job environment. To make absorbing the concepts even easier, this book uses the following conventions:

- » Text that you're meant to type just as it appears in the book is **bold**. The exception is when you're working through a step list: Because each step is bold, the text to type is not bold.
- » When you see words in *italics* as part of a typing sequence, you need to replace that value with something that works for you. For example, if you see "Type **Your Name** and press Enter," you need to replace *Your Name* with your actual name.
- » Web addresses and programming code appear in monospace. If you're reading a digital version of this book on a device connected to the Internet, note that you can click the web address to visit that website, like this: `www.dummies.com`.
- » When you need to type command sequences, you see them separated by a special arrow, like this: File ⇄ New File. In this case, you go to the File menu first and then select the New File entry on that menu. The result is that you see a new file created.

Foolish Assumptions

You might find it difficult to believe that I've assumed anything about you — after all, I haven't even met you yet! Although most assumptions are indeed foolish, I made these assumptions to provide a starting point for the book.

Familiarity with the platform you want to use is important because the book doesn't provide any guidance in this regard. (Chapter 2 does provide Python installation instructions for various platforms, and Chapter 4 tells you how to install Anaconda, which includes Jupyter Notebook — the Integrated Development Environment, or IDE, used for this book.) To provide you with maximum information about Python, this book doesn't discuss any platform-specific issues. You really do need to know how to install applications, use applications, and generally work with your chosen platform before you begin working with this book.

This book also assumes that you can locate information on the Internet. Sprinkled throughout are numerous references to online material that will enhance your

learning experience. However, these added sources are useful only if you actually find and use them.

Icons Used in This Book

As you read this book, you see icons in the margins that indicate material of interest (or not, as the case may be). This section briefly describes each icon in this book.



TIP

Tips are nice because they help you save time or perform some task without a lot of extra work. The tips in this book are time-saving techniques or pointers to resources that you should try in order to get the maximum benefit from Python.



WARNING

I don't want to sound like an angry parent or some kind of maniac, but you should avoid doing anything marked with a Warning icon. Otherwise, you could find that your program only serves to confuse users, who will then refuse to work with it.



TECHNICAL
STUFF

Whenever you see this icon, think advanced tip or technique. You might find these tidbits of useful information just too boring for words, or they could contain the solution you need to get a program running. Skip these bits of information whenever you like.



REMEMBER

If you don't get anything else out of a particular chapter or section, remember the material marked by this icon. This text usually contains an essential process or a bit of information that you must know to write Python programs successfully.

Beyond the Book

This book isn't the end of your Python programming experience — it's really just the beginning. I provide online content to make this book more flexible and better able to meet your needs. That way, as I receive email from you, I can do things like address questions and tell you how updates to either Python or its associated libraries affect book content. In fact, you gain access to all these cool additions:

» **Cheat sheet:** You remember using crib notes in school to make a better mark on a test, don't you? You do? Well, a cheat sheet is sort of like that. It provides you with some special notes about tasks that you can do with Python that not every other developer knows. You can find the cheat sheet for this book by going to www.dummies.com and searching Beginning Programming For Dummies, 2nd Edition Cheat Sheet. It contains really neat information like the top ten mistakes developers make when working with Python and some of the Python syntax that gives most developers problems.

» **Updates:** Sometimes changes happen. For example, I might not have seen an upcoming change when I looked into my crystal ball during the writing of this book. In the past, that simply meant the book would become outdated and less useful, but you can now find updates to the book at by going to www.dummies.com and searching this book's title.

In addition to these updates, check out the blog posts with answers to reader questions and demonstrations of useful book-related techniques at <http://blog.johnmuelเลอร์books.com/>.

» **Companion files:** Hey! Who really wants to type all the code in the book? Most readers would prefer to spend their time actually working through coding examples, rather than typing. Fortunately for you, the source code is available for download, so all you need to do is read the book to learn Python coding techniques. Each of the book examples even tells you precisely which example project to use. You can find these files at going to www.dummies.com and searching this book's title. On the page that appears, scroll down to the graphic of the book's cover and click it; then click More About This Book. Click the Downloads tab on the page that appears.

Where to Go from Here

It's time to start your Programming with Python adventure! If you're a complete programming novice, you should start with Chapter 1 and progress through the book at a pace that allows you to absorb as much of the material as possible.

If you're a novice who's in an absolute rush to get going with Python as quickly as possible, you could skip to Chapter 2 with the understanding that you may find some topics a bit confusing later. Skipping to Chapter 3 is possible if you already have Python installed, but be sure to at least skim Chapter 2 so that you know what assumptions were made when writing this book.

Readers who have some exposure to Python can save time by moving directly to Chapter 4. It's essential to install Anaconda to gain access to Jupyter Notebook, which is the IDE used for this book. Otherwise, you won't be able to use the downloadable source easily. Anaconda is free, so there is no cost involved.

Assuming that you already have Jupyter Notebook installed and know how to use it, you can move directly to Chapter 6. You can always go back to earlier chapters as necessary when you have questions. However, it's important that you understand how each example works before moving to the next one. Every example has important lessons for you, and you could miss vital content if you start skipping too much information.

1

Getting Started with Python

IN THIS PART . . .

Communicate with your computer.

Install Python on your Linux, Mac, or Windows system.

Interact with the Python-supplied tools.

Install and use Anaconda to write your first application.

Use Anaconda to perform useful work.

- » Talking to your computer
- » Creating programs to talk to your computer
- » Understanding programs and their creation
- » Considering why you want to use Python

Chapter **1**

Talking to Your Computer

Having a conversation with your computer might sound like the script of a science fiction movie. After all, the members of the *Enterprise* on *Star Trek* regularly talked with their computer. In fact, the computer often talked back. However, with the rise of Apple's Siri (<http://www.apple.com/ios/siri/>), Amazon's Echo (<https://www.amazon.com/dp/B00X4WHP5E/>), and other interactive software, perhaps you really don't find a conversation so unbelievable.



REMEMBER

Asking the computer for information is one thing, but providing it with instructions is quite another. This chapter considers why you want to instruct your computer about anything and what benefit you gain from it. You also discover the need for a special language when performing this kind of communication and why you want to use Python to accomplish it. However, the main thing to get out of this chapter is that programming is simply a kind of communication that is akin to other forms of communication you already have with your computer.

Understanding Why You Want to Talk to Your Computer

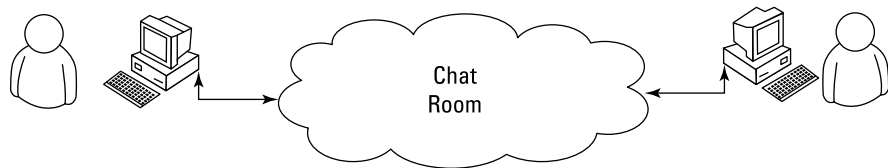
Talking to a machine may seem quite odd at first, but it's necessary because a computer can't read your mind — yet. Even if the computer did read your mind, it would still be communicating with you. Nothing can occur without an exchange of information between the machine and you. Activities such as

- » Reading your email
- » Writing about your vacation
- » Finding the greatest gift in the world

are all examples of communication that occurs between a computer and you. That the computer further communicates with other machines or people to address requests that you make simply extends the basic idea that communication is necessary to produce any result.

In most cases, the communication takes place in a manner that is nearly invisible to you unless you really think about it. For example, when you visit a chat room online, you might think that you're communicating with another person. However, you're communicating with your computer, your computer is communicating with the other person's computer through the chat room (whatever it consists of), and the other person's computer is communicating with that person. Figure 1-1 gives you an idea of what is actually taking place.

FIGURE 1-1: Communication with your computer may be invisible unless you really think about it.



Notice the cloud in the center of Figure 1-1. The cloud could contain anything, but you know that it at least contains other computers running other applications. These computers make it possible for your friend and you to chat. Now, think about how easy the whole process seems when you're using the chat application. Even though all these things are going on in the background, it seems as if you're simply chatting with your friend, and the process itself is invisible.

Knowing that an Application is a Form of Communication

Computer communication occurs through the use of applications. You use one application to answer your email, another to purchase goods, and still another to create a presentation. An *application* (sometimes called an *app*) provides the means to express human ideas to the computer in a manner the computer can understand and defines the tools needed to shape the data used for the communication in specific ways. Data used to express the content of a presentation is different from data used to purchase a present for your mother. The way you view, use, and understand the data is different for each task, so you must use different applications to interact with the data in a manner that both the computer and you can understand.

You can obtain applications to meet just about any general need you can conceive of today. In fact, you probably have access to applications for which you haven't even thought about a purpose yet. Programmers have been busy creating millions of applications of all types for many years now, so it may be hard to understand what you can accomplish by creating some new method for talking with your computer through an application. The answer comes down to thinking about the data and how you want to interact with it. Some data simply isn't common enough to have attracted the attention of a programmer, or you may need the data in a format that no application currently supports, so you don't have any way to tell the computer about it unless you create a custom application to do it.

The following sections describe applications from the perspective of working with unique data in a manner that is special in some way. For example, you might have access to a video library database but no method to access it in a way that makes sense to you. The data is unique and your access needs are special, so you may want to create an application that addresses both the data and your needs.

Thinking about procedures you use daily

A *procedure* is simply a set of steps you follow to perform a task. For example, when making toast, you might use a procedure like this:

1. Get the bread and butter from the refrigerator.
2. Open the bread bag and take out two pieces of toast.
3. Remove the cover from the toaster.
4. Place each piece of bread in its own slot.
5. Push the toaster lever down to start toasting the bread.

6. Wait for the toasting process to complete.
7. Remove toast from the toaster.
8. Place toast on a plate.
9. Butter the toast.

Your procedure might vary from the one presented here, but it's unlikely that you'd butter the toast before placing it in the toaster. Of course, you do actually have to remove the bread from the wrapper before you toast it (placing the bread, wrapper and all, into the toaster would likely produce undesirable results). Most people never actually think about the procedure for making toast. However, you use a procedure like this one even though you don't think about it.



REMEMBER

Computers can't perform tasks without a procedure. You must tell the computer which steps to perform, the order in which to perform them, and any exceptions to the rule that could cause failure. All this information (and more) appears within an application. In short, an application is simply a written procedure that you use to tell the computer what to do, when to do it, and how to do it. Because you've been using procedures all your life, all you really need to do is apply the knowledge you already possess to what a computer needs to know about specific tasks.

Writing procedures down

When I was in grade school, our teacher asked us to write a paper about making toast. After we turned in our papers, she brought in a toaster and some loaves of bread. Each paper was read and demonstrated. None of our procedures worked as expected, but they all produced humorous results. In my case, I forgot to tell the teacher to remove the bread from the wrapper, so she dutifully tried to stuff the piece of bread, wrapper and all, into the toaster. The lesson stuck with me. Writing about procedures can be quite hard because we know precisely what we want to do, but often we leave steps out — we assume that the other person also knows precisely what we want to do.

Many experiences in life revolve around procedures. Think about the checklist used by pilots before a plane takes off. Without a good procedure, the plane could crash. Learning to write a great procedure takes time, but it's doable. You may have to try several times before you get a procedure that works completely, but eventually you can create one. Writing procedures down isn't really sufficient, though — you also need to test the procedure by using someone who isn't familiar with the task involved. When working with computers, the computer is your perfect test subject.

Seeing applications as being like any other procedure

A computer acts like the grade school teacher in my example in the previous section. When you write an application, you're writing a procedure that defines a series of steps that the computer should perform to accomplish whatever task you have in mind. If you leave out a step, the results won't be what you expected. The computer won't know what you mean or that you intended for it to perform certain tasks automatically. The only thing the computer knows is that you have provided it with a specific procedure and it needs to perform that procedure.

Understanding that computers take things literally

People eventually get used to the procedures you create. They automatically compensate for deficiencies in your procedure or make notes about things that you left out. In other words, people compensate for problems with the procedures that you write.



REMEMBER

When you begin writing computer programs, you'll get frustrated because computers perform tasks precisely and read your instructions literally. For example, if you tell the computer that a certain value should equal 5, the computer will look for a value of exactly 5. A human might see 4.9 and know that the value is good enough, but a computer doesn't see things that way. It sees a value of 4.9 and decides that it doesn't equal 5 exactly. In short, computers are inflexible, unintuitive, and unimaginative. When you write a procedure for a computer, the computer will do precisely as you ask absolutely every time and never modify your procedure or decide that you really meant for it to do something else.

Defining What an Application Is

As previously mentioned, applications provide the means to define express human ideas in a manner that a computer can understand. To accomplish this goal, the application relies on one or more procedures that tell the computer how to perform the tasks related to the manipulation of data and its presentation. What you see onscreen is the text from your word processor, but to see that information, the computer requires procedures for retrieving the data from disk, putting it into a form you can understand, and then presenting it to you. The following sections define the specifics of an application in more detail.

Understanding that computers use a special language

Human language is complex and difficult to understand. Even applications such as Siri and Alexa have serious limits in understanding what you're saying. Over the years, computers have gained the capability to input human speech as data and to understand certain spoken words as commands, but computers still don't quite understand human speech to any significant degree. The difficulty of human speech is exemplified in the way lawyers work. When you read legalese, it appears as a gibberish of sorts. However, the goal is to state ideas and concepts in a way that isn't open to interpretation. Lawyers seldom succeed in meeting their objective precisely because human speech is imprecise.

Given what you know from previous sections of this chapter, computers could never rely on human speech to understand the procedures you write. Computers always take things literally, so you'd end up with completely unpredictable results if you were to use human language to write applications. That's why humans use special languages, called *programming languages*, to communicate with computers. These special languages make it possible to write procedures that are both specific and completely understandable by both humans and computers.



Computers don't actually speak any language. They use binary codes to flip switches internally and to perform math calculations. Computers don't even understand letters — they understand only numbers. A special application turns the computer-specific language you use to write a procedure into binary codes. For the purposes of this book, you really don't need to worry too much about the low-level specifics of how computers work at the binary level. However, it's interesting to know that computers speak math and numbers, not really a language at all.

Helping humans speak to the computer

It's important to keep the purpose of an application in mind as you write it. An application is there to help humans speak to the computer in a certain way. Every application works with some type of data that is input, stored, manipulated, and output so that the humans using the application obtain a desired result. Whether the application is a game or a spreadsheet, the basic idea is the same. Computers work with data provided by humans to obtain a desired result.

When you create an application, you're providing a new method for humans to speak to the computer. The new approach you create will make it possible for other humans to view data in new ways. The communication between human and computer should be easy enough that the application actually disappears from view. Think about the kinds of applications you've used in the past. The best applications are the ones that let you focus on whatever data you're interacting with.

For example, a game application is considered immersive only if you can focus on the planet you're trying to save or the ship you're trying to fly, rather than the application that lets you do these things.



TIP

One of the best ways to start thinking about how you want to create an application is to look at the way other people create applications. Writing down what you like and dislike about other applications is a useful way to start discovering how you want your applications to look and work. Here are some questions you can ask yourself as you work with the applications:

- » What do I find distracting about the application?
- » Which features were easy to use?
- » Which features were hard to use?
- » How did the application make it easy to interact with my data?
- » How would I make the data easier to work with?
- » What do I hope to achieve with my application that this application doesn't provide?

Professional developers ask many other questions as part of creating an application, but these are good starter questions because they begin to help you think about applications as a means to help humans speak with computers. If you've ever found yourself frustrated by an application you used, you already know how other people will feel if you don't ask the appropriate questions when you create your application. Communication is the most important element of any application you create.

You can also start to think about the ways in which you work. Start writing procedures for the things you do. It's a good idea to take the process one step at a time and write everything you can think of about that step. When you get finished, ask someone else to try your procedure to see how it actually works. You might be surprised to learn that even with a lot of effort, you can easily forget to include steps.



WARNING

The world's worst application usually begins with a programmer who doesn't know what the application is supposed to do, why it's special, what need it addresses, or whom it is for. When you decide to create an application, make sure that you know why you're creating it and what you hope to achieve. Just having a plan in place really helps make programming fun. You can work on your new application and see your goals accomplished one at a time until you have a completed application to use and show off to your friends (all of whom will think you're really cool for creating it).

Understanding Why Python is So Cool

Many programming languages are available today. In fact, a student can spend an entire semester in college studying computer languages and still not hear about them all. (I did just that during my college days.) You'd think that programmers would be happy with all these programming languages and just choose one to talk to the computer, but they keep inventing more.



REMEMBER

Programmers keep creating new languages for good reason. Each language has something special to offer — something it does exceptionally well. In addition, as computer technology evolves, so do the programming languages in order to keep up. Because creating an application is all about efficient communication, many programmers know multiple programming languages so that they can choose just the right language for a particular task. One language might work better to obtain data from a database, and another might create user interface elements especially well.

As with every other programming language, Python does some things exceptionally well, and you need to know what they are before you begin using it. You might be amazed by the really cool things you can do with Python. Knowing a programming language's strengths and weaknesses helps you use it better as well as avoid frustration by not using the language for things it doesn't do well. The following sections help you make these sorts of decisions about Python.

Unearthing the reasons for using Python

Most programming languages are created with specific goals in mind. These goals help define the language characteristics and determine what you can do with the language. There really isn't any way to create a programming language that does everything because people have competing goals and needs when creating applications. When it comes to Python, the main objective was to create a programming language that would make programmers efficient and productive. With that in mind, here are the reasons that you want to use Python when creating an application:

- » **Less application development time:** Python code is usually 2–10 times shorter than comparable code written in languages like C/C++ and Java, which means that you spend less time writing your application and more time using it.
- » **Ease of reading:** A programming language is like any other language — you need to be able to read it to understand what it does. Python code tends to be easier to read than the code written in other languages, which means you spend less time interpreting it and more time making essential changes.

» **Reduced learning time:** The creators of Python wanted to make a programming language with fewer odd rules that make the language hard to learn. After all, programmers want to create applications, not learn obscure and difficult languages.



TIP

Although Python is a popular language, it's not always the most popular language out there (depending on the site you use for comparison). In fact, it currently ranks fifth on sites such as TIOBE (<http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html>), an organization that tracks usage statistics (among other things). However, if you look at sites such as IEEE Spectrum (<https://spectrum.ieee.org/computing/software/the-2017-top-programming-languages>), you see that Python is actually the number-one language from that site's perspective. Tech Rapidly has it as the number-three language (see <http://techrapidly.com/top-10-best-programming-languages-learn-2018/>).

If you're looking for a language solely for the purpose of obtaining a job, Python is a great choice, but Java, C/C++, or C# might be better choices, depending on the kind of job you want to get. Visual Basic is also a great choice, even if it isn't currently quite as popular as Python. Make sure to choose a language you like and one that will address your application-development needs, but also choose on the basis of what you intend to accomplish. Python was the language of the year in both 2007 and 2010 and has ranked as high as the fourth most popular language in February 2011. So it truly is a good choice if you're looking for a job, but not necessarily the best choice. However, you may be surprised to learn that many colleges now use Python to teach coding, and it has become the most popular language in that venue. Check out my blog post at <http://blog.johnmuellerbooks.com/2014/07/14/python-as-a-learning-tool> for details.

Deciding how you can personally benefit from Python

Ultimately, you can use any programming language to write any sort of application you want. If you use the wrong programming language for the job, the process will be slow, error prone, bug ridden, and you'll absolutely hate it — but you can get the job done. Of course, most of us would rather avoid horribly painful experiences, so you need to know what sorts of applications people typically use Python to create. Here's a list of the most common uses for Python (although people do use it for other purposes):

» **Creating rough application examples:** Developers often need to create a *prototype*, a rough example of an application, before getting the resources to create the actual application. Python emphasizes productivity, so you can use it to create prototypes of an application quickly.

- » **Scripting browser-based applications:** Even though JavaScript is probably the most popular language used for browser-based application scripting, Python is a close second. Python offers functionality that JavaScript doesn't provide (see the comparison at <https://blog.glyphobet.net/essay/2557> for details) and its high efficiency makes it possible to create browser-based applications faster (a real plus in today's fast-paced world).
- » **Designing mathematic, scientific, and engineering applications:** Interestingly enough, Python provides access to some really cool libraries that make it easier to create math, scientific, and engineering applications. The two most popular libraries are NumPy (<http://www.numpy.org/>) and SciPy (<http://www.scipy.org/>). These libraries greatly reduce the time you spend writing specialized code to perform common math, scientific, and engineering tasks.
- » **Working with XML:** The eXtensible Markup Language (XML) is the basis of most data storage needs on the Internet and many desktop applications today. Unlike most languages, where XML is just sort of bolted on, Python makes it a first-class citizen. If you need to work with a Web service, the main method for exchanging information on the Internet (or any other XML-intensive application), Python is a great choice.
- » **Interacting with databases:** Business relies heavily on databases. Python isn't quite a query language, like the Structured Query Language (SQL) or Language INtegrated Query (LINQ), but it does do a great job of interacting with databases. It makes creating connections and manipulating data relatively painless.
- » **Developing user interfaces:** Python isn't like some languages like C# where you have a built-in designer and can drag and drop items from a toolbox onto the user interface. However, it does have an extensive array of graphical user interface (GUI) frameworks — extensions that make graphics a lot easier to create (see <https://wiki.python.org/moin/GuiProgramming> for details). Some of these frameworks do come with designers that make the user interface creation process easier. The point is that Python isn't devoted to just one method of creating a user interface — you can use the method that best suits your needs.

Discovering which organizations use Python

Python really is quite good at the tasks that it was designed to perform. In fact, that's why a lot of large organizations use Python to perform at least some application-creation (development) tasks. You want a programming language that has good support from these large organizations because these organizations tend to spend money to make the language better. Table 1-1 lists the large organizations that use Python the most.

TABLE 1-1**Large Organizations That Use Python**

Vendor	URL	Application Type
Alice Educational Software – Carnegie Mellon University	(https://www.alice.org/)	Educational applications
Fermilab	(https://www.fnal.gov/)	Scientific applications
Go.com	(http://go.com/)	Browser-based applications
Google	(https://www.google.com/)	Search engine
Industrial Light & Magic	(http://www.ilm.com/)	Just about every programming need
Lawrence Livermore National Library	(https://www.llnl.gov/)	Scientific applications
National Space and Aeronautics Administration (NASA)	(http://www.nasa.gov/)	Scientific applications
New York Stock Exchange	(https://nyse.nyx.com/)	Browser-based applications
Redhat	(http://www.redhat.com/)	Linux installation tools
Yahoo!	(https://www.yahoo.com/)	Parts of Yahoo! mail
YouTube	(http://www.youtube.com/)	Graphics engine
Zope – Digital Creations	(http://www.zope.org/en/latest/)	Publishing application

**TIP**

These are just a few of the many organizations that use Python extensively. You can find a more complete list of organizations at <http://www.python.org/about/success/>. The number of success stories has become so large that even this list probably isn't complete and the people supporting it have had to create categories to better organize it.

Finding useful Python applications

You might have an application written in Python sitting on your machine right now and not even know it. Python is used in a vast array of applications on the market today. The applications range from utilities that run at the console to full-fledged CAD/CAM suites. Some applications run on mobile devices, while others run on the large services employed by enterprises. In short, there is no limit to what you can do with Python, but it really does help to see what others have done. You can find a number of places online that list applications written in Python, but the best place to look is <https://wiki.python.org/moin/Applications>.

As a Python programmer, you'll also want to know that Python development tools are available to make your life easier. A *development tool* provides some level of automation in writing the procedures needed to tell the computer what to do. Having more development tools means that you have to perform less work in order to obtain a working application. Developers love to share their lists of favorite tools, but you can find a great list of tools broken into categories at <http://www.python.org/about/apps/>.



REMEMBER

Of course, this chapter describes a number of tools as well, such as NumPy and SciPy (two scientific libraries). The remainder of the book lists a few other tools; make sure that you copy down your favorite tools for later.

Comparing Python to other languages

Comparing one language to another is somewhat dangerous because the selection of a language is just as much a matter of taste and personal preference as it is any sort of quantifiable scientific fact. So before I'm attacked by the rabid protectors of the languages that follow, it's important to realize that I also use a number of languages and find at least some level of overlap among them all. There is no best language in the world, simply the language that works best for a particular application. With this idea in mind, the following sections provide an overview comparison of Python to other languages. (You can find comparisons to other languages at <https://wiki.python.org/moin/LanguageComparisons>.)

C#

A lot of people claim that Microsoft simply copied Java to create C#. That said, C# does have some advantages (and disadvantages) when compared to Java. The main (undisputed) intent behind C# is to create a better kind of C/C++ language — one that is easier to learn and use. However, we're here to talk about C# and Python. When compared to C#, Python has these advantages:

- » Significantly easier to learn
- » Smaller (more concise) code
- » Supported fully as open source
- » Better multiplatform support
- » Easily allows use of multiple development environments
- » Easier to extend using Java and C/C++
- » Enhanced scientific and engineering support

Java

For years, programmers looked for a language that they could use to write an application just once and have it run anywhere. Java is designed to work well on any platform. It relies on some tricks that you'll discover later in the book to accomplish this magic. For now, all you really need to know is that Java was so successful at running well everywhere that other languages have sought to emulate it (with varying levels of success). Even so, Python has some important advantages over Java, as shown in the following list:

- » Significantly easier to learn
- » Smaller (more concise) code
- » Enhanced variables (storage boxes in computer memory) that can hold different kinds of data based on the application's needs while running (dynamic typing)
- » Faster development times

Perl

Perl was originally an acronym for Practical Extraction and Report Language. Today, people simply call it Perl and let it go at that. However, Perl still shows its roots in that it excels at obtaining data from a database and presenting it in report format. Of course, Perl has been extended to do a lot more than that — you can use it to write all sorts of applications. (I've even used it for a Web service application.) In a comparison with Python, you'll find that Python has these advantages over Perl:

- » Simpler to learn
- » Easier to read
- » Enhanced protection for data
- » Better Java integration
- » Fewer platform-specific biases

R

Data scientists often have a tough time choosing between R and Python because both languages are adept at statistical analysis and the sorts of graphing that data scientists need to understand data patterns. Both languages are also open source

and support a large range of platforms. However, R is a bit more specialized than Python and tends to cater to the academic market. Consequently, Python has these advantages over R in that Python:

- » Emphasizes productivity and code readability
- » Is designed for use by enterprises
- » Offers easier debugging
- » Uses consistent coding techniques
- » Has greater flexibility
- » Is easier to learn

IN THIS CHAPTER

- » Obtaining a copy of Python for your system
- » Performing the Python installation
- » Finding and using Python on your system
- » Ensuring your installation works as planned

Chapter 2

Getting Your Own Copy of Python

Creating applications requires that you have another application, unless you really want to get low level and write applications in machine code — a decidedly difficult experience that even true programmers avoid if at all possible. If you want to write an application using the Python programming language, you need the applications required to do so. These applications help you work with Python by creating Python code, providing help information as you need it, and letting you run the code you write. This chapter helps you obtain a copy of the Python application, install it on your hard drive, locate the installed applications so that you can use them, and test your installation so that you can see how it works.

Downloading the Version You Need

Every *platform* (combination of computer hardware and operating system software) has special rules that it follows when running applications. The Python application hides these details from you. You type code that runs on any platform that Python supports, and the Python applications translate that code into something the platform can understand. However, in order for the translation to take

place, you must have a version of Python that works on your particular platform. Python supports these platforms (and possibly others):

- »» Advanced IBM Unix (AIX)
- »» Android
- »» BeOS
- »» Berkeley Software Distribution (BSD)/FreeBSD
- »» Hewlett-Packard Unix (HP-UX)
- »» IBM i (formerly Application System 400 or AS/400, iSeries, and System i)
- »» iPhone Operating System (iOS)
- »» Linux
- »» Mac OS X (comes pre-installed with the OS)
- »» Microsoft Disk Operating System (MS-DOS)
- »» MorphOS
- »» Operating System 2 (OS/2)
- »» Operating System 390 (OS/390) and z/OS
- »» PalmOS
- »» PlayStation
- »» Psion
- »» QNX
- »» RISC OS (originally Acorn)
- »» Series 60
- »» Solaris
- »» Virtual Memory System (VMS)
- »» Windows 32-bit (XP and later)
- »» Windows 64-bit
- »» Windows CE/Pocket PC

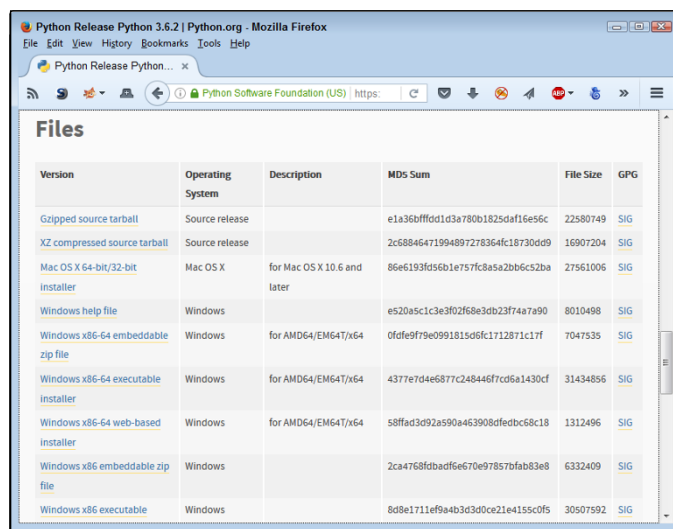


TIP

Wow, that's a lot of different platforms! This book is tested with the Windows, Mac OS X, and Linux platforms. However, the examples could very well work with these other platforms, too, because the examples don't rely on any platform-specific code. Let me know if it works on your non-Windows, Mac, or Linux

platform at John@JohnMuellerBooks.com. The current version of Python at the time of this writing is 3.6.2. I'll talk about any Python updates on my blog at <http://blog.johnmuellerbooks.com>. You can find the answers to your Python book-specific questions there, too.

To get the right version for your platform, you need to go to <https://www.python.org/downloads/release/python-362/>. The download section is initially hidden from view, so you need to scroll halfway down the page. You see a page similar to the one shown in Figure 2-1. The main part of the page contains links for Windows, Mac OS X, and Linux downloads. These links provide you with the default setup that is used in this book. The platform-specific links on the left side of the page show you alternative Python configurations that you can use when the need arises. For example, you may want to use a more advanced editor than the one provided with the default Python package, and these alternative configurations can provide one for you.



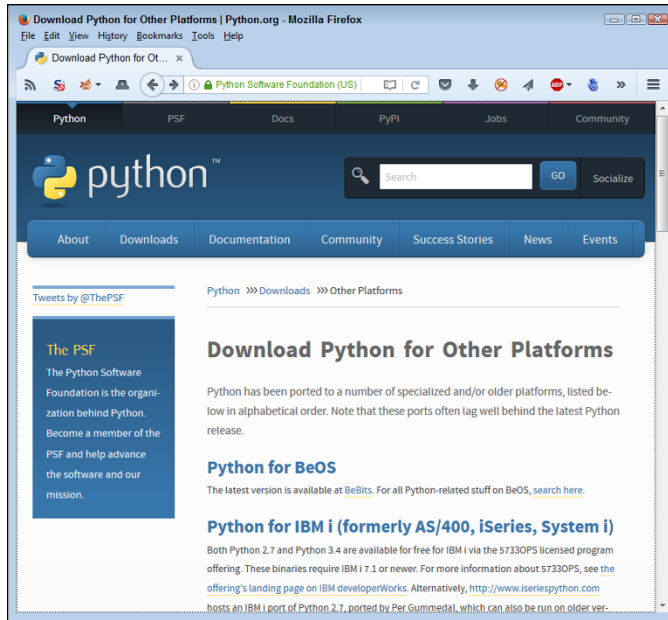
The screenshot shows a web browser window displaying the Python 3.6.2 download page. The page title is "Python Release Python 3.6.2 | Python.org - Mozilla Firefox". The browser address bar shows the URL "https://www.python.org/downloads/release/python-362/". The main content area is titled "Files" and contains a table with the following data:

Version	Operating System	Description	MD5 Sum	File Size	PGP
Gzipped source tarball	Source release		e1a36bffd1d3a780b1825daf16e56c	22580749	SIG
XZ compressed source tarball	Source release		2c68846471994897278364fc18730d09	16907204	SIG
Mac OS X 64-bit/32-bit installer	Mac OS X	for Mac OS X 10.6 and later	86e6193fd56b1e757fc8a5a2bb6c52ba	27561006	SIG
Windows help file	Windows		e520a5c1c3e3f02f68e3db23f74a7a90	8010498	SIG
Windows x86-64 embeddable zip file	Windows	for AMD64/EM64T/x64	0fd9ef979e0991815d66c1712871c17f	7047535	SIG
Windows x86-64 executable installer	Windows	for AMD64/EM64T/x64	4377e7d4e6877c248446f7cd5a1430cf	31434856	SIG
Windows x86-64 web-based installer	Windows	for AMD64/EM64T/x64	58ffad3d92a590a463908dfedbc68c18	1312496	SIG
Windows x86 embeddable zip file	Windows		2ca4768fdbadf6e670e97857bfab83e8	6332409	SIG
Windows x86 executable	Windows		8d9e1711ef9a4b3d3d0ce21e4155c0f5	30507592	SIG

FIGURE 2-1: The Python download page contains links for all sorts of versions.

If you want to work with another platform, go to <https://www.python.org/download/other/> instead. You see a list of Python installations for other platforms, as shown in Figure 2-2. Many of these installations are maintained by volunteers rather than by the people who create the versions of Python for Windows, Mac OS X, and Linux. Make sure you contact these individuals when you have installation questions because they know how best to help you get a good installation on your platform.

FIGURE 2-2:
Volunteers have made Python available on all sorts of platforms.



Installing Python

After you download your copy of Python, it's time to install it on your system. The downloaded file contains everything needed to get you started:

- » Python interpreter
- » Help files (documentation)
- » Command-line access
- » Integrated DeveLopment Environment (IDLE) application
- » Preferred Installer Program (pip)
- » Uninstaller (only on platforms that require it)

This book assumes that you're using one of the default Python setups found at <https://www.python.org/downloads/release/python-362/>. If you use a version other than 3.6.2, some of the examples won't work as anticipated. The following sections describe how to install Python on the three platforms directly supported by this book: Windows, Mac OS X, and Linux.

Working with Windows

The installation process on a Windows system follows the same procedure that you use for other application types. The main difference is in finding the file you downloaded so that you can begin the installation process. The following procedure should work fine on any Windows system, whether you use the 32-bit or the 64-bit version of Python.

1. Locate the downloaded copy of Python on your system.

The name of this file varies, but normally it appears as `python-3.6.2.exe` for both 32-bit systems and `python-3.6.2-amd64.exe` for 64-bit systems. The version number is embedded as part of the filename. In this case, the filename refers to version 3.6.2, which is the version used for this book.

2. Double-click the installation file.

(You may see an Open File – Security Warning dialog box that asks whether you want to run this file. Click Run if you see this dialog box pop up.) You see a Python Setup dialog box similar to the one shown in Figure 2-3. The exact dialog box you see depends on which version of the Python installation program you download.



FIGURE 2-3:
The setup process begins by asking you who should have access to Python.

3. Choose a user installation option (the book uses the default setting of Install for All Users).

Using a personalized installation can make it easier to manage systems that have multiple users. In some cases, the personalized installation also reduces the number of Security Warning dialog boxes you see.



TIP

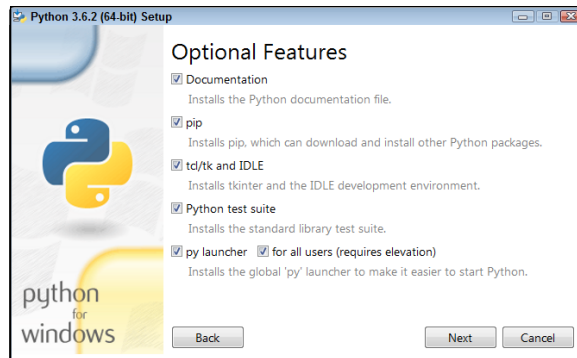
4. Select Add Python 3.6 to PATH.

Adding this setting enables you to access Python from anywhere on your hard drive. If you don't select this setting, you must manually add Python to the path later.

5. Click Customize Installation.

Install asks you to choose which features to use with your copy of Python, as shown in Figure 2-4. Keep all the features selected for this book. However, for your own installation, you may find that you don't actually require all the Python features.

FIGURE 2-4: Choose the Python features you want to install.



6. Click Next.

You see the Advanced Options dialog box, shown in Figure 2-5. Note that Install for All Users isn't selected, despite your having requested that feature earlier. Install also asks you to provide the name of an installation directory for Python. Using the default destination will save you time and effort later. However, you can install Python anywhere you desire.

FIGURE 2-5: Decide on an installation location for your copy of Python.

