Learning Python by Building Games

A beginner's guide to Python programming and game development

Packt>

www.packt.com

Sachin Kafle

Learning Python by Building Games

A beginner's guide to Python programming and game development

Sachin Kafle

Packt>

BIRMINGHAM - MUMBAI

Learning Python by Building Games

Commissioning Editor: Amarabha Banerjee
Acquisition Editor: Kajal Bhagure
Content Development Editor: Aamir Ahmed
Senior Editor: Hayden Edwards
Technical Editor: Jinesh Topiwala
Copy Editor: Safis Editing
Project Coordinator: Manthan Patel
Proofreader: Safis Editing
Indexer: Tejal Daruwale Soni
Production Designer: Alishon Mendonsa

First published: October 2019

Production reference: 1111019

Published by Packt Publishing Ltd.
Livery Place
35 Livery Street
Birmingham
B3 2PB, UK.

ISBN 978-1-78980-298-6

www.packt.com

Dedicated to mom and dad, for all your love and support.
And, to Sonu and Susaan, for the wonderful memories of growing up.

– Sachin Kafle

Packt.com

Subscribe to our online digital library for full access to over 7,000 books and videos, as well as industry leading tools to help you plan your personal development and advance your career. For more information, please visit our website.

Why subscribe?

- Spend less time learning and more time coding with practical eBooks and Videos from over 4,000 industry professionals
- Improve your learning with Skill Plans built especially for you
- Get a free eBook or video every month
- Fully searchable for easy access to vital information
- Copy and paste, print, and bookmark content

Did you know that Packt offers eBook versions of every book published, with PDF and ePub files available? You can upgrade to the eBook version at www.packt.com and as a print book customer, you are entitled to a discount on the eBook copy. Get in touch with us at customercare@packtpub.com for more details.

At www.packt.com, you can also read a collection of free technical articles, sign up for a range of free newsletters, and receive exclusive discounts and offers on Packt books and eBooks.

Foreword

I have known and worked with Sachin Kafle for more than four years. Sachin is one of the most well-known individuals among Nepal-based cyber and Python experts. In this book, *Learning Python by Building Games*, Sachin takes you on a learning journey of core and advanced Python programming paradigms with the help of hands-on examples. For more than 15 years, Python has continued to evolve to meet the needs of developers around the world. For the majority of this time, Sachin has been a key team member in initiating projects by creating and reusing modular programs.

In his presentations and examples, Sachin shows you how easy it is to create a wide range of applications/games using different Python libraries, such as Pygame, Pymunk, and PyOpenGL. Sachin has also helped developers to create a game with a taste for AI.

With *Learning Python by Building Games*, you'll learn the best practices for writing high-quality, reliable, and maintainable code with Python, a general-purpose language. After you have completed Sachin's book, you'll understand how to create and deploy your own mobile/computer games and apps.

Beyond developing apps for desktops and smartphones, you'll learn how to use the Python programming paradigm to accomplish architecture based on AI and simulation.

In *Learning Python by Building Games*, Sachin encapsulates the knowledge gained through years as an academic specialist and Python developer, a Python cybersecurity analyst, and a passionate advocate. Through his words, step-by-step instructions, screenshots, source code snippets, examples, and links to additional sources of information, you will learn how to continuously enhance your skills and apps.

Become a proficient Python developer and build stunning cross-platform apps with Python.

Prof. Dr. Subarna Shakya

Chairman, Computer Engineering Subject committee, Ministry of Education, National Curriculum Development Center (Nepal)

Contributors

About the author

Sachin Kafle is a computer engineer from Tribhuvan University, Nepal, and a programming instructor currently living in Kathmandu. He is the founder of Bitfourstack Technologies, a software company that provides services including automation for real-time problems in businesses. One of his courses, named *Python Game Development*, is the best seller on many e-learning websites. His interests lie in software development and integration practices in the areas of computation and quantitative fields of trade. He has been utilizing his expertise in Python, C, Java, and C# by teaching since 2012. He has been a source of motivation to younger people, and even his peers, regardless of their educational background, who are embarking on their journey in programming.

I would like to acknowledge the amazing staff and editorial team at Packt Publishing: without their talent and dedication, this book would not be such a valuable asset. In particular, I would like to thank Aamir Ahmed and Mohammed Yusuf Imaratwale for having faith in this book from the beginning. Adapting Aamir's many insightful comments and suggestions really uplifted the quality of this book, and I am grateful for all the time and effort he put into this book.
I'd also like to thank the technical reviewer, Jose Angel Munoz, and the technical editor, Jinesh Topiwala, for their thorough attention to the programming aspect of this book. Their detailed labels and understanding of target audiences, along with their invaluable comments, greatly improved the clarity of this book.
Finally, a special thanks to all of my students for their support and zeal for having this book published. Your voracity toward learning game development using Python is what inspired me to write this book.

About the reviewer

Jose Angel Munoz is a system engineer and architect with multiple years of IT infrastructure and infrastructure-as-code development experience. He is an expert in a variety of technologies, has collaborated with different open source projects, including Ansible, Microsoft, Inspec by Chef, Pimoroni, and XLDeploy, and has published different articles in Linux specialised magazines. For Packt, he has reviewed two PowerShell-related books. You can find him on GitHub (`@imjoseangel`).

Packt is searching for authors like you

If you're interested in becoming an author for Packt, please visit `authors.packtpub.com` and apply today. We have worked with thousands of developers and tech professionals, just like you, to help them share their insight with the global tech community. You can make a general application, apply for a specific hot topic that we are recruiting an author for, or submit your own idea.

Table of Contents

Preface

In September of 2018, I was teaching some of my students about game programming and automation using Python. Then, I realized that it was time to create a book that not only offers information on the rich content of game programming using Python but also shows how to make and deploy games that mimic real, world-famous games such as Flappy Bird and Angry Birds. I wanted to equip you with all the essentials and primitives of game programming to become a real-world Python game developer. This book is not your usual and traditional Python theoretical book; our approach will be as practical as possible. Each chapter will contain a single, yet powerful, real-world game example that will not only be interesting but will also edify you with programming paradigms, which will be your first step to becoming a proficient Python developer.

Python is one of the most widely used programming languages of 2018/19, according to a survey conducted by Stack Overflow and TIOBE, and its rate of popularity growth is not expected to decrease any time soon. If you observe what big tech companies use for handling their businesses, you can see that they depend highly upon Python because of its easy usage and rapid prototyping. Not only that, but you can also see that Python can be used to develop a variety of applications ranging from data science to high-end web applications, and as you proceed to learn the basics of Python, you will be ready to create almost anything you want.

There are many reasons to learn Python, and a big one is the Python community. Many of the world's greatest developers contribute incessantly to this Python community by adding new libraries/modules and functionalities. These libraries prove to be extremely helpful if you want to create something new and rapidly. As such, Python is focused on products rather than being bogged down in the routines and complexities of low-level programming, which makes it the most loved programming language of beginners.

In this book, we will start by introducing some important programming concepts, such as variables, numbers, Boolean logic, conditionals, and looping. After building a solid foundation of core programming concepts, we will hop into advanced sections such as data structures and functions. The pace of learning will be increased with the difficulty of the chapters. After finishing `Chapter 7`, *List Comprehension and Properties*, we will be fully equipped with all the basics to be applied while creating advanced things such as flappy bird emulators, angry bird emulators, and AI players. In each chapter, there will be a *game testing and possible modification* topic to compel you to think about how errors should be handled and how programs should be refined.

Requirements for this book

To get a good grasp of each of the topics written about in this book, I encourage you to follow along with the source code and examples. To write code properly, you will need to install Python on your machine. I have used Python's latest version (as of September 2019), version 3.7, but you can use any version newer than 3.5+. The thorough installation process of Python is covered in the first chapter for your machine, based on the OS (Linux, macOS, or Windows) you're using. You will also need an internet connection up and running to download GitHub code and Python third-party libraries. We will be installing different Python libraries, including PyGame, Pymunk, and PyOpenGL later in this book. For each of them, the installation process will be covered in the chapter concerned. While using such modules, our programs will tend to become lengthier, so we strongly encourage you to use a good Python text editor. I will be using the PyCharm IDE to create complex games using Python, and its installation is also covered in the first chapter. Apart from these software requirements, there are no specific requirements for this book.

Who this book is for

This book is for anyone who wants to learn Python. You can be a beginner or someone who has tried learning it previously, but a boring course or book set you off track, or someone who wants to brush up on their skills. This book will help you gain core knowledge and advance your skills in the most interesting way: by building games. It primarily focuses on GUI programming using the Python modules PyGame, PyOpenGL, and Pymunk. No programming skills are expected from learners as we will cover everything you need to know about Python in this book. We will study the `turtle` module by building three mini-games, and you will learn how to create your very own 2D games, even if you are a complete beginner. If you ever wanted to explore game development with Python's PyGame module, this book is for you.

What this book covers

`Chapter 1`, *Getting to Know Python – Setting Up Python and the Editor,* covers the background of game development and the scope of Python in game development. We will set up Python on our local machine and install the appropriate editor. We will also become familiar with the project settings and the interface of the editor. We will see how to install modules in PyCharm. We will execute our first Python program in this chapter.

`Chapter 2`, *Learning the Fundamentals of Python*, takes us through the invigorating stuff of the Python ecosystem, giving us knowledge about the basic concepts of programming such as variables, numbers, and modules. This chapter will give us with knowledge of values, types, and type-casting techniques. We will make a simple tic-tac-toe game using concepts learned in this chapter. This will teach us how to track data in Python programs.

`Chapter 3`, *Flow Control – Building a Decision Maker for Your Game*, covers the concepts of Boolean logic, conditionals, and looping. This chapter will be life-changing for any learning developer. This chapter will provide mainly deal with how things can be automated with logic. We will also see looping patterns and debugging. Some practical examples will be covered in this section. We will refine our tic-tac-toe game by incorporating game logic and flow controls.

`Chapter 4`, *Data Structures and Functions*, covers lists, dictionaries, sets, and tuples. This chapter will help programmers to distinguish between, and choose among, different built-in storage solutions based on different situations. We will learn how to create each of these data structures and how to perform different operations, including adding, deleting, and traversing. We will make use of advanced data structures such as trees and queues in our tic-tac-toe game, which will make our game more rugged.

`Chapter 5`, *Learning About Curses by Building a Snake Game*, covers terminal-independent screen-painting and keyboard-handling facilities for text-based terminals; such terminals include VT100s, the Linux console, and the simulated terminals provided by various programs. We will make a snake game using curses events and screen painting. We will make simple snake game logic using curses properties.

`Chapter 6`, *Object-Oriented Programming*, deals with creating and using objects in your project. We will learn how to wrap data using properties and restrict data access using specifiers. We will also learn how to use the built-in methods of Python to execute overloading. This chapter will mainly deal with the terminologies of **object-oriented programming** (**OOP**), such as classes, encapsulation, inheritance, and polymorphism. We will use the OOP paradigm to make our snake game made with curses more robust and reusable.

`Chapter 7`, *List Comprehension and Properties*, targets making our code simpler and faster in execution. This chapter will teach us how to work with conditions and logic to implement more understandable single-line code. We will see list comprehension and properties in action with our snake game.

`Chapter 8`, *Turtle Class – Drawing on the Screen*, deals with the `turtle` module of Python. This chapter will give a detailed explanation of how to use Python's turtle to draw all over the screen with simple forward/backward commands. We will learn how to make basic objects with turtle and build some skeleton code with Python in this chapter.

`Chapter 9`, *Data Model Implementation*, covers base class implementation. The base class makes use of operator overloading using special built-in Python methods. We will make use of vectors to specify the positions of objects and we will manipulate them with some algebraic operations. Special functions such as `__add__()`, `__mul__()`, `__str__()`, and `__repr__()` will be used to overload operators.

`Chapter 10`, *Upgrading the Snake Game with Turtle*, shows us how to create our first 2D game with a Python script. We will make use of the `turtle` module to create animations on the screen. This will be a simple game, but we will learn how to use the methods of the `turtle` module to move a pen and draw all over our canvas. We will modify our snake game, made following simple OOP concepts, to one that contains simple animations made with turtle. In addition to the snake game, we will also see how to make games such as Pong and Flappy Bird with turtle.

`Chapter 11`, *Outdoing Turtle – Snake Game UI with PyGame,* covers the installation of PyGame on your machine, and we will also cover how to make the basic skeleton code of our game containing display initialization, game loops, states, events, and colors. We will modify our snake game, made with the `turtle` module, by using a sprite and a game controller library named PyGame.

`Chapter 12`, *Learning About Character Animation, Collision, and Movement*, covers game animation, game character movement (such as jumping and walking), random object generation, game loops, collision and hit pipes, scrolling backgrounds, and scoreboards.

`Chapter 13`, *Coding the Tetris Game with PyGame*, deals with basic PyGame graphics, multi-dimensional list processing, increasing game speed and difficulty, the menu for a game, the creation of a game grid, and shapes and valid space determination.

`Chapter 14`, *Getting to Know PyOpenGL*, covers the installation of PyOpenGL on your machine. We will see how to create an OpenGL window. We will make a simple rectangle to begin with, and then look at PyOpenGL and see how the `draw()` method of PyOpenGL works. We will also learn how to draw objects from vertices and edges, adding views for object and clipping parameters.

`Chapter 15`, *Getting to Know Pymunk by Building an Angry Birds Game*, covers Pythonic 2D physics simulation. We will create a space that contains the simulation and sets its gravity, create a body with mass and moment, set the position of the body, create a box shape and attach it to the body, and then add both the body and shape to the simulation. We will create a complete Angry Birds game clone with Pymunk, dealing with sprite sheets and 2D physics.

`Chapter 16`, *Learning Game AI – Building a Bot to Play*, shows how to create game AI. In this game (snake), both the computer and you play as a snake, and the computer snake tries to catch you. The opponent AI tries to determine and go to the destination point based on your location on the board.

To get the most out of this book

To make the most of the information presented in this book, you are encouraged to follow along with the examples. Prior knowledge of Python is not required, but experience of mathematical concepts such as arithmetic and logical operations is essential for understanding the code thoroughly. Python-based applications are not limited to any particular OS, so all that is required is a decent code editor and a browser. Throughout the book, we have used the PyCharm Community 2019.2 editor, which is an open source editor and is free to download.

Download the example code files

You can download the example code files for this book from your account at `www.packt.com`. If you purchased this book elsewhere, you can visit `www.packtpub.com/support` and register to have the files emailed directly to you.

You can download the code files by following these steps:

1. Log in or register at `www.packt.com`.
2. Select the **Support** tab.
3. Click on **Code Downloads**
4. Enter the name of the book in the **Search** box and follow the onscreen instructions.

Once the file is downloaded, please make sure that you unzip or extract the folder using the latest version of:

- WinRAR/7-Zip for Windows
- Zipeg/iZip/UnRarX for Mac
- 7-Zip/PeaZip for Linux

The code bundle for the book is also hosted on GitHub at `https://github.com/PacktPublishing/Learning-Python-by-building-games`. In case there's an update to the code, it will be updated on the existing GitHub repository.

We also have other code bundles from our rich catalog of books and videos available at `https://github.com/PacktPublishing/`. Check them out!

Code in Action

Visit the following link to check out videos of the code being run:

`http://bit.ly/2oE9mHV`

Conventions used

There are a number of text conventions used throughout this book.

`CodeInText`: Indicates code words in text, database table names, folder names, filenames, file extensions, pathnames, dummy URLs, user input, and Twitter handles. Here is an example: "The screenshot shows the edited `python_ex_1.py` file."

A block of code is set as follows:

```
n = int(input("Enter any number"))
for i in range(1,100):
    if i == n:
        print(i)
        break
```

When we wish to draw your attention to a particular part of a code block, the relevant lines or items are set in bold:

```
def fun(b):
    print("message")
    a = 9 + b
    move_player(a)

fun(3)
```

Any command-line input or output is written as follows:

```
>>> cd Desktop
```

Bold: Indicates a new term, an important word, or words that you see on screen. For example, words in menus or dialog boxes appear in the text like this. Here is an example: "In the installer, make sure you check the **Add Python to PATH** box."

Warnings or important notes appear like this.

Tips and tricks appear like this.

Get in touch

Feedback from our readers is always welcome.

General feedback: If you have questions about any aspect of this book, mention the book title in the subject of your message and email us at `customercare@packtpub.com`.

Errata: Although we have taken every care to ensure the accuracy of our content, mistakes do happen. If you have found a mistake in this book, we would be grateful if you would report this to us. Please visit `www.packtpub.com/support/errata`, selecting your book, clicking on the Errata Submission Form link, and entering the details.

Piracy: If you come across any illegal copies of our works in any form on the internet, we would be grateful if you would provide us with the location address or website name. Please contact us at `copyright@packt.com` with a link to the material.

If you are interested in becoming an author: If there is a topic that you have expertise in, and you are interested in either writing or contributing to a book, please visit `authors.packtpub.com`.

Reviews

Please leave a review. Once you have read and used this book, why not leave a review on the site that you purchased it from? Potential readers can then see and use your unbiased opinion to make purchase decisions, we at Packt can understand what you think about our products, and our authors can see your feedback on their book. Thank you!

For more information about Packt, please visit `packt.com`.

1
Getting to Know Python - Setting Up Python and the Editor

Python is notorious in the data and analytics industry, but it is still a hidden artifact in the gaming industry. While making games using other gaming engines such as Unity and Godot, we tend to combine our design logic with core programming principles. But in the case of Python, it is mostly the analysis of problems and programming paradigms that coalesce together. A program flow or structure is a sequence that is dovetailed with its programming paradigms. A programming paradigm, as its name suggests, facilitates the programmer to write a solution to a problem in the most economical and efficient way possible. For instance, writing a program in two lines of code instead of ten lines is an outcome of using a programming paradigm. The purpose of program flow analysis or structural analysis is to uncover information about procedures that need to be invoked for various design patterns.

In this chapter, we will learn about the following topics:

- Introducing programming with Python
- Installing Python
- The building blocks of Python
- Installing the PyCharm IDE
- Programming code without *Hello World*

Technical requirements

The following is a list of the minimum hardware requirements you'll need for this book:

- A working PC with a minimum of 4GB RAM
- An external mouse adapter (if you are using a laptop)
- A minimum of 5GB of hard disk space to download an external IDE and Python packages

You will need the following software to get the most out of this book (we will download all of them in this chapter):

- Various open source Python packages like pygame, pymunk and pyopenGL
- The Pycharm IDE (community version), which you can find at `https://www.jetbrains.com/pycharm/`
- Various open source packages, such as `pygame` and `pycharm`
- The code for this chapter, which can be found in this book's GitHub repository: `https://github.com/PacktPublishing/Learning-Python-by-building-games/tree/master/Chapter01`

Check out the following video to see the code in action:

`http://bit.ly/2o2pVgA`

Introducing programming with Python

The old adage of programming states the following:

> *"Coding is basically the computer language that's used to develop apps, websites, and software. Without it, we'd have none of the major technology we've come to rely on such as Facebook, our smartphones, the browser we choose to view our favorite blogs on, or even the blogs themselves. It all runs on code."*

We couldn't agree more with this. Computer programming can be both a rewarding and tedious activity. Sometimes, we might be in a situation where we can't find the tweaks of the exception (unexpected behavior of the program) that we caught in the program and, later, we find that the error was because of wrong modules or bad practices. Writing programs is similar to *writing essays*; first, we have to learn about the patterns of an essay; then, we analyze the topics and write them; and finally, we check the grammar.

Similar to the process of writing an essay, when writing code, we have to analyze the patterns or grammar of the programming language, then we analyze the problems, and then we write a program. Finally, we check its grammar, which we normally do with alpha and beta testing.

This book will try to turn you into a person who can analyze a problem, build noble logic, and come up with an idea that will solve that problem. We won't make this journey monotonous; instead, we will learn about Python syntax by building games in each chapter. By the end of this book, you will be thinking like a programmer—maybe not a professional one, but at least you will have developed the skill to make your own programs using Python.

There are two crucial things you'll learn about in this book:

- Firstly, you will learn about the vocabulary and grammar of Python. I don't mean learning about Python theory or history. First, we have to learn about Python syntax; then, we will see how we can create statements and expressions with that syntax. This step includes collecting data and information and storing it in an appropriate data structure.
- Then, you will learn about the procedures that come with the idea of calling the appropriate methods. This process includes using the data that was collected in the first step to get the intended output. This second step is not specific to any programming language. This is going to teach us about various programming prototypes rather than just Python.

Learning any other programming languages after learning about Python is a lot easier. The only difference you will observe in other programming language is syntax complexities and program debugging tools. In this book, we will try to learn about as many programming paradigms as possible so that we can start a programming career.

Are you still unsure about Python?

Let's take a look at some of the products that have been made with Python:

- No list starts without mentioning Google. They use it in their web search system and page rank algorithm.
- Disney uses Python for its creative processes.
- BitTorrent and DropBox are written in Python.

- Mozilla Firefox uses it to explore content and is a major contributor to Python packages.
- NASA uses it for scientific purposes.

The list goes on and on!

Let's take a look at how code procedures work in simple terms.

Explaining code procedures

To explain how code procedures work in simple terms, let's take the example of making an omelet. You start by learning the basics from a recipe book. First, you gather some utensils and make sure they are clean and dry. After that, you beat the eggs, salt, and pepper until it's all blended. Then, you add butter to your non-stick pan, add your egg mixture, and cook it or even tilt the pan to check whether every part of the omelet is cooked or not.

In terms of programming, first, we talk about collecting our tools, such as the utensils and eggs, which relates to collecting data that will be manipulated by the instructions we write in our programs. After that, we talk about cooking the eggs, which is your methods. We normally manipulate data in methods to get output in a form that is meaningful to the user. Here, the output is an omelet.

Giving instructions to a program is the job of a programmer. But let's distinguish between a client and a programmer. If you are using a product where you give instructions to the computer to perform tasks for you, then you are a client, but if you design instructions that will complete tasks for a product you've created for everyone, this indicates that you are a programmer. It is only a matter of *for one* or *for everyone* to determine whether a user is a client or programmer.

Some of the instructions we will use in our Windows Command Prompt or Linux Terminal will be for opening the directory of our machine. There are two ways of performing this action. You can either do it using a GUI, or you can use the Terminal or command prompt. If you type in the `dir` command in the respective field, you are now telling the computer to display the directories in that location. The same thing can be done in any programming language. In Python, we have modules to do this for us. We have to import that module before we can use it. Python provides a lot of modules and libraries to perform such operations. In a procedural programming language such as C, which allows low-level interaction with memory, this makes it harder to code, but with Python, it is easier to use the standard library, which makes the code shorter and readable. David Beazley, the author of *How to Think Like a Computer Scientist Learning Python*, was once asked, *why Python?* He simply replied, *Python is simply a lot of fun and more productive*.

Conversing with Python

Python has been around for many years (nearly 29), and regardless of all of the upgrades it has had to go through, it's still standing as the easiest language for beginners to learn. The primary reason for this is that it can be correlated to the English vocabulary. Similar to how we make statements with English words and vocabulary, we can write statements and operations with Python syntax that commands can interpret, execute, and provide us with a result. We can make a sentence such as *go there* as a command to reflect the position of something with conditionals and flow controls. Learning the syntax of Python is pretty easy; the actual task is to use all of the resources provided by Python to build brand new logic to solve intricate problems. Just learning the basic syntax and writing a couple of programs is never enough; you have to practice enough so that you can come up with revolutionary ideas to solve real-world problems.

We have a lot of vocabulary in the English dictionary. Unlike the English dictionary, Python only contains a few words in its container, which we normally call reserved words. There are 33 of them in total. They are instructions that tell the Python interpreter to perform specific operations. Modifying them isn't possible—they can only be used to perform specific tasks. In addition, when we call a print statement and write some text in it, it is expected that it prints out that message. If you want to make a program that takes input from the user, calling the print statement is useless; the input statement has to be called to achieve that. The following table shows our 33 reserved words:

`False`	`class`	`finally`	`is`	`return`
`None`	`continue`	`for`	`lambda`	`try`
`True`	`def`	`from`	`nonlocal`	`while`
`and`	`del`	`global`	`not`	`with`
`as`	`elif`	`if`	`or`	`yield`
`assert`	`else`	`import`	`pass`	
`break`	`except`	`in`	`raise`	

Each of the preceding words can be found in our English dictionary. In addition, if we search for the word `return` in the dictionary, it simply gives us the verb meaning of coming or going back to the original place. The same semantics are used in Python; when you use the return statement with functions, then you are pulling out something from the function. In the upcoming chapters, we will see all of these keywords in action.

Now that we have started to learn how to converse in Python by examining its keywords, we will install Python. Gear yourself up and open your machine for some fun.

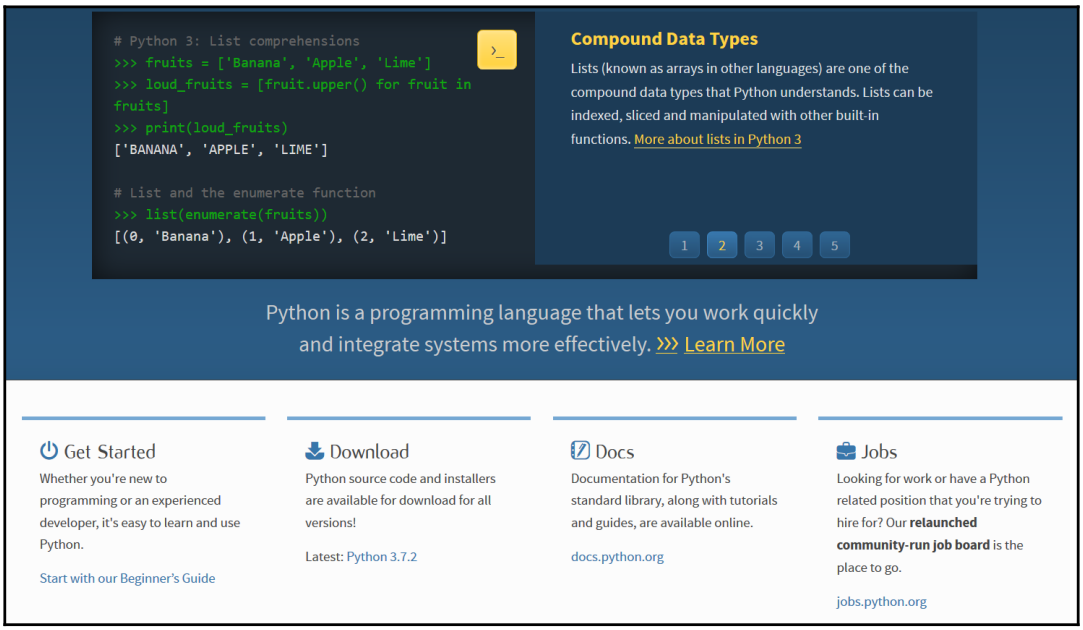
Installing Python

In this section, we will look at installing Python on Windows and macOS.

For the Windows platform

Python doesn't come pre-installed on Windows. We have to download it manually from its official website and then install it. Let's look at how to do this:

1. First of all, open your favorite browser and open the following URL: `https://www.Python.org/`.
2. You will be directed to the page that's shown in the following screenshot. Once you have been redirected to Python's official website, you will see three sections: **Download**, **Docs**, and **Jobs**. Click on the **Download** section at the bottom of the page:



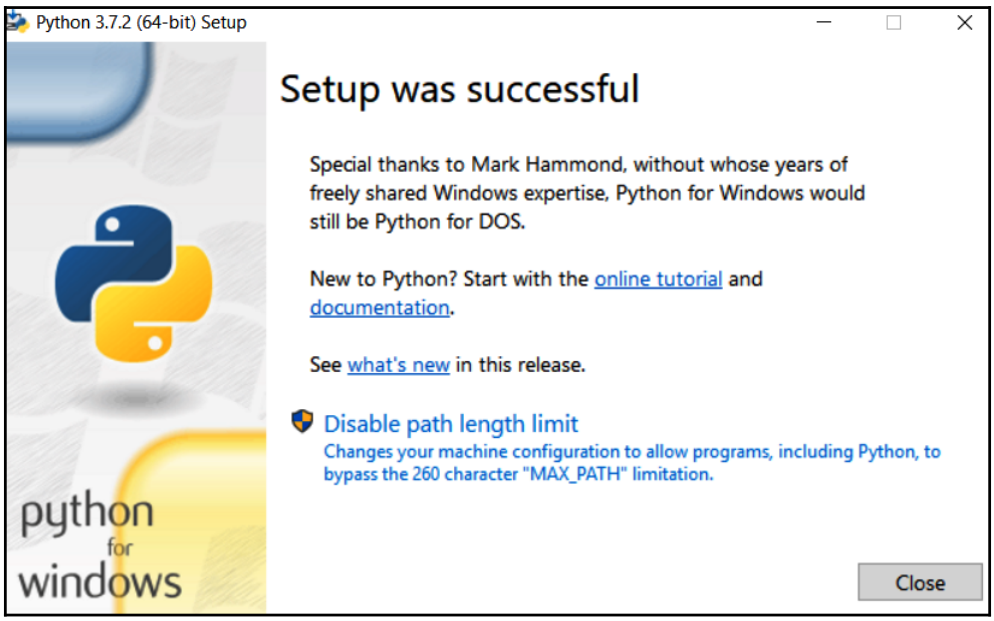
3. You will see a list of files, as shown in the following screenshot. Pick the file that's appropriate for your platform. We're looking at the installation for Windows in this section, so we will click on the Windows executable link. This is highlighted in the following screenshot:

Files

Version	Operating System	Description	MD5 Sum	File Size	GPG
Gzipped source tarball	Source release		02a75015f7cd845e27b85192bb0ca4cb	22897802	SIG
XZ compressed source tarball	Source release		df6ec36011808205beda239c72f947cb	17042320	SIG
macOS 64-bit/32-bit installer	Mac OS X	for Mac OS X 10.6 and later	d8ff07973bc9c009de80c269fd7efcca	34405674	SIG
macOS 64-bit installer	Mac OS X	for OS X 10.9 and later	0fc95e9f6d6b4881f3b499da338a9a80	27766090	SIG
Windows help file	Windows		941b7d6279c0d4060a927a65dcab88c4	8092167	SIG
Windows x86-64 embeddable zip file	Windows	for AMD64/EM64T/x64	f81568590bef56e5997e63b434664d58	7025085	SIG
Windows x86-64 executable installer	Windows	for AMD64/EM64T/x64	ff258093f0b3953c886192dec9f52763	26140976	SIG
Windows x86-64 web-based installer	Windows	for AMD64/EM64T/x64	8de2335249d84fe1eeb61ec25858bd82	1362888	SIG
Windows x86 embeddable zip file	Windows		26881045297dc1883a1d61baffeecaf0	6533256	SIG
Windows x86 executable installer	Windows		38156b62c0cbcb03bfddeb86e66c3a0f	25365744	SIG
Windows x86 web-based installer	Windows		1e6c626514b72e21008f8cd53f945f10	1324648	SIG

4. After clicking on that, you will get a file that needs to be downloaded. After opening that downloaded file, you will get the installer, as follows:

5. In the installer, make sure you check the **Add Python to PATH** box. This will put the Python library files in our environment variables so that we can execute our Python programs. Afterward, you will get a message about its successful installation:



6. Press the Windows key + *R* to open **Run** and type `cmd` in the **Run** tab to open your Windows Command Prompt. Then, type `Python` in the command shell:

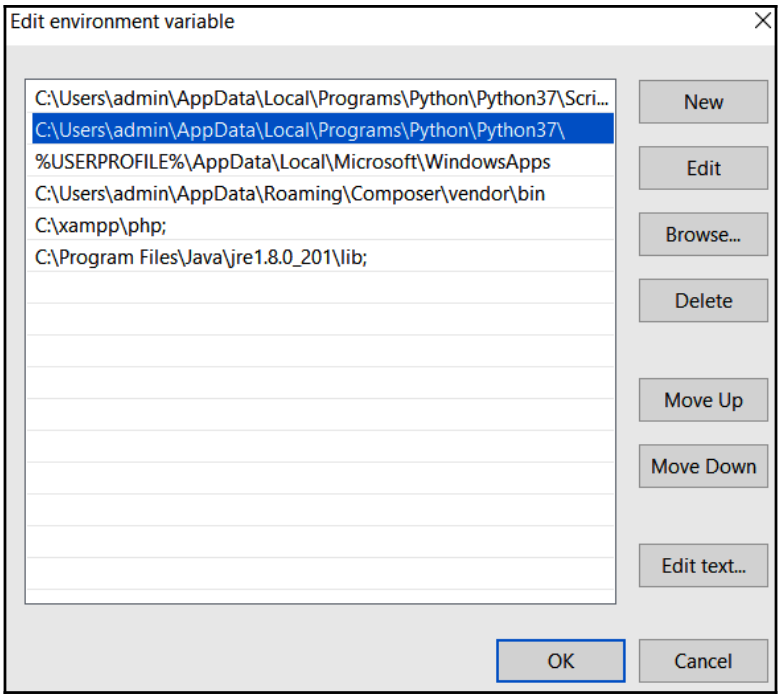
```
Microsoft Windows [Version 10.0.17763.316]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\admin>python
Python 3.7.2 (tags/v3.7.2:9a3ffc0492, Dec 23 2018, 23:09:28) [MSC v.1916 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

If you get the Python version that's displayed in the preceding screenshot, then Python has been successfully installed on your machine. Congratulations! Now, you can get your hands dirty by writing your first program with Python.

If you get an error saying **Python is not recognized as an internal or external command**, you have to explicitly add Python to the path environment variable. Follow these steps to do so:

1. Open the **Control Panel**, navigate to **System and Security**, and then go to **System** to view the basic information about your system.
2. Open your **Advanced system settings** and then **Environment Variables....**
3. In the **Variable** section, search for **Path**. Select the **Path** variable and press the **Edit...** tab.
4. Click **New** in the **Edit Environment Variable** tab.
5. Add this path so that it's pointing to your Python installation directory, that is, **C:\Users\admin\AppData\Local\Programs\Python\Python37**.
6. Click on the **OK** button to save these changes:

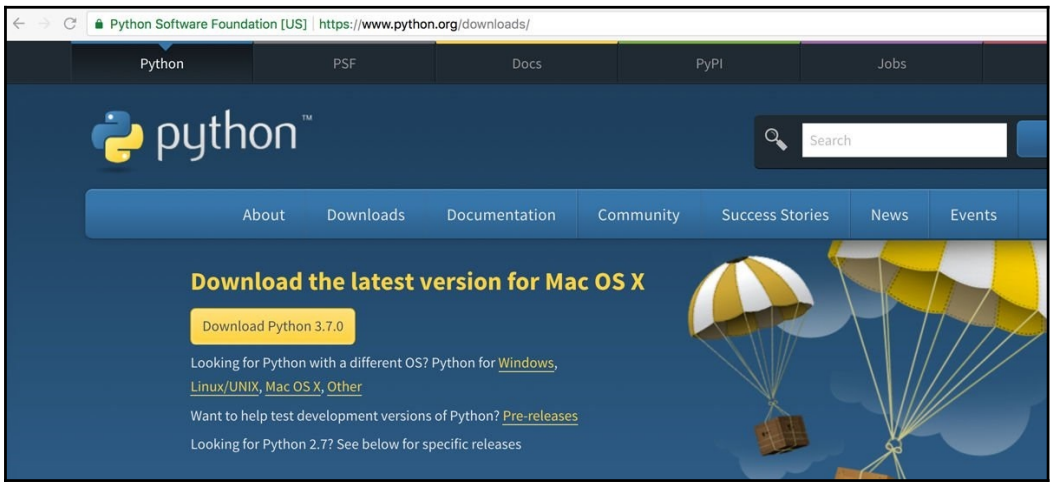


Now, we have successfully installed Python for Windows. If you are using a Mac, the next section will help you to access Python too.

For the Mac platform

Python comes pre-installed with Mac OS X. To check the version of Python you have installed, you should open your command line and type `Python --version`. If you get a version number of 3.5 or newer, you don't need to go through the installation process, but if you have version 2.7, you should follow these instructions to download the latest available version:

1. Open your browser and type in `https://www.Python.org/downloads/`. You will be sent to the following page:



2. Click on the **macOS 64-bit/32-bit installer**. You will be provided with a `.pkg` file. Download it. Then, navigate to that installed directory and click on that installer. You will see the following tab. Press **Continue** to initiate the installer:

Whenever you download Python, a bundle of packages will be installed on your computer. We can't use those packages directly, so we should call them individually for each independent task. To write programs, we need an environment where we can call Python so that it can complete tasks for us. In the next section, we will explore the user-friendly environment provided by Python where we can write our own programs and run them to view their output.

Now that you have installed Python version 3.7 on Mac OS X, you can open your Terminal and check the version of Python you have with the `python --version` command. You will see Python 2.7.10. The reason for this is that Mac OS X comes preinstalled with version 2.7+ of Python. To use the newer version of Python, you have to use the `python3` command. Type the following command into your Terminal and observe the result:

```
python3 --version
```

Now, to make sure Python uses the interpreter with the newer version that you just installed, you can use an aliasing technique that will replace the current working Python version with Python3. To perform aliasing, you have to follow these steps:

1. Open your Terminal and type in the `nano ~/.bash_profile` command to open a bash file using the nano editor.
2. Next, go to the end of the file (after import PATH) and type in the `alias python=python3` command. To save a nano file, press *Ctrl* + *X* and then *Y* to save.

Now, open your Terminal again and type in the same command that we used previously to check the Python version we have. It will be updated to the newer version of Python. From now on, in order to run any Python file from your Mac, you can use this Python command, followed by the signature of the file or filename.

Introducing the Python Shell and IDLE

The Python Shell is similar to Command Prompt for Windows and the Terminal for Linux and Mac OS X where you write commands that will be executed in the filesystem. The results of these commands are printed instantly within the shell. You can also get direct access to this shell using a Python command (> python) in any Terminal. The result will contain an exception and an error due to the improper execution of the code, as follows:

```
>>> imput("Enter something")
Traceback (most recent call last):
  File "<pyshell#5>", line 1, in <module>
    imput()
NameError: name 'imput' is not defined


>>> I love Python
SyntaxError: invalid syntax
```

As you can see, we ran into an error and the Python IDE is explicitly telling us the name of error we ran into, which in this case is `NameError` (a type of syntax error). `SyntaxError` occurs due to an incorrect pattern of code. In the preceding code example, when you write the `I love Python` syntax, this implies nothing to the Python interpreter. You should write proper commands or define something properly if you want to rectify that problem. Writing `imput` instead of input is also a syntax error.

Logic errors or semantic errors occur even if your program syntax is correct. However, this doesn't solve your problem domain. They are dangerous as they are hard to track. The program is perfectly correct but does not solve any problem that it's intended to.

When you download the Python package on your machine, a **Python Integrated Development Environment** (**IDE**) called IDLE (Python's built-in IDE) is downloaded automatically onto your machine. You can type `IDLE` into the search bar to navigate to this environment. IDLE is a free open source program that provides two interfaces where you can write code. We can write scripts and Terminal commands in IDLE.

Now that we are familiar with what not to do in the Python Shell, let's talk about the particulars of the Python Shell—an environment where you can write your Python code.