



MICHAEL DRISCOLL

PYTHON 101

SECOND EDITION



Python 101

2nd Edition

Michael Driscoll

This book is for sale at <http://leanpub.com/py101>

This version was published on 2021-01-15



This is a [Leanpub](#) book. Leanpub empowers authors and publishers with the Lean Publishing process. [Lean Publishing](#) is the act of publishing an in-progress ebook using lightweight tools and many iterations to get reader feedback, pivot until you have the right book and build traction once you do.

© 2020 - 2021 Michael Driscoll

Tweet This Book!

Please help Michael Driscoll by spreading the word about this book on [Twitter](#)!

The suggested hashtag for this book is [#Python101](#).

Find out what other people are saying about the book by clicking on this link to search for this hashtag on Twitter:

[#Python101](#)

Also By Michael Driscoll

[Python 201](#)

[ReportLab - PDF Processing with Python](#)

[Jupyter Notebook 101](#)

[Creating GUI Applications with wxPython](#)

Contents

About the Technical Reviewers	1
Ethan Furman	1
Martin Breuss	1
Acknowledgments	2
Introduction	3
Part I - The Basics	3
Part II - Intermediate Materials	3
Part III - Tutorials	4
Part IV - Python Packaging and Distribution	4
Target Audience	5
About the Author	5
Conventions	5
Requirements	6
Book Source Code	6
Reader Feedback	6
Errata	6
Part I - The Python Language	7
Chapter 1 - Installing Python	8
Installing on Windows	9
Installing on Mac	15
Installing on Linux	21
Android / iOS	22
Other Operating Systems	22
Other Python Variants	22
Wrapping Up	22
Chapter 2 - Python Editors	24
What About the REPL?	25
Getting Started with IDLE	28
Getting Started with PyCharm Community Edition	36
Getting Started with Wing Personal	40

CONTENTS

Getting Started with Visual Studio Code	42
Wrapping Up	45
Chapter 3 - Documenting Your Code	46
What are Comments?	46
Commenting Out	47
Multiline Comments	48
Learning About docstrings	48
Python's Style Guide: PEP8	49
Tools that can help	49
Wrapping Up	50
Review Questions	50
Chapter 4 - Working with Strings	51
Creating Strings	51
String Methods	52
String Formatting	54
Formatting Strings Using %s (printf-style)	55
Formatting Strings Using .format()	56
Formatting Strings with f-strings	59
String Concatenation	60
String Slicing	61
Wrapping Up	62
Review Questions	62
Chapter 5 - Numeric Types	64
Integers	64
Floats	65
Complex Numbers	66
Numeric Operations	66
Augmented Assignment	67
Wrapping Up	68
Review Questions	68
Chapter 6 - Learning About Lists	69
Creating Lists	69
List Methods	70
List Slicing	77
Copying a List	78
Wrapping Up	79
Review Questions	79
Chapter 7 - Learning About Tuples	81
Creating Tuples	81

CONTENTS

Working With Tuples	82
Concatenating Tuples	83
Special Case Tuples	83
Wrapping Up	84
Review Questions	84
Chapter 8 - Learning About Dictionaries	86
Creating Dictionaries	86
Accessing Dictionaries	87
Dictionary Methods	88
Modifying Your Dictionary	92
Deleting Items From Your Dictionary	92
Wrapping Up	93
Review Questions	93
Chapter 9 - Learning About Sets	95
Creating a Set	95
Accessing Set Members	96
Changing Items	97
Adding Items	97
Removing Items	98
Clearing or Deleting a Set	99
Set Operations	100
Wrapping Up	101
Review Questions	102
Chapter 10 - Boolean Operations and None	103
The bool() Function	103
What About None?	103
Wrapping Up	103
Review Questions	103
Chapter 11 - Conditional Statements	104
Comparison Operators	104
Creating a Simple Conditional	104
Branching Conditional Statements	104
Nesting Conditionals	104
Logical Operators	104
Special Operators	104
Wrapping Up	105
Review Questions	105
Chapter 12 - Learning About Loops	106
Creating a for Loop	106

CONTENTS

Looping Over a String	106
Looping Over a Dictionary	106
Extracting Multiple Values in a Tuple While Looping	106
Using enumerate with Loops	106
Creating a while Loop	106
Breaking Out of a Loop	107
Using continue	107
Loops and the else Statement	107
Nesting Loops	107
Wrapping Up	107
Review Questions	107
Chapter 13 - Python Comprehensions	108
List Comprehensions	108
Nested List Comprehensions	108
Dictionary Comprehensions	108
Set Comprehensions	108
Wrapping Up	108
Review Questions	109
Chapter 14 - Exception Handling	110
The Most Common Exceptions	110
Handling Exceptions	110
Raising Exceptions	110
Examining the Exception Object	110
Using the finally Statement	110
Using the else Statement	110
Wrapping Up	111
Review Questions	111
Chapter 15 - Working with Files	112
The open() Function	112
Reading Files	112
Reading Binary Files	112
Writing Files	112
Seeking Within a File	112
Appending to Files	112
Catching File Exceptions	113
Wrapping Up	113
Review Questions	113
Chapter 16 - Importing	114
Using import	114
Using from to Import Specific Bits & Pieces	114

CONTENTS

Using as to assign a new name	114
Importing Everything	114
Wrapping Up	114
Review Questions	114
Chapter 17 - Functions	115
Creating a Function	115
Calling a Function	115
Passing Arguments	115
Type Hinting Your Arguments	115
Passing Keyword Arguments	115
Required and Default Arguments	115
What are *args and **kwargs?	116
Positional-only Parameters	116
Scope	116
Wrapping Up	116
Review Questions	116
Chapter 18 - Classes	117
Class Creation	117
Figuring Out self	117
Public and Private Methods / Attributes	117
Subclass Creation	117
Polymorphism	117
Making the Class Nicer	117
Wrapping Up	118
Review Questions	118
Part II - Beyond the Basics	119
Chapter 19 - Introspection	120
Using the type() Function	120
Using the dir() Function	120
Getting help()	120
Other Built-in Introspection Tools	120
Wrapping Up	121
Review Questions	121
Chapter 20 - Installing Packages with pip	122
Installing a Package	122
Exploring Command Line Options	122
Installing with requirements.txt	122
Upgrading a Package	122
Checking What's Installed	122

CONTENTS

Uninstalling Packages	123
Alternatives to pip	123
Wrapping Up	123
Review Questions	123
Chapter 21 - Python Virtual Environments	124
Python's venv Library	124
The virtualenv Package	124
Other Tools	124
Wrapping Up	124
Review Questions	124
Chapter 22 - Type Checking in Python	125
Pros and Cons of Type Hinting	125
Built-in Type Hinting / Variable Annotation	125
Collection Type Hinting	125
Hinting Values That Could be None	125
Type Hinting Functions	125
What To Do When Things Get Complicated	125
Classes	126
Decorators	126
Aliasing	126
Other Type Hints	126
Type Comments	126
Static Type Checking	126
Wrapping Up	126
Review Questions	126
Chapter 23 - Creating Multiple Threads	127
Pros of Using Threads	127
Cons of Using Threads	127
Creating Threads	127
Subclassing Thread	127
Writing Multiple Files with Threads	127
Wrapping Up	128
Review Questions	128
Chapter 24 - Creating Multiple Processes	129
Pros of Using Processes	129
Cons of Using Processes	129
Creating Processes with multiprocessing	129
Subclassing Process	129
Creating a Process Pool	129
Wrapping Up	130

Review Questions	130
Chapter 25 - Launching Subprocesses with Python	131
The subprocess.run() Function	131
The subprocess.Popen() Class	131
The subprocess.Popen.communicate() Function	131
Reading and Writing with stdin and stdout	131
Wrapping Up	131
Review Questions	132
Chapter 26 - Debugging Your Code with pdb	133
Starting pdb in the REPL	133
Starting pdb on the Command Line	133
Stepping Through Code	133
Adding Breakpoints in pdb	133
Creating a Breakpoint with set_trace()	133
Using the built-in breakpoint() Function	134
Getting Help	134
Wrapping Up	134
Review Questions	134
Chapter 27 - Learning About Decorators	135
Creating a Function	135
Creating a Decorator	135
Applying a Decorator with @	135
Creating a Decorator for Logging	135
Stacking Decorators	135
Passing Arguments to Decorators	136
Using a Class as a Decorator	136
Python's Built-in Decorators	136
Python Properties	136
Wrapping Up	136
Review Questions	136
Chapter 28 - Assignment Expressions	137
Using Assignment Expressions	137
What You Cannot Do With Assignment Expressions	137
Wrapping Up	137
Review Questions	137
Chapter 29 - Profiling Your Code	138
Learning How to Profile with cProfile	138
Profiling a Python Script with cProfile	138
Working with Profile Data Using pstats	138

CONTENTS

Other Profilers	138
Wrapping Up	139
Review Questions	139
Chapter 30 - An Introduction to Testing	140
Using doctest in the Terminal	140
Using doctest in Your Code	140
Using doctest From a Separate File	140
Using unittest For Test Driven Development	140
Wrapping Up	141
Review Questions	141
Chapter 31 - Learning About the Jupyter Notebook	142
Installing The Jupyter Notebook	142
Creating a Notebook	142
Adding Content	142
Adding an Extension	143
Exporting Notebooks to Other Formats	143
Wrapping Up	144
Review Questions	144
Part III - Practical Python	145
Chapter 32 - How to Create a Command-line Application with argparse	146
Parsing Arguments	146
Creating Helpful Messages	146
Adding Aliases	146
Using Mutually Exclusive Arguments	146
Creating a Simple Search Utility	146
Wrapping Up	147
Review Questions	147
Chapter 33 - How to Parse XML	148
Parsing XML with ElementTree	148
Creating XML with ElementTree	148
Editing XML with ElementTree	148
Manipulating XML with lxml	148
Wrapping Up	148
Review Questions	148
Chapter 34 - How to Parse JSON	149
Encoding a JSON String	149
Saving JSON to Disk	149
Decoding a JSON String	149

CONTENTS

Loading JSON from Disk	149
Validating JSON with <code>json.tool</code>	149
Wrapping Up	149
Review Questions	150
Chapter 35 - How to Scrape a Website	151
Rules for Web Scraping	151
Preparing to Scrape a Website	151
Scraping a Website	151
Downloading a File	151
Wrapping Up	151
Review Questions	151
Chapter 36 - How to Work with CSV files	152
Reading a CSV File	152
Reading a CSV File with <code>DictReader</code>	152
Writing a CSV File	152
Writing a CSV File with <code>DictWriter</code>	152
Wrapping Up	152
Review Questions	152
Chapter 37 - How to Work with a Database Using <code>sqlite3</code>	153
Creating a SQLite Database	153
Adding Data to Your Database	153
Searching Your Database	153
Editing Data in Your Database	153
Deleting Data From Your Database	153
Wrapping Up	154
Review Questions	154
Chapter 38 - Working with an Excel Document in Python	155
Python Excel Packages	155
Getting Sheets from a Workbook	155
Reading Cell Data	155
Iterating Over Rows and Columns	155
Writing Excel Spreadsheets	155
Adding and Removing Sheets	156
Adding and Deleting Rows and Columns	156
Wrapping Up	156
Review Questions	156
Chapter 39 - How to Generate a PDF	157
Installing ReportLab	157
Creating a Simple PDF with the Canvas	157

CONTENTS

Creating Drawings and Adding Images Using the Canvas	157
Creating Multi-page Documents with PLATYPUS	157
Creating a Table	157
Wrapping Up	158
Review Questions	158
Chapter 40 - How to Create Graphs	159
Installing Matplotlib	159
Creating a Simple Line Chart with PyPlot	159
Creating a Bar Chart	159
Creating a Pie Chart	159
Adding Labels	159
Adding Titles to Plots	159
Creating a Legend	160
Showing Multiple Figures	160
Wrapping Up	160
Review Questions	160
Chapter 41 - How to Work with Images in Python	161
Installing Pillow	161
Opening Images	161
Cropping Images	161
Using Filters	161
Adding Borders	161
Resizing Images	162
Wrapping Up	162
Review Questions	162
Chapter 42 - How to Create a Graphical User Interface	163
Installing wxPython	163
Learning About Event Loops	163
How to Create Widgets	163
How to Lay Out Your Application	163
How to Add Events	163
How to Create an Application	164
Wrapping Up	164
Review Questions	164
Part IV - Distributing Your Code	165
Chapter 43 - How to Create a Python Package	166
Creating a Module	166
Creating a Package	166
Packaging a Project for PyPI	166

CONTENTS

Creating Project Files	166
Creating setup.py	166
Generating a Python Wheel	167
Uploading to PyPI	167
Wrapping Up	167
Review Questions	167
Chapter 44 - How to Create an Exe for Windows	168
Installing PyInstaller	168
Creating an Executable for a Command-Line Application	168
Creating an Executable for a GUI	168
Wrapping Up	168
Review Questions	168
Chapter 45 - How to Create an Installer for Windows	169
Installing Inno Setup	169
Creating an Installer	169
Testing Your Installer	169
Wrapping Up	169
Review Questions	169
Chapter 46 - How to Create an “exe” for Mac	170
Installing PyInstaller	170
Creating an Executable with PyInstaller	170
Wrapping Up	170
Review Questions	170
Afterword	171
Appendix A - Version Control	172
Version Control Systems	172
Distributed vs Centralized Versioning	172
Common Terminology	172
Python IDE Version Control Support	174
Wrapping Up	175
Appendix B - Version Control with Git	176
Installing Git	176
Configuring Git	176
Creating a Project	176
Ignoring Files	177
Initializing a Repository	177
Checking the Project Status	177
Adding Files to a Repository	177

CONTENTS

Committing Files	177
Viewing the Log	177
Changing a File	177
Reverting a File	178
Checking Out Previous Commits	178
Pushing to Github	178
Wrapping Up	178
Review Question Answer Key	179
Chapter 3 - Documenting Your Code	179
Chapter 4 - Working with Strings	179
Chapter 5 - Numeric Types	180
Chapter 6 - Learning About Lists	181
Chapter 7 - Learning About Tuples	181
Chapter 8 - Learning About Dictionaries	182
Chapter 9 - Learning About Sets	183
Chapter 10 - Boolean Operations and None	184
Chapter 11 - Conditional Statements	184
Chapter 12 - Learning About Loops	185
Chapter 13 - Python Comprehensions	186
Chapter 14 - Exception Handling	187
Chapter 15 - Working with Files	187
Chapter 16 - Importing	188
Chapter 17 - Functions	189
Chapter 18 - Classes	190
Chapter 19 - Introspection	190
Chapter 20 - Installing Packages with pip	191
Chapter 21 - Python Virtual Environments	191
Chapter 22 - Type Checking in Python	192
Chapter 23 - Creating Multiple Threads	192
Chapter 24 - Creating Multiple Processes	193
Chapter 25 - Launching Subprocesses with Python	194
Chapter 26 - Debugging Your Code	194
Chapter 27 - Learning About Decorators	195
Chapter 28 - Assignment Expressions	195
Chapter 29 - Profiling Your Code	196
Chapter 30 - An Introduction to Testing	196
Chapter 31 - Learning About the Jupyter Notebook	197
Chapter 32 - How to Create a Command Line Application with argparse	198
Chapter 33 - How to Parse XML	198
Chapter 34 - How to Parse JSON	199
Chapter 35 - How to Scrape a Website	199
Chapter 36 - How to Work with CSV files	200

CONTENTS

Chapter 37 - How to Work with a Database Using <code>sqlite</code>	200
Chapter 38 - Working with an Excel Document in Python	201
Chapter 39 - How to Generate a PDF	202
Chapter 40 - How to Create Graphs	203
Chapter 41 - How to Work with Images in Python	203
Chapter 42 - How to Create a Graphical User Interface	204
Chapter 43 - How to Create a Python Package	204
Chapter 44 - How to Create an Exe for Windows	205
Chapter 45 - How to Create an Installer for Windows	206
Chapter 46 - How to Create an “exe” for Mac	206

About the Technical Reviewers

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Ethan Furman

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Martin Breuss

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Acknowledgments

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Introduction

Welcome to the 2nd Edition of **Python 101**! The original Python 101 came out in the summer of 2014 and was written with Python 3.5 in mind. The 2nd Edition of this book has been completely updated and rearranged for the latest version of Python, which at the time of writing is 3.8.

Some publishers / authors will only do minor updates when creating a new edition of a book. That is not how I roll. I have personally gone through the entire book and updated every single chapter. I have removed content that was either no longer relevant or could lead to confusion to readers. I have also added several new chapters to the book that cover such things as using version control and setting up projects.

Many programming books will only teach you the basics of the language. With **Python 101**, the goal is to help you not only learn the basics of the language but to go beyond the basics and dig into some intermediate level material. The reason for this is that you usually need to know more than the basics to create something valuable.

Because of this, the book will be split up into the following four parts:

- Part one will cover Python's basics
- Part two will be intermediate material
- Part three will be a series of small tutorials
- Part four will cover Python packaging and distribution

Note that not all sections will be the same length.

Let's go ahead and talk about each of these sections in turn!

Part I - The Basics

This is the heart of the book. In this section you will learn all the basics that you need to know to start using Python effectively. Each chapter will teach you something new and they are ordered in such a way that they will build on each other. If you already know Python well, then you can skip this section and move on to **Part II**.

Part II - Intermediate Materials

Now that you know how Python works, you can dive into more intermediate level material. In this section, you will learn about the following topics:

- Virtual environments
- Type hinting
- Threads and Processes
- Debugging
- Decorators
- Code profiling
- Basic testing

These topics cover some intermediate level Python and also help you learn some key software development skills, like knowing how to debug your code, add basic unit tests, and use version control.

Part III - Tutorials

This part of the book is where you will put it all together. You will learn how to use Python with some real world scripts. These scripts will be basic, but they will demonstrate the power of Python and what you can do with it.

Here is what will be covered:

- How to Create a Command Line Application
- How to Parse XML
- How to Parse JSON
- How to Scrape a Website
- How to Work with CSV Files
- How to Work with a SQLite Database
- How to Create an Excel Document
- How to Generate a PDF

Part IV - Python Packaging and Distribution

Now that you know how to write programs, you will probably want to know how to share them with your friends. In this section, you will learn how to transform your code into something that other developers or users can use.

Specifically you will learn how to:

- Create a Cross Platform Python Package
- Create an Exe for Windows
- Create an Installer for Windows
- Create an “exe” for Mac

By the end of this section, you should be able to confidently distribute your code all on your own!

Target Audience

This book is written for people that have used other programming languages or taken some computer science or related classes. While this book won't handhold you through all the terminology, it will help you learn how to use Python effectively. It also covers some intermediate level topics that most beginner books do not.

About the Author

Michael Driscoll has been programming with Python for more than a decade. He is active in multiple Python communities and is a contributor for Real Python. Mike has also been blogging about Python at <http://www.blog.pythonlibrary.org/> for many years and has written several books about Python:

- Python 101 (1st Edition)
- Python 201: Intermediate Python
- wxPython Recipes
- Python Interviews
- ReportLab: PDF Publishing with Python
- Jupyter Notebook 101
- Creating GUI Applications with wxPython

Conventions

All technical books have their own conventions for how things are presented. In this book, new topics will be in **bold**. When referring to Python related keywords or code in a sentence, they will be in monospace.

Code blocks will look like this:

```
1 def greeter(name: str) -> None:
2     print(f'Hello {name}')
3
4 greeter('Mike')
```

There will also be blocks of code that represent Python's interactive interpreter, also known as a REPL:

```
1 >>> name = 'Mike'
2 >>> print(f'My name is {name}')
3 My name is Mike
```

This demonstrates how the interpreter should behave.

Requirements

You will need the Python language to follow along in this book. See chapter 1 for installation details or go get the official Python distribution for free at:

- <http://python.org/download/>

If you need anything beyond what comes with Python, the chapter will tell you how to install it.

Book Source Code

The book's source code can be found on Github:

- <https://github.com/driscollis/python101code>

Reader Feedback

If you enjoyed the book or have any other kind of feedback, I would love to hear from you. You can contact me at the following:

- comments@pythonlibrary.org

Errata

I try my best not to publish errors in my writings, but it happens from time to time. If you happen to see an error in this book, feel free to let me know by emailing me at the following:

* errata@pythonlibrary.org

Now let's get started!

Part I - The Python Language

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Chapter 1 - Installing Python

Depending on which operating system you are using, you may need to install the Python programming language. This chapter will cover the primary ways to install Python.

First of all, there are several different versions of Python, which are called “distributions”. A distribution is a word used to describe a collection of software. A Python distribution will include the core Python language at a minimum and sometimes include extra 3rd party libraries.

The official version is called Python or CPython and you can get it from the following:

- <https://www.python.org/>

Another popular distribution of Python is called **Anaconda** and comes from the Anaconda company. This variation of Python is focused on data science and includes many additional 3rd party packages in addition to Python’s standard library. You can read more about it here:

- <https://www.anaconda.com/>

Anaconda is designed to use a command-line tool called **conda** for installing additional packages whereas Python uses **pip**, although you can also use **pip** with Anaconda. Also note that the Anaconda download is much larger than the official Python one is because it has so many extra packages included with it.

If you are on a Windows PC and don’t have administrator privileges on your machine, then you might want to check out **WinPython**, which can be run from a USB:

- <https://winpython.github.io/>

There are many other Python distributions to choose from. You can see a bunch more here:

- <https://wiki.python.org/moin/PythonDistributions>

This book is focused on Python 3. The current version at the time of writing is Python 3.8. It is recommended that you use the official Python distribution rather than Anaconda, although the examples in this book should work for both. Any examples that use a specific feature only found in 3.8 or newer will be noted as such.

There are 32-bit and 64-bit distributions of Python. If you are unsure what your computer uses, you should opt for the 32-bit version if that is available. Newer Macs no longer support 32-bit, so in that case you only have one choice.

Installing on Windows

The <https://www.python.org/> website has a download section where you can download an installer for Python.

After the installer is downloaded, double-click it and go through the installation wizard. Here is the first screen you should see:



Fig. 1-1: Installing Python 3.8 via the installer

There is a checkbox in the wizard for adding Python to the path. If you don't have an older version of Python already installed or if you want to use the latest as your default Python, then I recommend that you check that checkbox. It is unchecked by default, as shown in the image above.

If you install Python to your path, it will allow you to run Python from the command line (cmd.exe) or Powershell by just typing python. If you install Python using the Windows Store, it will automatically add Python to your path.

The next page of the installation wizard allows you to enable or disable optional features:

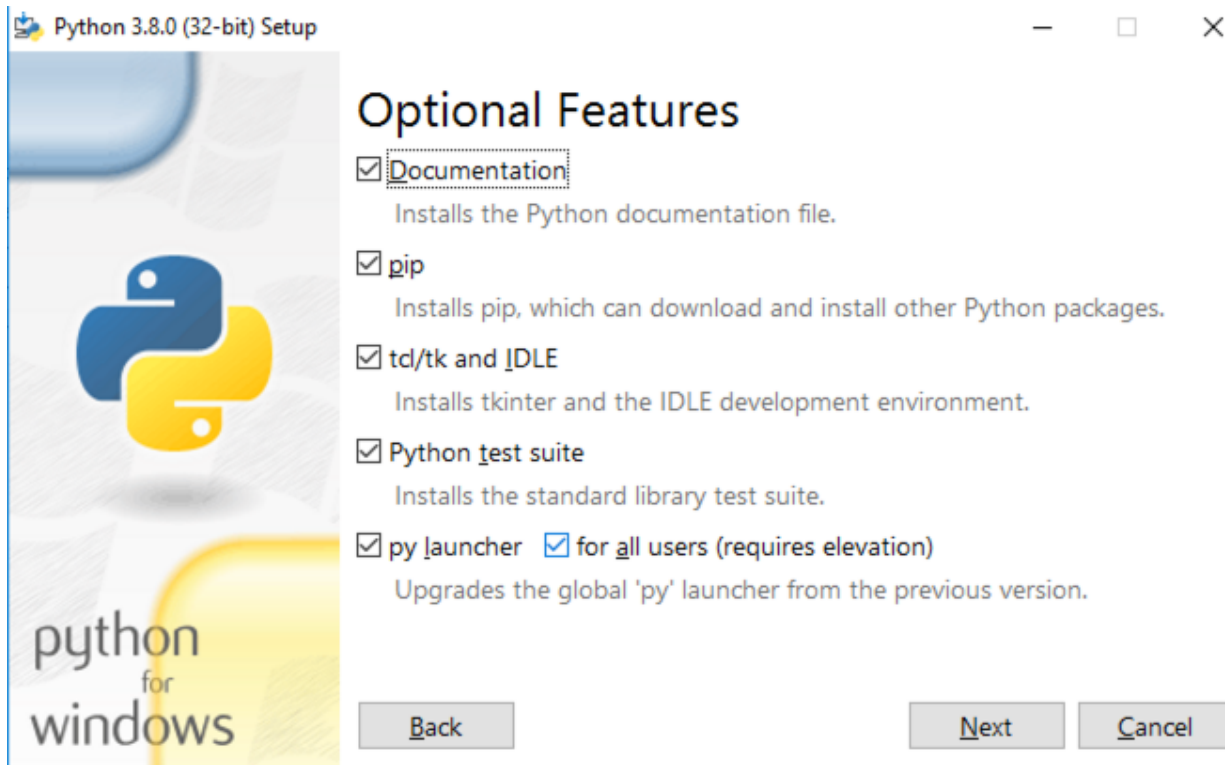


Fig. 1-2: Enabling Optional Python features

You can leave the defaults enabled, but if you are short on space, you should untick the “Python Test Suite” option. The next page of the wizard will allow you to enable Advanced Options:

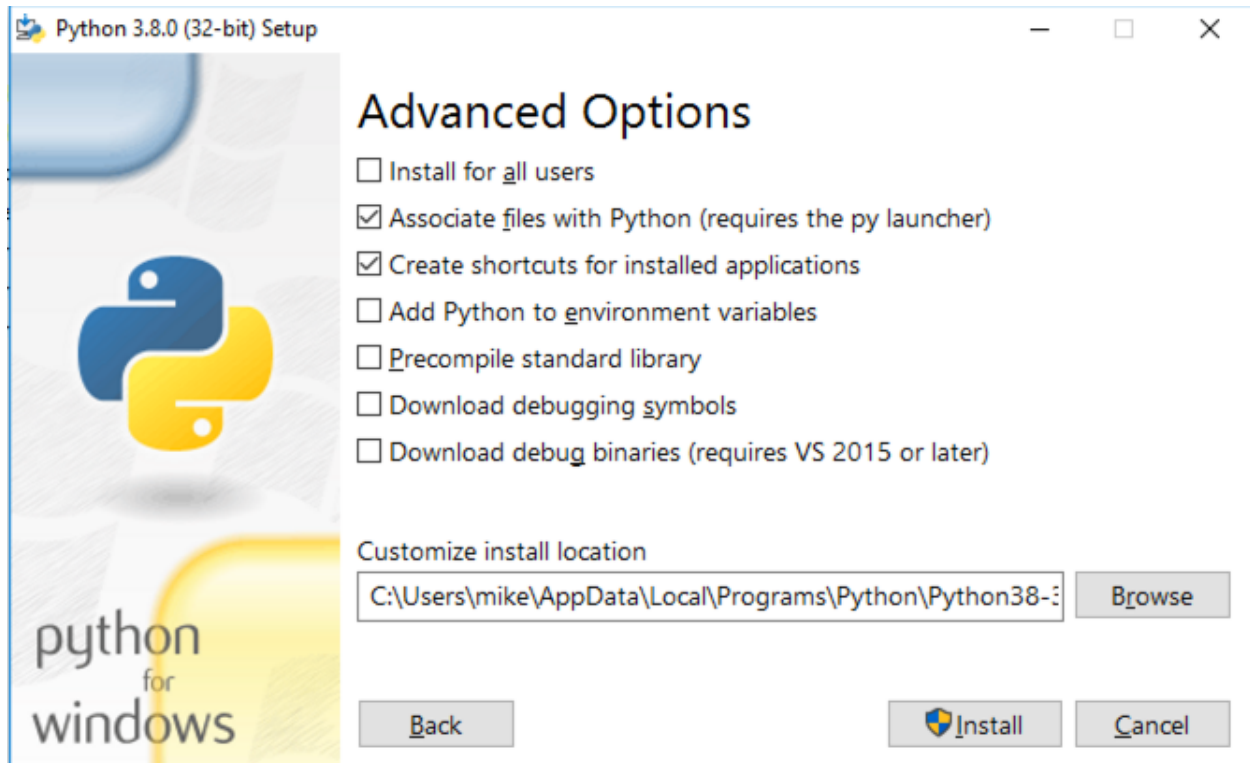


Fig. 1-3: Advanced Options

Here you can install Python for all users of the machine. You can also associate Python files with Python, create shortcuts, add Python to your environment and more. Most of the time, the defaults will be fine. However it's a good idea to go ahead and check the "Precompile standard library" as that can make Python run better on your machine.

When you press **Next** here, you will probably get a warning from Window's User Access Control:

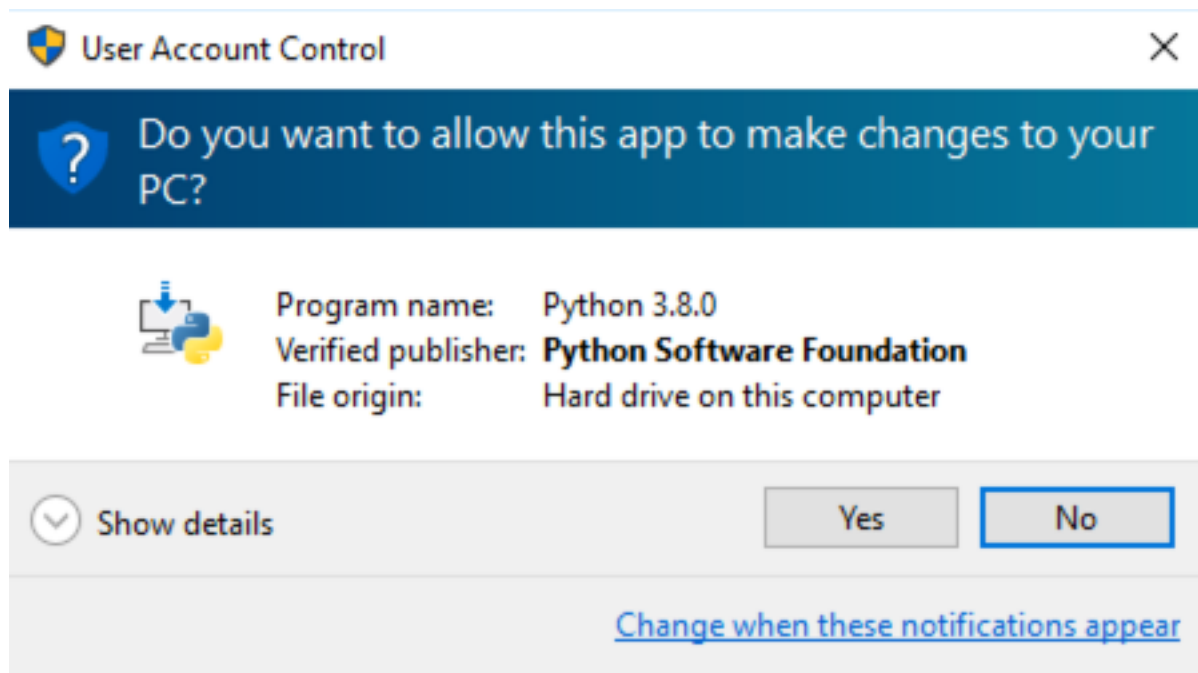


Fig. 1-4: User Access Control Warning

This is a verification step that asks if you really want to proceed with installing Python. Go ahead and press **Yes**. Now Python is being installed:

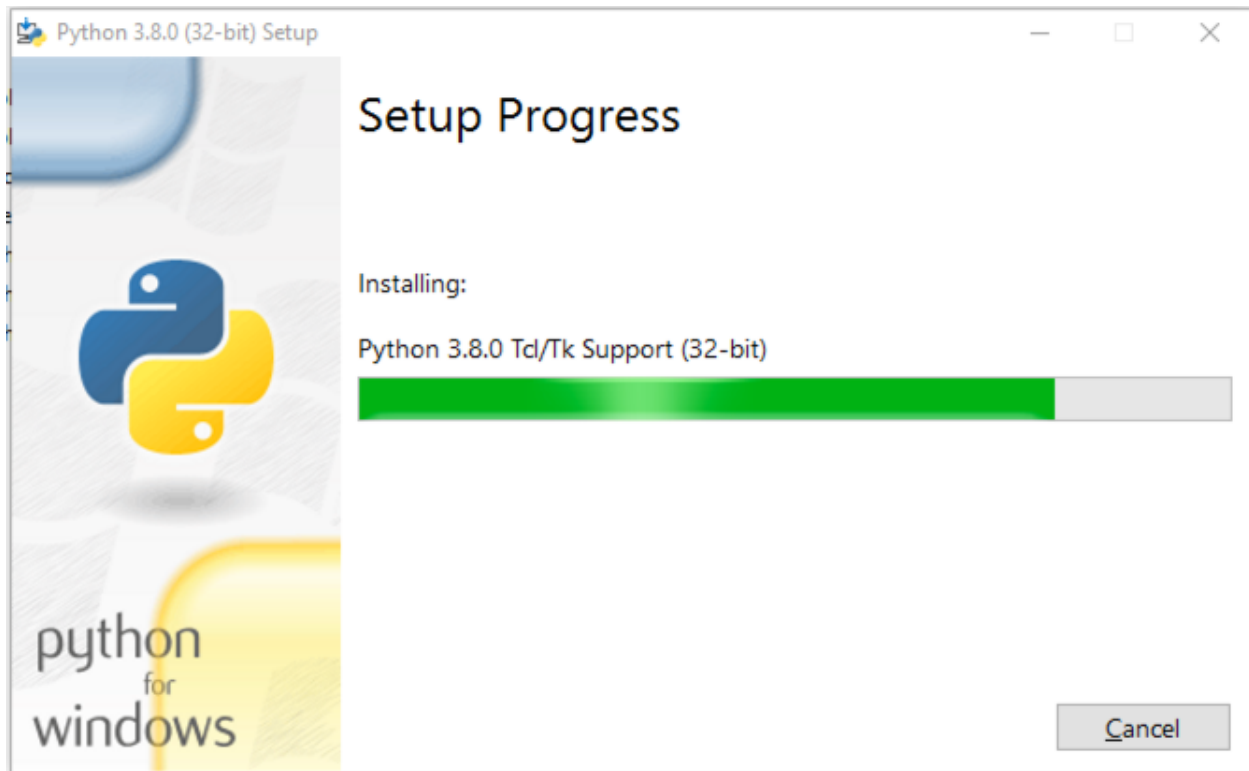


Fig. 1-5: Installing Python

Once the install finishes, you will see this final dialog:

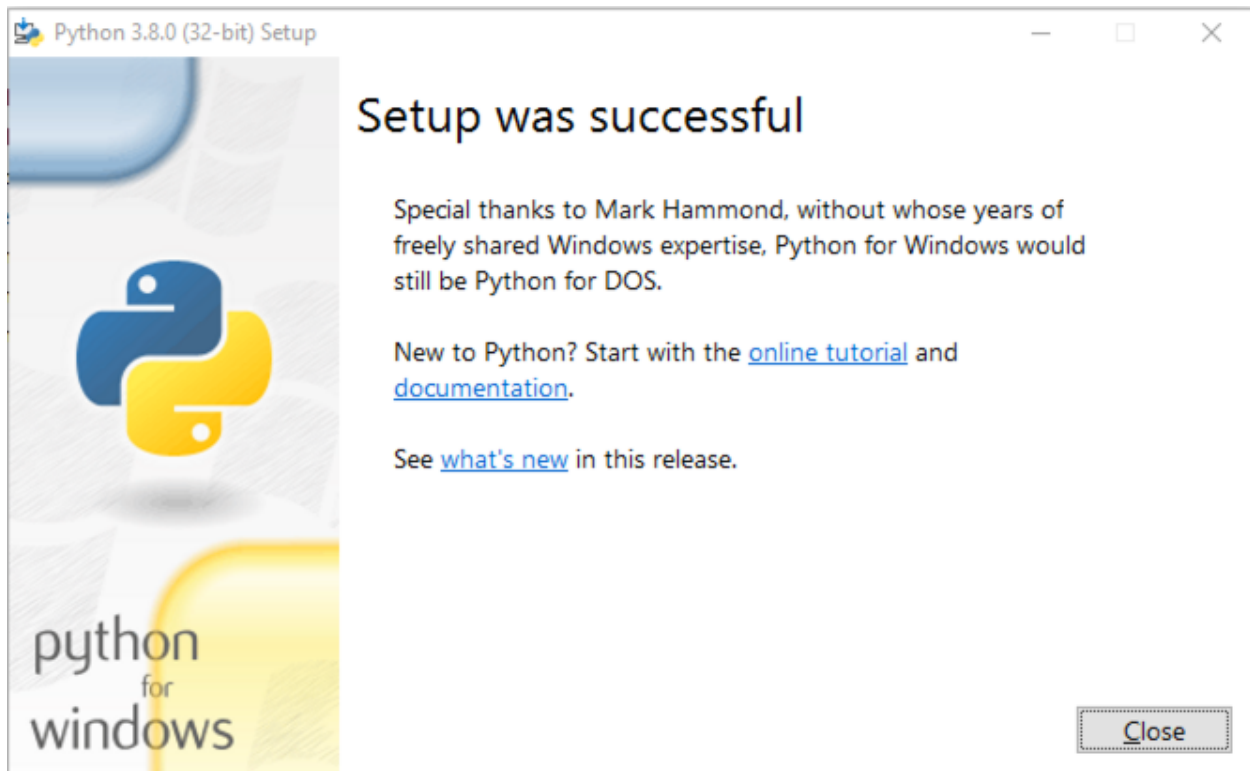


Fig. 1-6: Setup Successful

You now have Python installed on your Windows machine! Try running Python by opening `cmd.exe`:

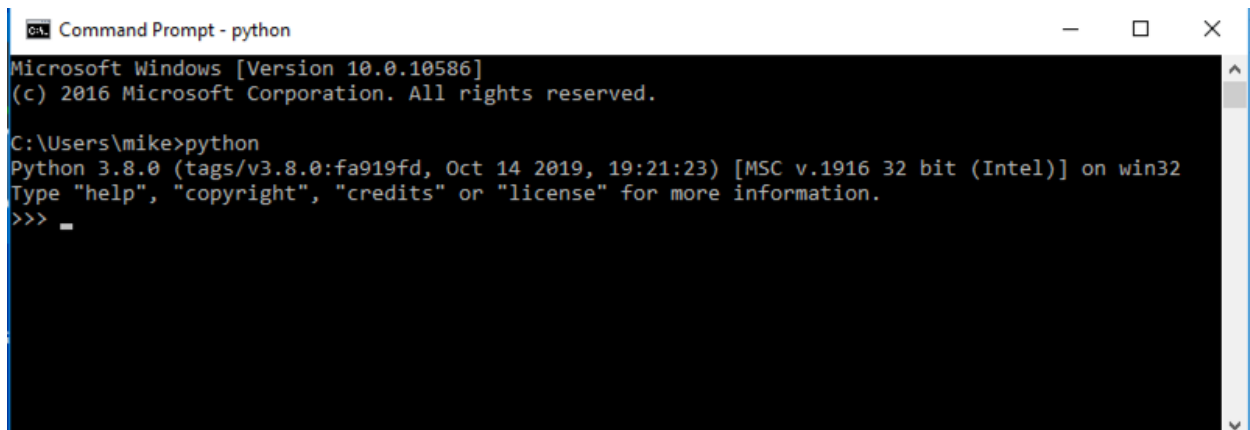


Fig. 1-7: Running Python in cmd.exe

You should see something like the above. To exit, you can press **CTRL+D** on Linux and Mac, **CTRL+Z** on Windows, or type `exit()` on any platform and press **Enter**.

Installing on Mac

Macs usually come with Python pre-installed. However, if you want the latest version of Python, then you may need to download Python.

The <https://www.python.org/> website has a download section where you can also download a Python installer for Mac. Once downloaded, you will need to double-click it to install Python. You may need to tell your Mac to allow you to install Python as the system might bring up a dialog box that warns you about installing programs downloaded from the internet.

Note that the App Store does not have Python in it, so using the Python website is the way to go.

Let's take a moment to learn how to install Python on a Mac. Here's the first thing you will see when you double-click the downloaded pkg file:



Fig. 1-8: Installing Python on Mac OSX

This screen basically tells you what you are about to install and that you will need to install some SSL certificates as well. Go ahead and press **Continue**:

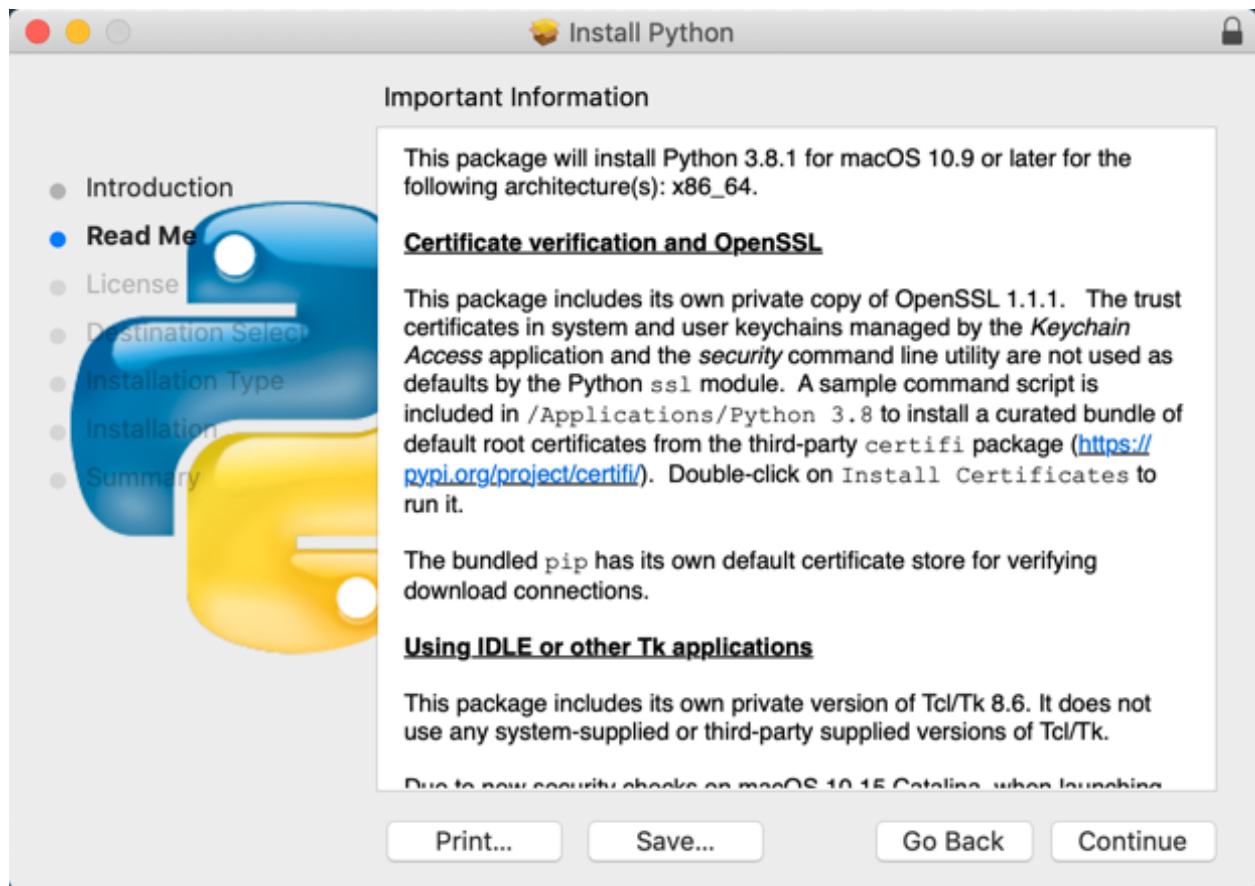


Fig. 1-9: Installing Python on Mac OSX (Read More)

This page gives you more information about the SSL certificate as well as general information about using IDLE and Tkinter on a Mac. IDLE is Python's built-in code editor. You will learn more about that in the next chapter. Tkinter is a Python library that you can use to create cross-platform graphical user interfaces.

Tkinter is the library that is used to create IDLE, the Python editor that comes with Python.

If you are interested, read through the information on this page. Otherwise, go ahead and **Continue**:



Fig. 1-10: Installing Python on Mac OSX (License Agreement)

This is Python's license agreement page. It also has a little bit of history about Python on it. Feel free to check it out or skip it and press **Continue**:



Fig. 1-11: Installing Python on Mac OSX (Install Destination)

This page allows you to choose which disk on your computer you want to install Python. I usually use the default, but if you want you can change it to some other location.

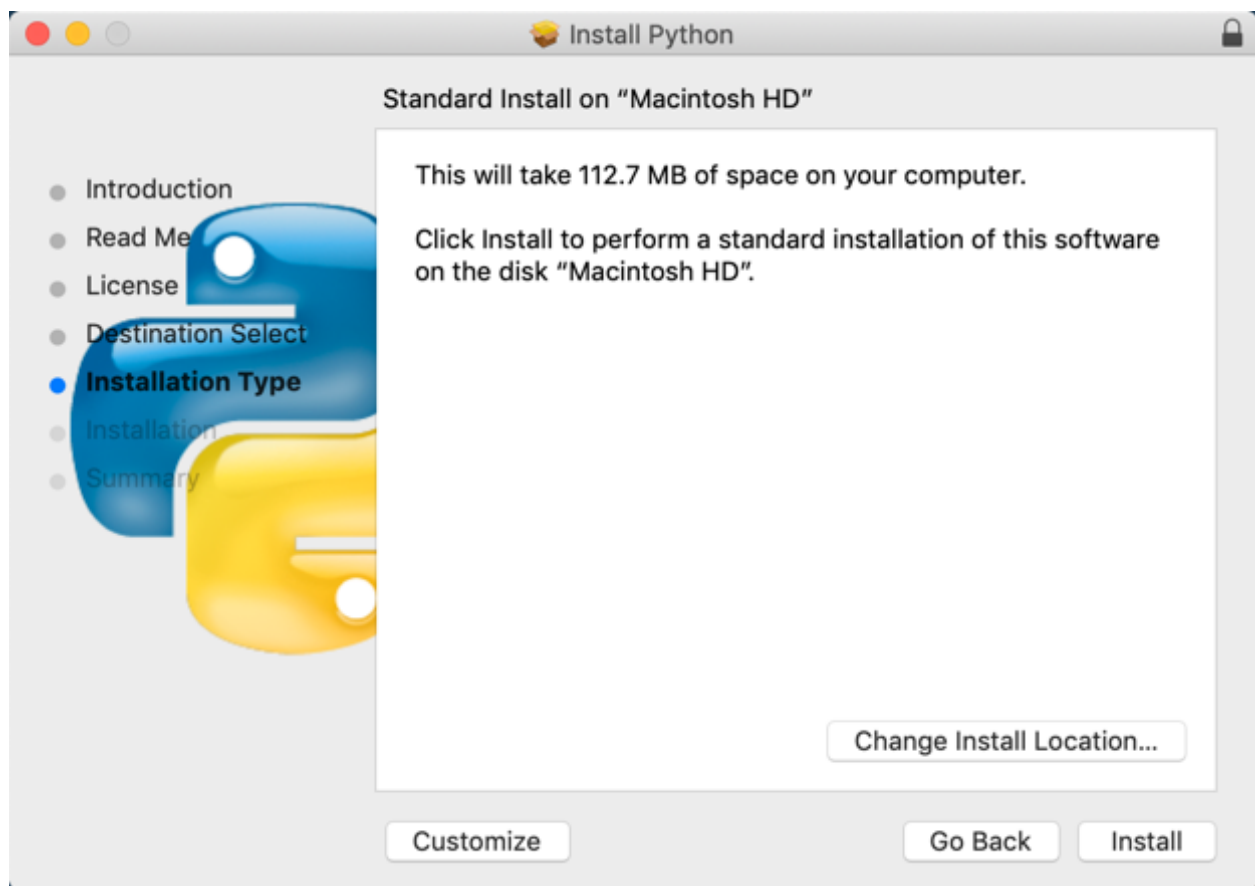


Fig. 1-12: Installing Python on Mac OSX (Standard Install)

This page allows you to choose *which folder* to install Python to, in contrast to the previous page which lets you pick *which disk* to install to.

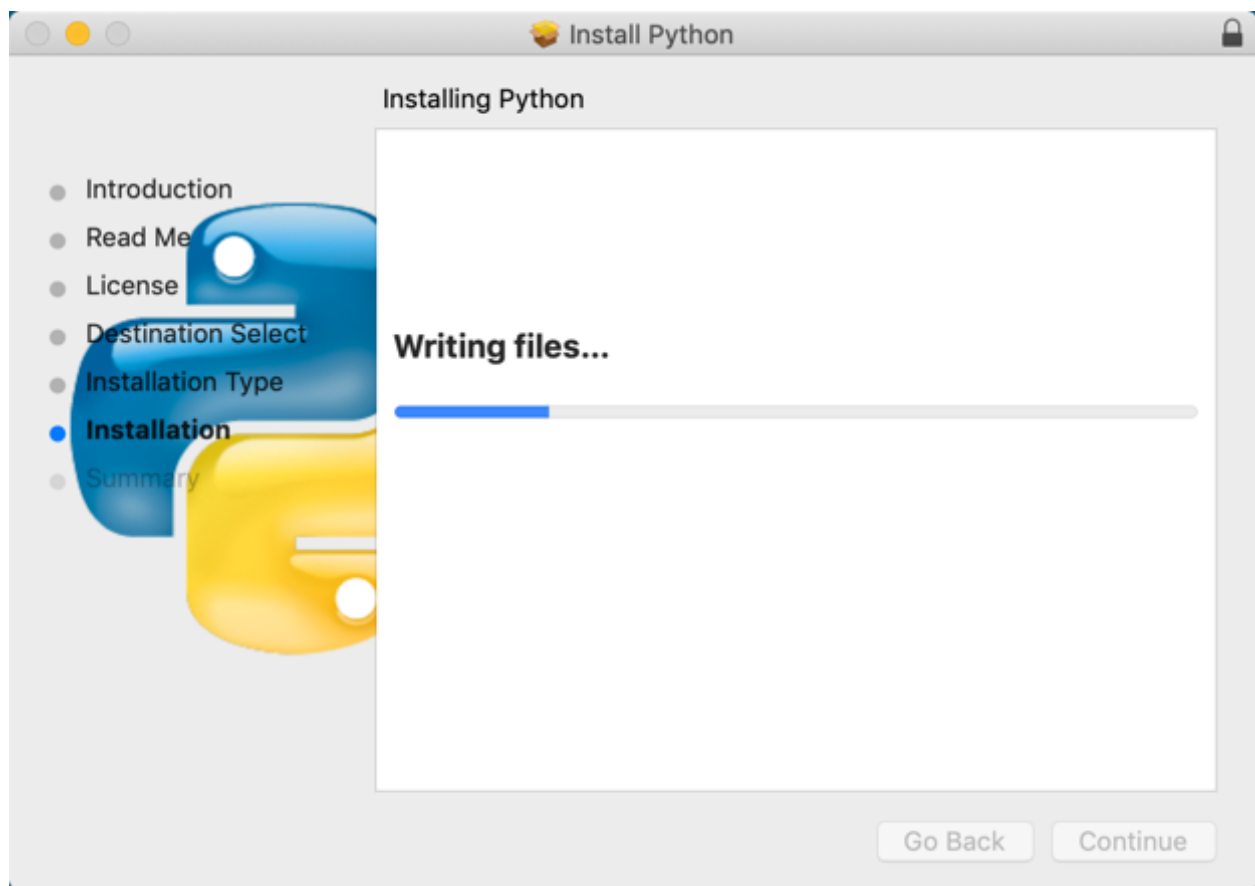


Fig. 1-13: Installing Python on Mac OSX (Actual Installation)

This page shows the installation as it happens. You can wait and watch Python get installed, or go get something to drink. Either way, Python will be installed before too long.

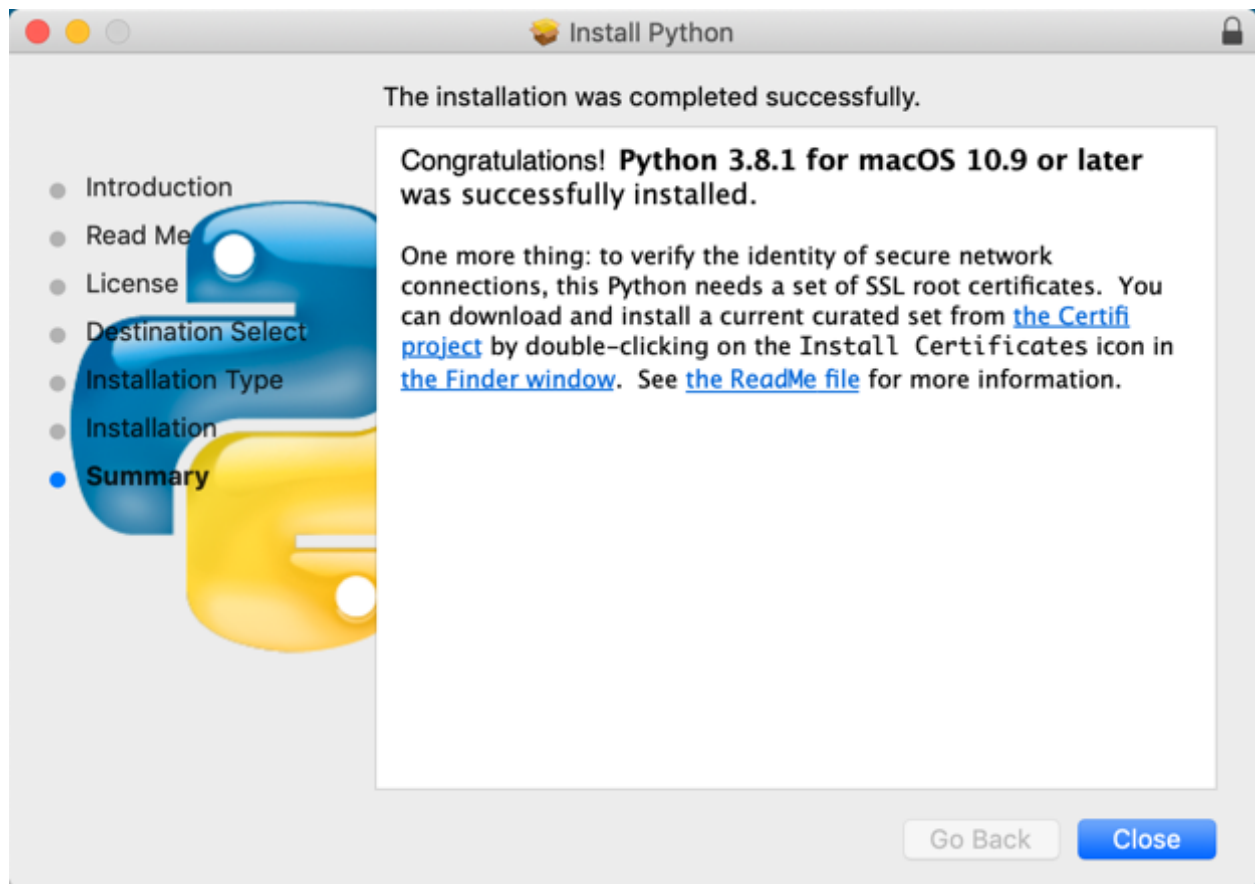


Fig. 1-14: Installing Python on Mac OSX (Finished)

Here is the last page of the installation wizard.

Installing on Linux

Linux also usually comes with Python pre-installed, although it will likely not be the latest version of Python. The Python website has source downloads and directions that you can use to build the latest Python for your Linux distribution.

The source download is Python source code with some build scripts included that will allow you to build and install Python on Linux.

For full instructions, you should read the dev guide:

- <https://devguide.python.org/setup/>

Sometimes you can also install a pre-built copy of the latest Python using your package manager. One good source for Python on Linux is the “deadsnakes” PPA. You will need to use Google to find it, but that makes it much easier to install another version of Python on Linux.

Android / iOS

You can also run Python on Android and iOS via downloadable applications. **Pydroid** is a popular application for Android while **Pythonista** is one of the popular choices for iOS. Trying to write code on a phone can be really problematic due to the on-screen keyboards. If you must go this route, you may want to use a tablet.

Other Operating Systems

Python can run on Raspberry Pi as well. If you do not have a computer, this is one of the most cost effective ways to get Python as a Raspberry Pi can cost as little as \$10. Of course, you will need to hook it up to a monitor, keyboard and mouse. Most of the time, a Raspberry Pi will also need an SD card for storage. But they are a very feasible development environment.

Other Python Variants

In addition to Anaconda, Python has several other variants that are worth mentioning:

- Jython - an implementation of Python written in Java that allows you to use Java code in Python
- IronPython - an implementation of Python written in .NET
- PyPy - written in RPython, PyPy has a just-in-time (JIT) compiler that makes it much faster than regular Python

The main reason for trying out these other implementations of Python is for speed or flexibility. For example, if you are already familiar with .NET or Java, then you might find IronPython or Jython a bit easier to jump into. Another reason to use Jython or IronPython is because you have pre-existing code in Java or .NET that you still need to use with Python.

In the case of PyPy, I usually recommend it if you have a slow Python program and you need a simple way to speed it up. Try running it with PyPy and you might be surprised at the performance improvement. Note that none of these variants are completely compatible with all of Python's 3rd party packages, so if your program uses one it may not work with these variants.

Wrapping Up

Most of the time, installing Python is straight-forward and easy to do. It can get tricky when you need to have multiple versions of Python installed on your machine at the same time, but if you are just starting out I think you'll find the installation process pretty painless.

Now that you have Python installed, you can congratulate yourself. You have started on a new endeavor and you have just taken the first step! However, before you try running Python, you may want to read the next chapter where you will learn about additional tools that will help you get the most out of your Python adventure!

Chapter 2 - Python Editors

The Python programming language comes with its own built-in Integrated Development Environment (IDE) called **IDLE**. The name, IDLE, supposedly came from the actor, Eric Idle, who was a part of the Monty Python troupe, which is what Python itself is named after.

IDLE comes with Python on Windows and some Linux variants. You may need to install IDLE separately on your particular flavor of Linux or on Mac if you plan to use the Python that came with the operating system. You should check out the Python website for full instructions on how to do so as each operating system is different.

Here are some of the reasons that Integrated Development Environments are useful:

- They provide syntax highlighting which helps prevent coding errors
- Autocomplete of variable names and built-in names
- Breakpoints and debugging.

On that last point, breakpoints tell the debugger where to pause execution. Debugging is the process of going through your code step-by-step to figure out how it works or to fix an issue with your code.

IDLE itself has other attributes that are useful, such as access to Python documentation, easy access to the source code via the Class Browser, and much more. However, IDLE is not the only way to code in Python. There are many useful IDEs out there. You can also use a text editor if you prefer. Notepad, SublimeText, Vim, etc., are examples of text editors. Text editors do not have all the features that a full-fledged IDE has, but tend to have the advantage of being simpler to use.

Here is a shortlist of IDEs that you can use to program in Python:

- PyCharm
- Wing Python IDE
- VS Code (also called Visual Studio Code)
- Spyder
- Eclipse with PyDev

PyCharm and WingIDE both have free and paid versions of their programs. The paid versions have many more features, but if you are just starting out, their free offerings are quite nice. VS Code and Spyder are free. VS Code can also be used for coding in many other languages. Note that to use VS Code effectively with Python, you will need to install a Python extension. You can also use the PyDev plugin for Eclipse to program in Python.

Other popular editors for Python include SublimeText, vim, emacs, and even Notepad++. These editors may not be 100% up-to-date on the syntax of the language, but you can use them for multiple programming languages.

But let's back up a bit and talk about Python's basic console, also known as the REPL, which stands for Read Evaluate Print Loop.

What About the REPL?

REPL or READ, EVAL, PRINT, LOOP is basically Python's interpreter. Python allows you to type code into an interpreter which will run your code live and let you learn about the language. You can access the interpreter, or REPL, by running Python in your terminal (if you are on Mac or Linux) or command console (if you are on Windows).

On Windows, you can go to the Start menu and search for cmd or "Command Prompt" to open the console or terminal:

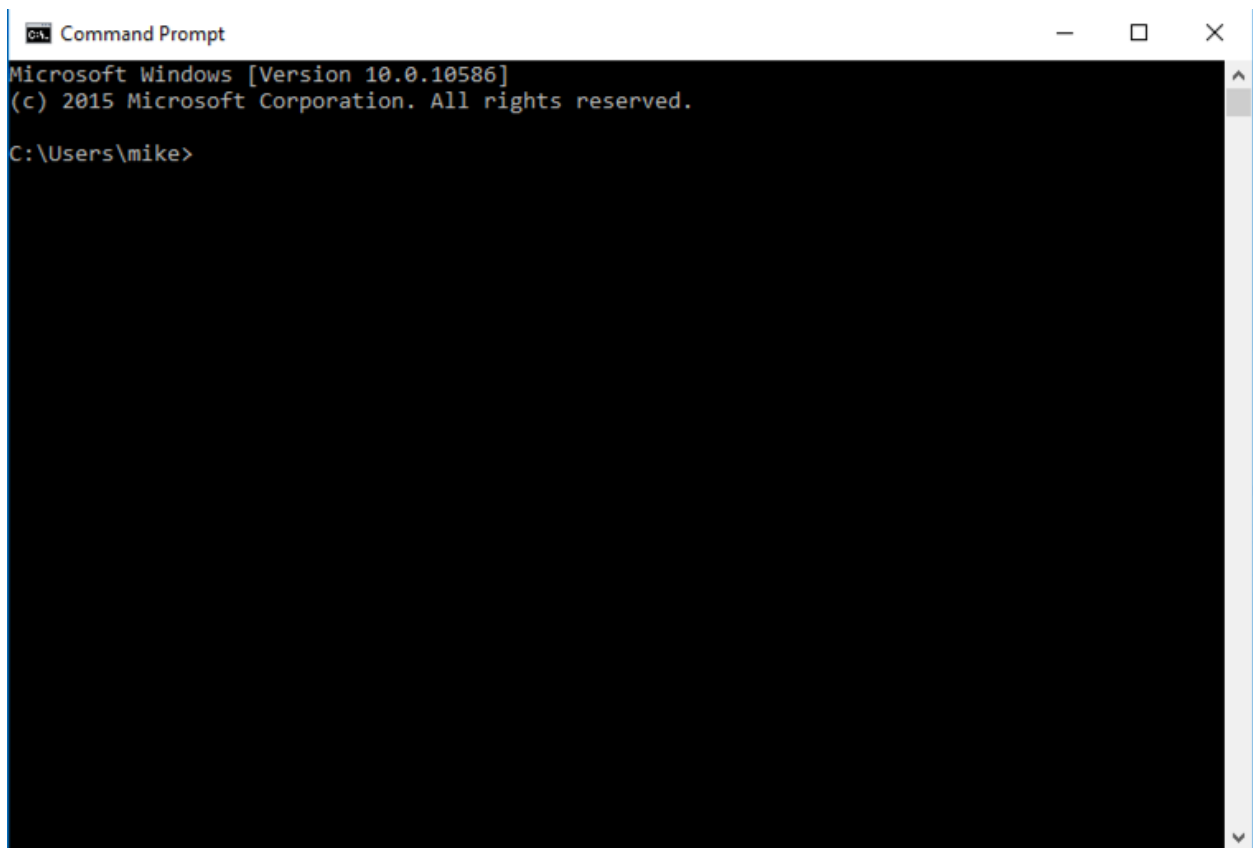
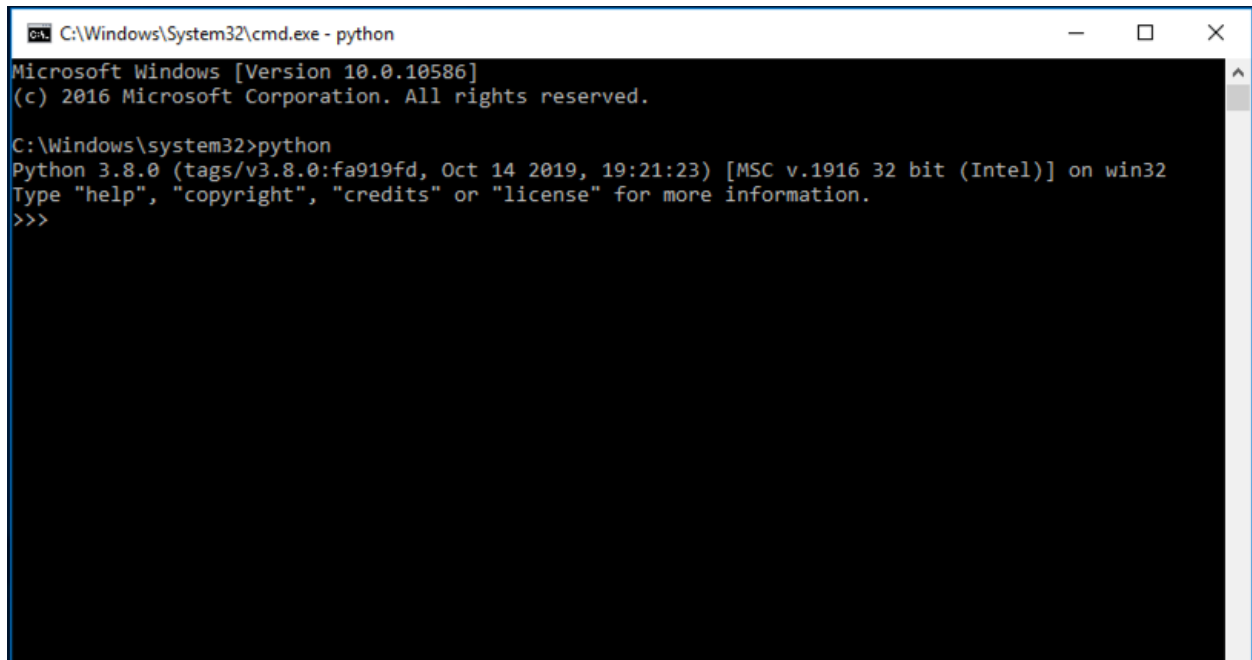


Fig. 2-1: Running the Command Prompt in Windows

Once you have the terminal open you can try typing `python`. You should see something like this:



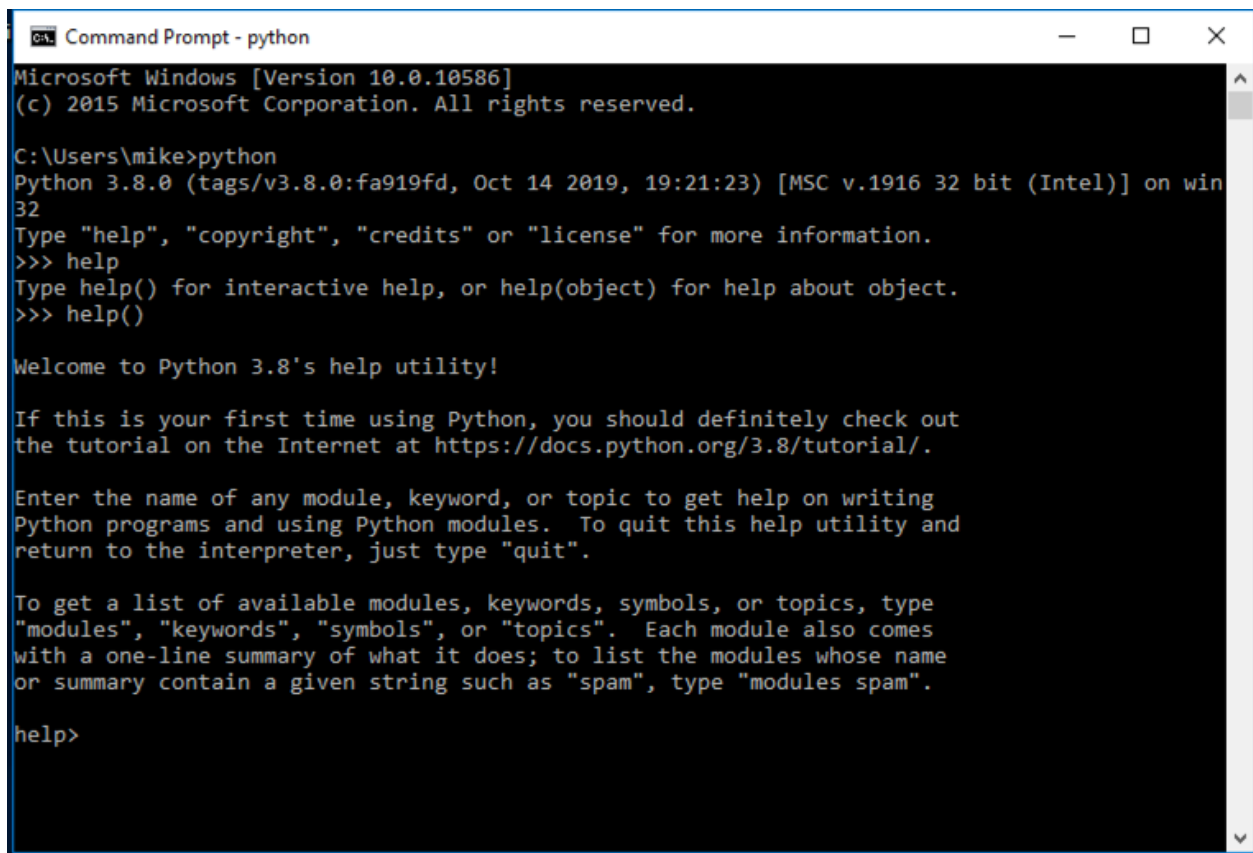
```
C:\Windows\System32\cmd.exe - python
Microsoft Windows [Version 10.0.10586]
(c) 2016 Microsoft Corporation. All rights reserved.

C:\Windows\system32>python
Python 3.8.0 (tags/v3.8.0:fa919fd, Oct 14 2019, 19:21:23) [MSC v.1916 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Fig. 2-2: Running Python in cmd.exe

If this doesn't work and you get an "Unrecognized Command" or some other error, then Python may not be installed or configured correctly. On Windows, you may need to add Python to your system's path or you can just type out the full path to Python in your command console. For example, if you installed Python in `C:\Python\Python38`, then you can run it using `cmd.exe` like you did above, but instead of typing `python`, you would type `C:\Python\Python38\python`.

If you need to get help in the REPL, you can type `help()`:



```
Command Prompt - python
Microsoft Windows [Version 10.0.10586]
(c) 2015 Microsoft Corporation. All rights reserved.

C:\Users\mike>python
Python 3.8.0 (tags/v3.8.0:fa919fd, Oct 14 2019, 19:21:23) [MSC v.1916 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> help
Type help() for interactive help, or help(object) for help about object.
>>> help()

Welcome to Python 3.8's help utility!

If this is your first time using Python, you should definitely check out
the tutorial on the Internet at https://docs.python.org/3.8/tutorial/.

Enter the name of any module, keyword, or topic to get help on writing
Python programs and using Python modules.  To quit this help utility and
return to the interpreter, just type "quit".

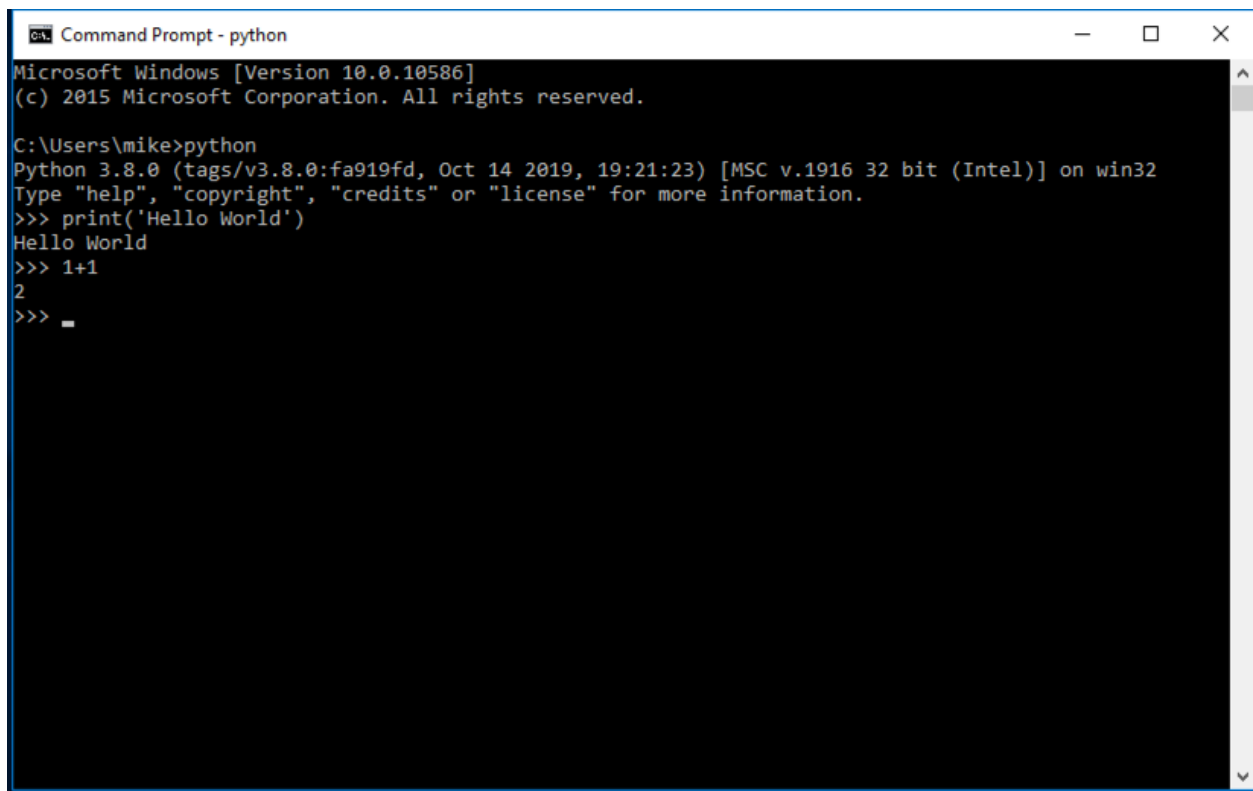
To get a list of available modules, keywords, symbols, or topics, type
"modules", "keywords", "symbols", or "topics".  Each module also comes
with a one-line summary of what it does; to list the modules whose name
or summary contain a given string such as "spam", type "modules spam".

help>
```

Fig. 2-3: REPL Help

You can type live Python code into the REPL and it will be immediately evaluated, which means the code will run as soon as you press enter.

Here's how you would print out "Hello World" and add some numbers in the REPL:



```
Command Prompt - python
Microsoft Windows [Version 10.0.10586]
(c) 2015 Microsoft Corporation. All rights reserved.

C:\Users\mike>python
Python 3.8.0 (tags/v3.8.0:fa919fd, Oct 14 2019, 19:21:23) [MSC v.1916 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> print('Hello World')
Hello World
>>> 1+1
2
>>> _
```

Fig. 2-4: REPL Example Code

Python comes with its own code editor called IDLE. Let's learn about that next!

Getting Started with IDLE

IDLE is a good place to start learning Python. Once you have it installed, you can start it up and the initial screen will look like this:

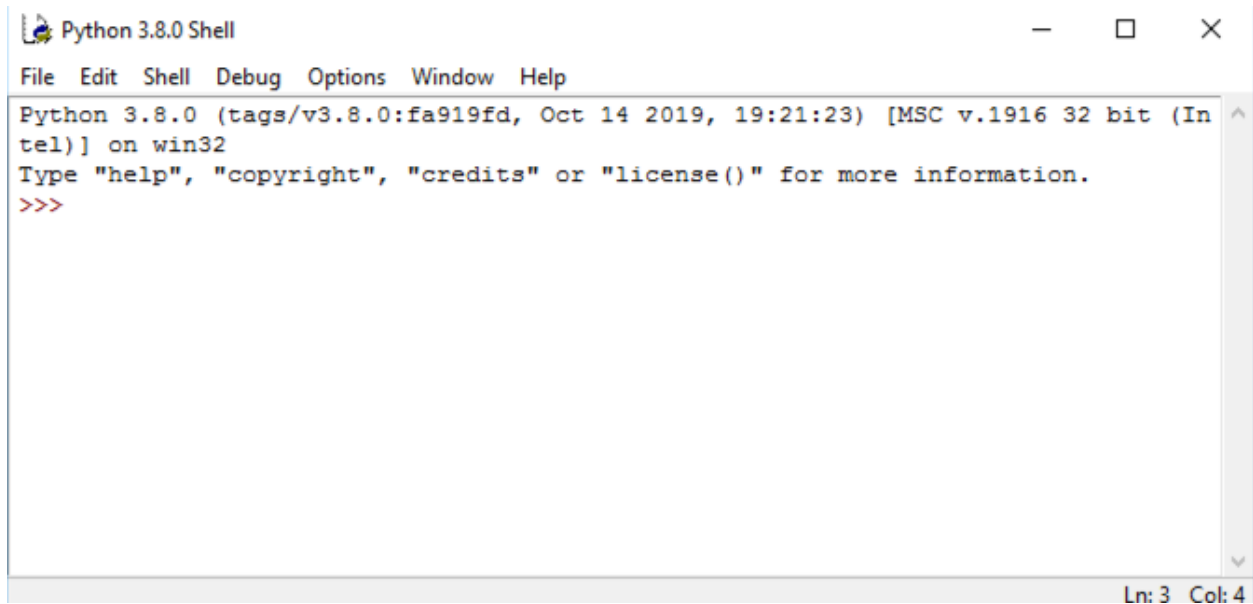


Fig. 2-5: The IDLE Shell

This is a REPL. You can enter code here and it will be evaluated as soon as you press the Return or Enter key.

If you want to actually write a full program, then you will need to open up the editor view by going to **File** -> **New**.

You should now have the following dialog on your screen:

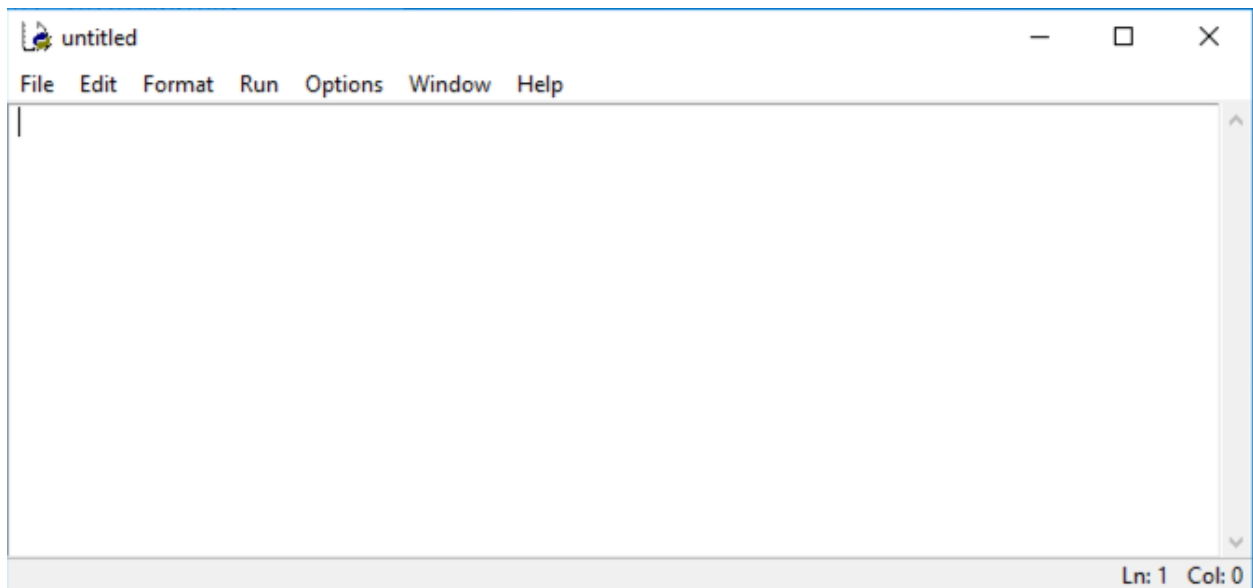


Fig. 2-6: The IDLE Editor

You can enter your code here and save it.

Running Your Code

Let's write a small bit of code in our code editor and then run it. Enter the following code and then save the file by going to **File** -> **Save**.

```
1 print('Hello World')
```

To run this code in IDLE, go to the **Run** menu and choose the first option labeled **Run Module**:

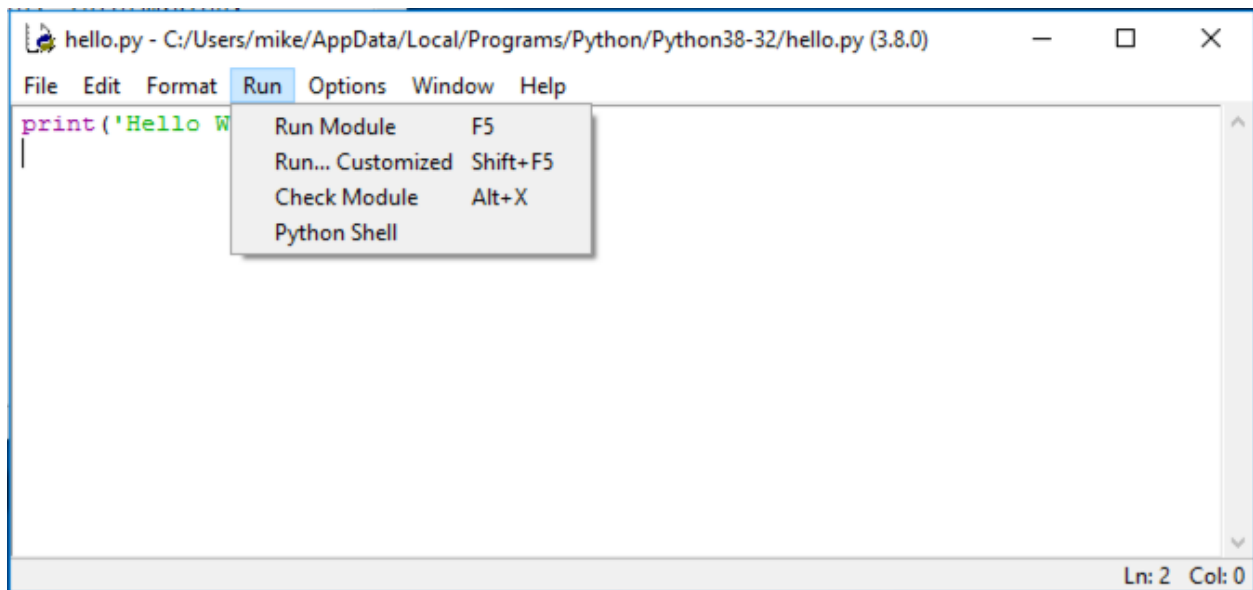


Fig. 2-7: Running Code in IDLE

When you do this, IDLE will switch to the Shell and show you the output of your program, if there is any:

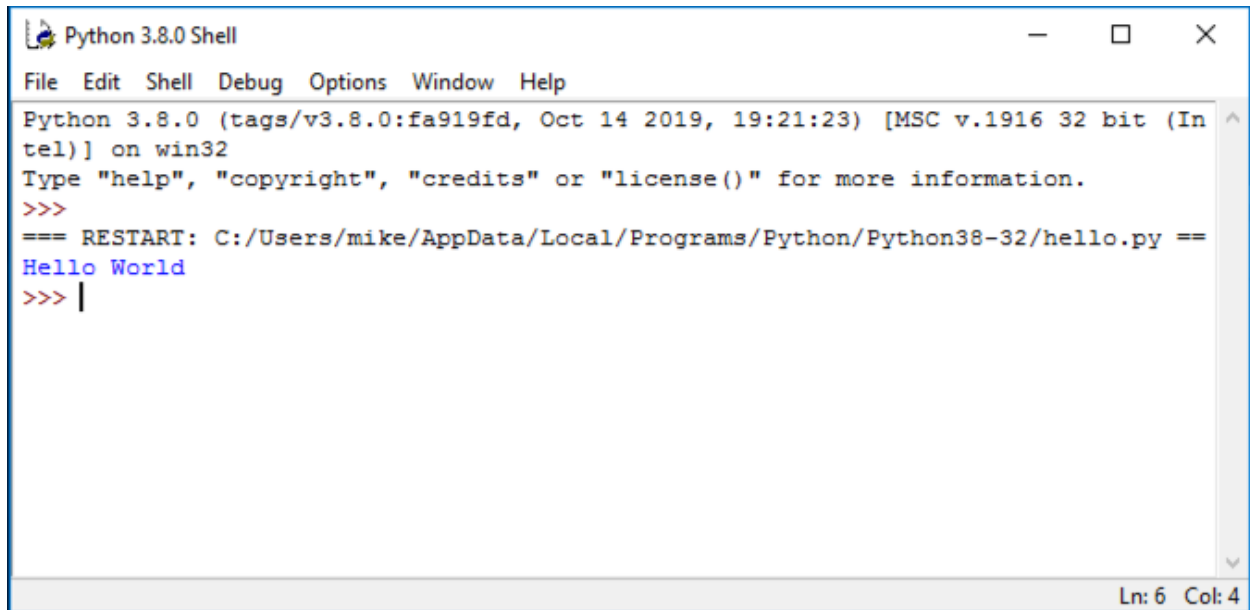


Fig. 2-8: Output of Code in IDLE

You can also use the **Run** menu's **Check Module** option to check your code for syntax errors.

Accessing Help / Documentation

Sometimes you need help. Fortunately IDLE has some built-in help about itself and the Python language, too! You can access help about IDLE by going to the **Help** menu and choosing **IDLE Help**:

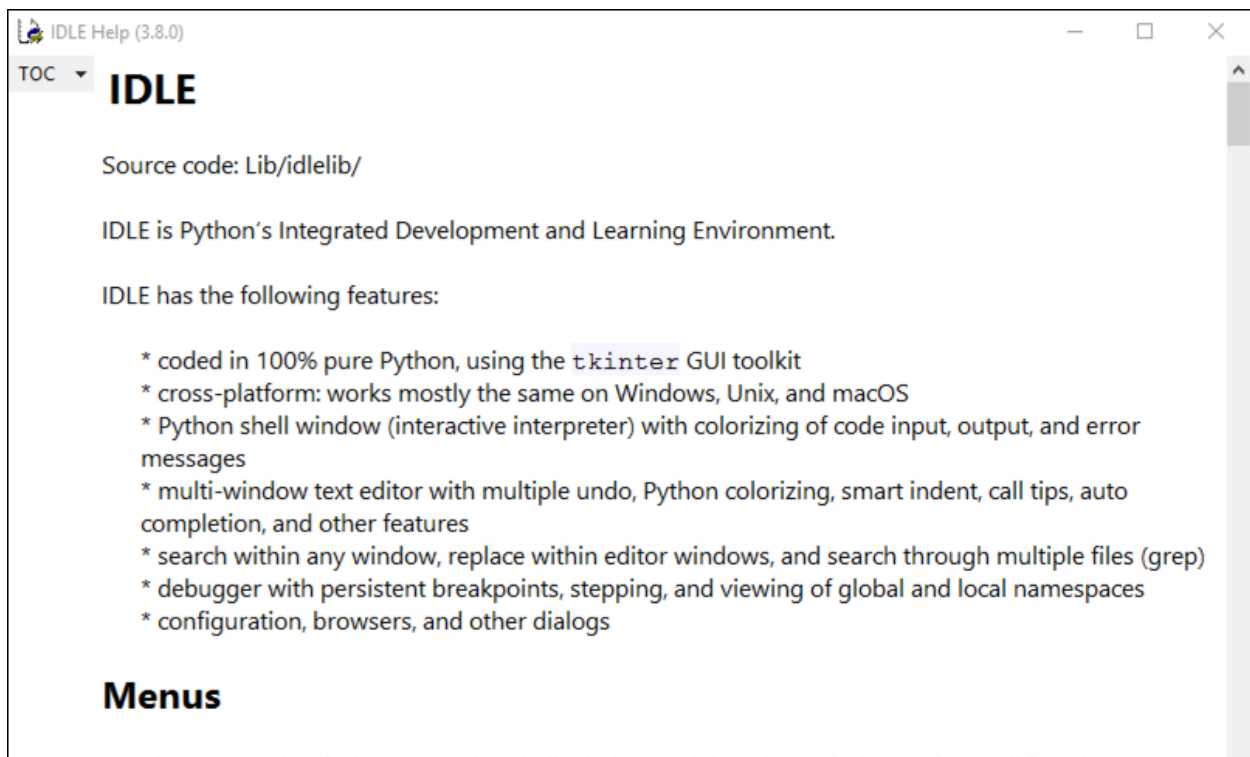


Fig. 2-9: IDLE Help

If you'd rather look up how something works in the Python language, then go to the **Help** menu and choose **Python Docs** or press F1 on your keyboard:

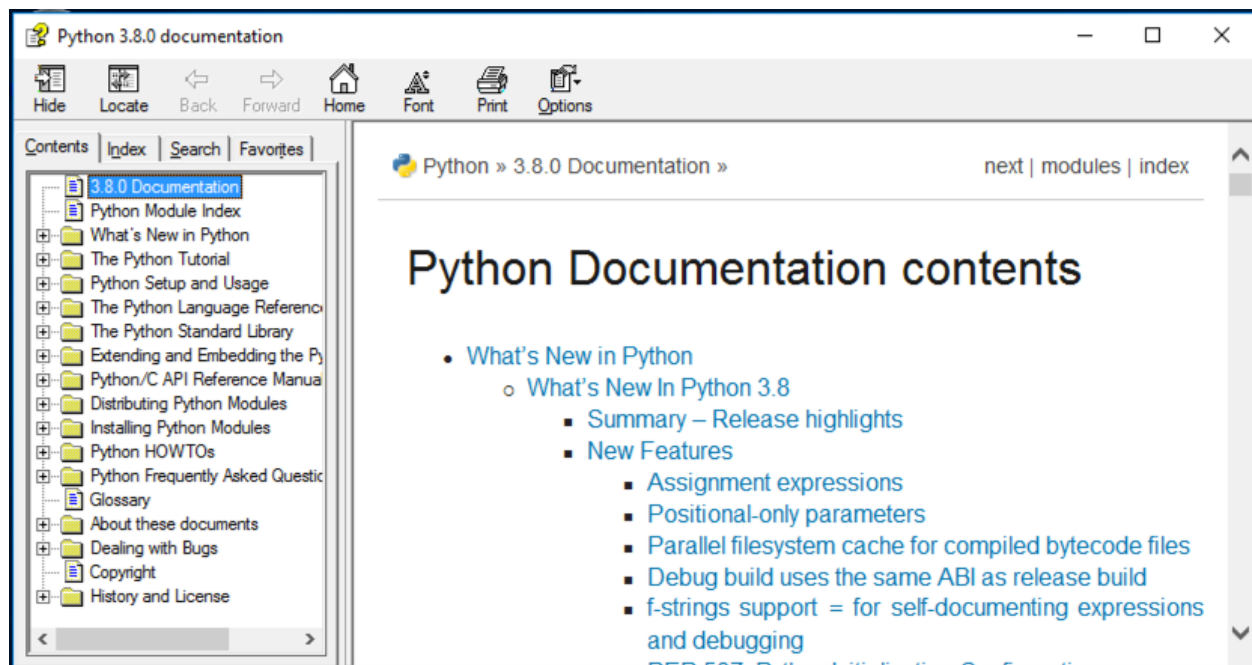


Fig. 2-10: IDLE Python Documentation

This will show you Python's official documentation. Depending on your O/S this may load local help files, or start a browser to show the official on-line help documents.

Restarting the Shell

Let's go back to the Shell screen of IDLE rather than the editor. It has several other functions that are worth going over. The first is that you can restart the shell.

Restarting the shell is useful when you need to start over with a clean slate but don't want to close and reopen the program. To restart the shell, go to the **Shell** menu and choose **Restart Shell**:

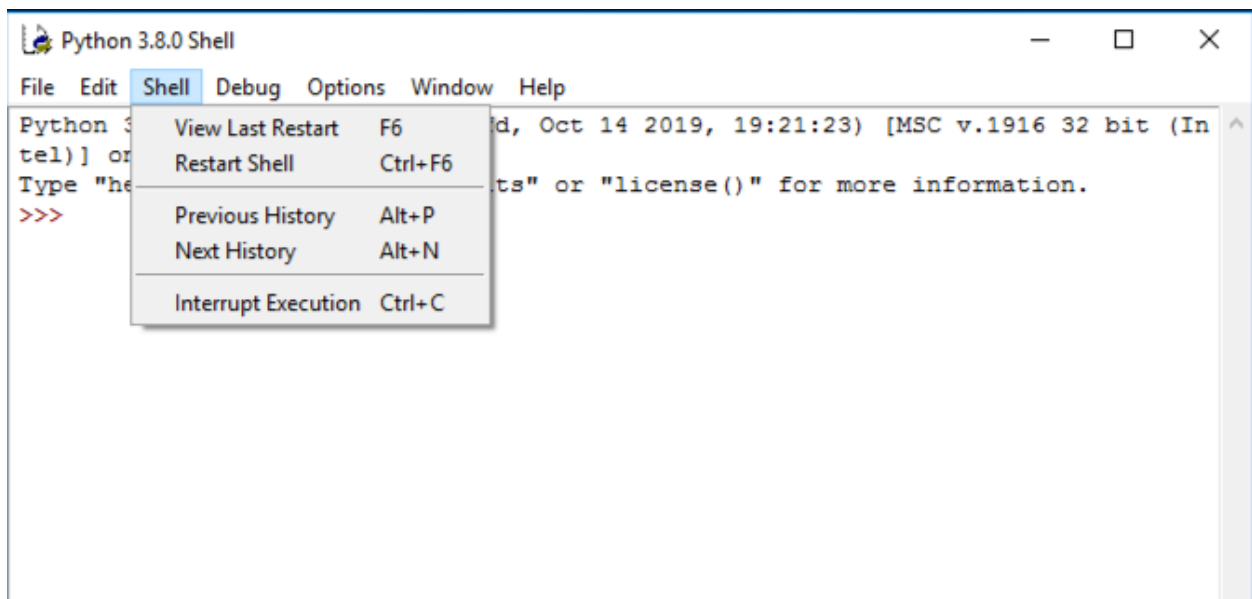


Fig. 2-11: IDLE Restart Menu

If you haven't restarted the shell before, then your screen will look like this:

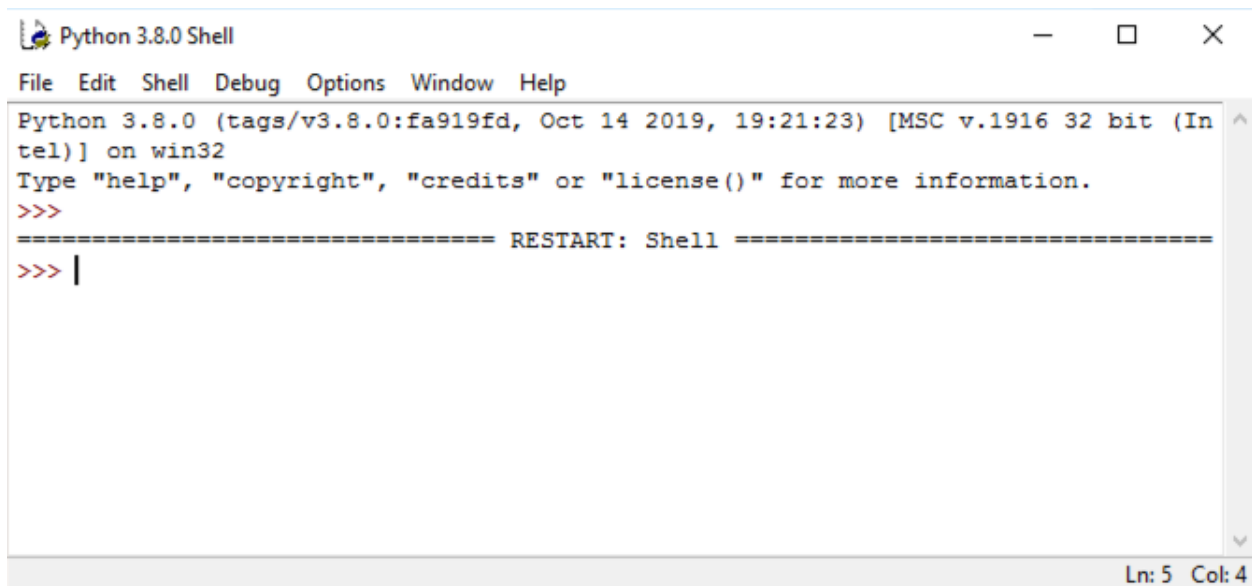


Fig. 2-12: IDLE Restarted

This tells you that your shell has restarted.

Module Browser

IDLE comes with a handy tool called the **Module Browser**. This tool can be found in the **File** menu. When you open it, you will see the following:

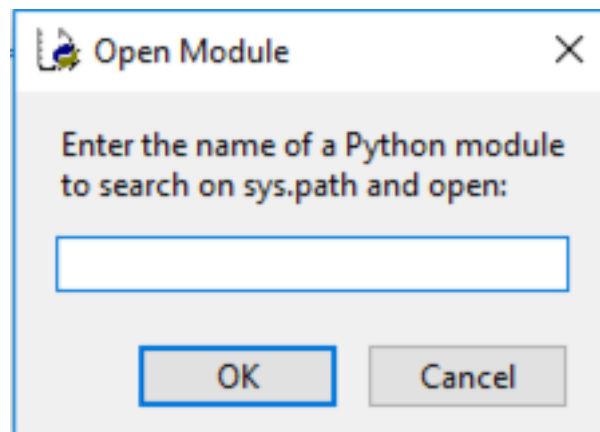


Fig. 2-13: IDLE Opening Module Browser

Modules in Python are code that the Python core development team has created for you. You can use the **Module Browser** to browse the source code of Python itself.

Try entering the following into the dialog above: `os`. Then press OK.

You should now see the following:

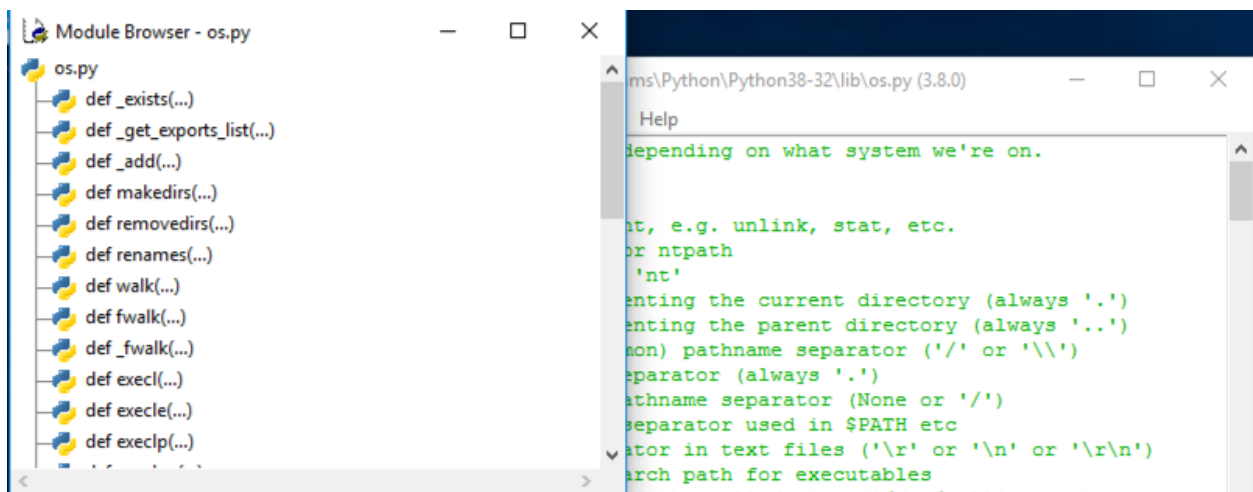


Fig. 2-14: IDLE Opening Module Browser

This allows you to browse the source code for `os.py`. You can double-click anything in the **Module Browser** and it will jump to the beginning of where that code is defined in IDLE's code editor.

Path Browser

Another useful tool that you can use in IDLE is the **Path Browser**. The **Path Browser** allows you to see where Python is installed and also what paths Python uses to import modules from. You will learn more about importing and modules later on in this book.

You can open it by going to **File** and then **Path Browser**:

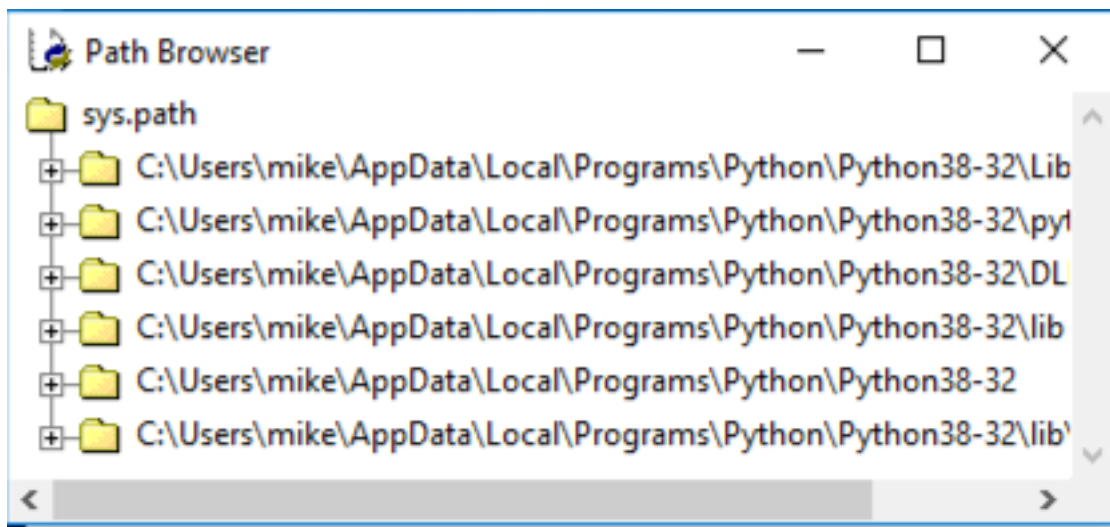


Fig. 2-15: IDLE Path Browser

The **Path Browser** is a good way to diagnose issues with importing modules. It can show you that you might not have Python configured correctly. Or it might show you that you have installed a 3rd party module in the wrong location.

Getting Started with PyCharm Community Edition

PyCharm is a commercial Python IDE from a company called JetBrains. They have a professional version, which costs money, and a community edition, which is free. PyCharm is one of the most popular choices for creating and editing Python programs.

PyCharm Professional has tons of features and a great debugger. However, if you are a beginner, you may find all the functionality in this software to be a bit overwhelming.

To get a copy of PyCharm Community Edition, you can go to the following website:

<https://www.jetbrains.com/pycharm/>

The Community Edition does not have all the features that PyCharm Professional has. But that is okay when you are new to Python. If you would like to try PyCharm, go ahead and download and install the software.

When you run PyCharm it may ask you to import settings. You can ignore that or import settings if you have used PyCharm previously and already have some.

Next, you will probably need to accept their privacy policy / EULA. Depending on the operating system, you may also get asked what theme to apply. The default is Darkula on Windows.

At this point you should see the following Welcome banner:

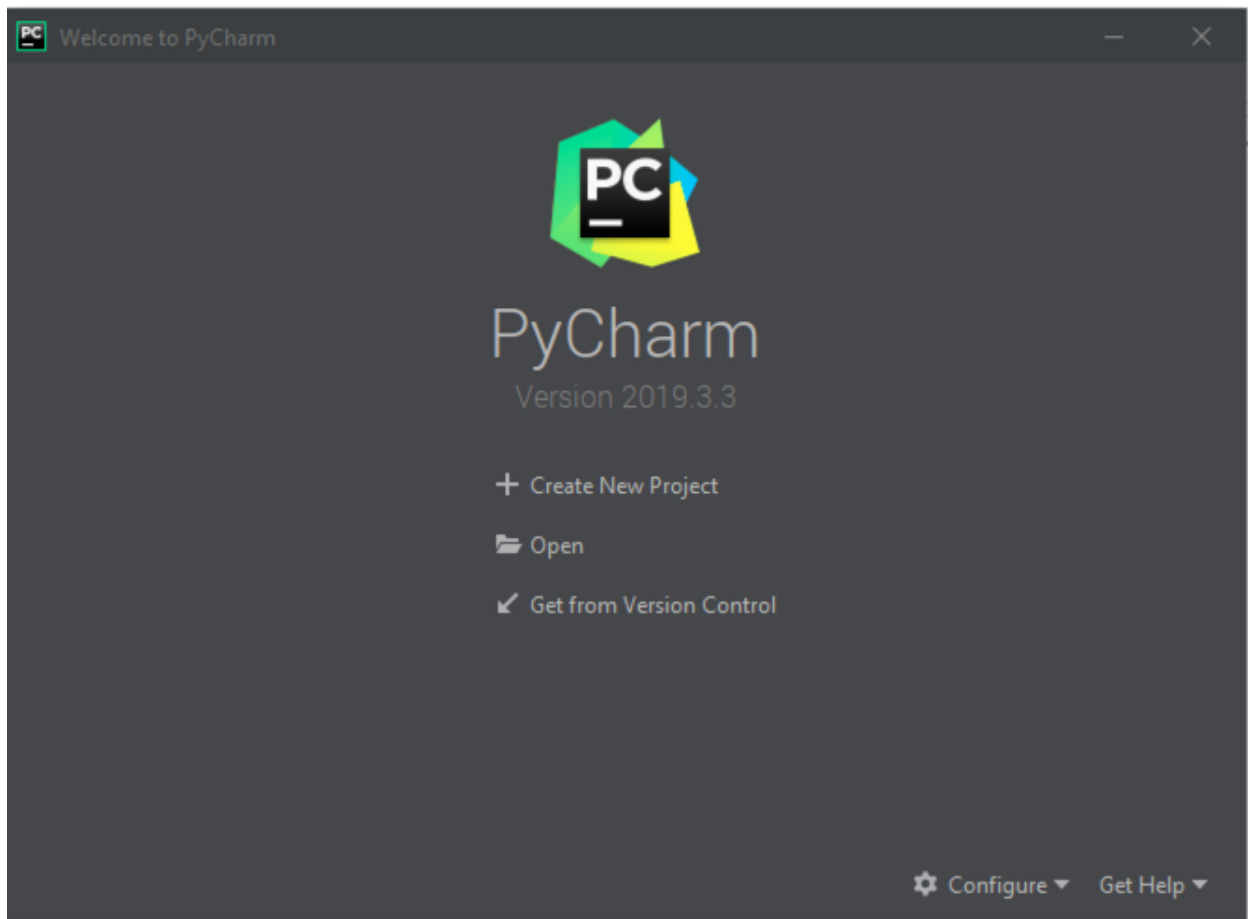


Fig. 2-16: PyCharm Welcome

PyCharm prefers that you work in a project rather than opening a simple file. Projects are typically collections of related files or scripts. You can set up a new project here or open a pre-existing one.

Once you have gone through that process, your screen should look like this:

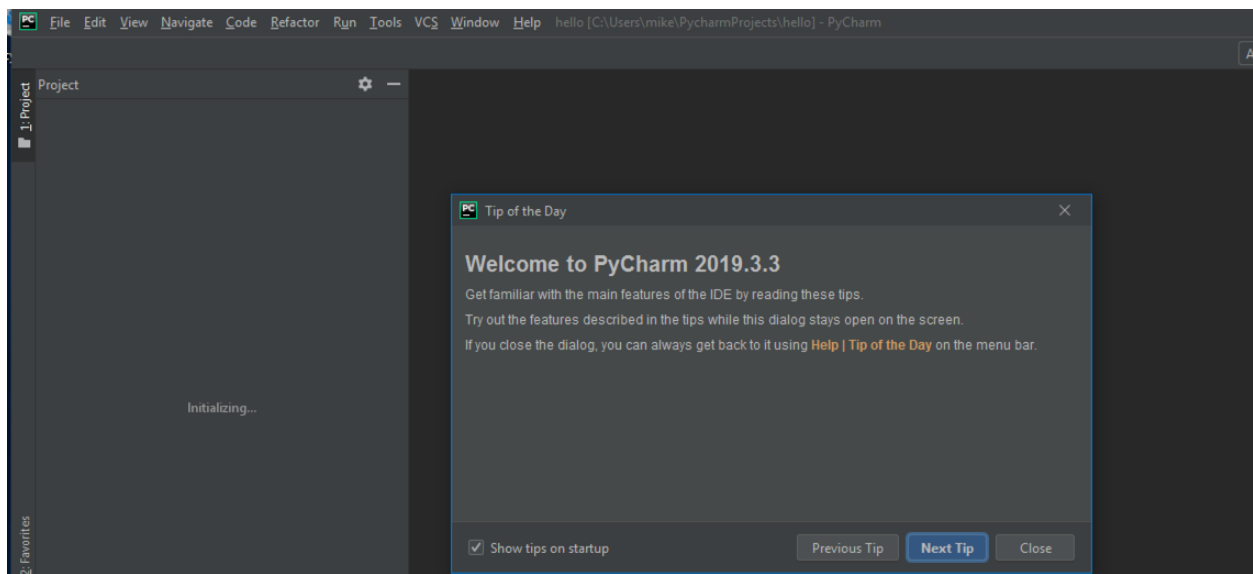


Fig. 2-17: PyCharm Project

Creating a Python Script

To create a new Python script in PyCharm, you can go to **File** and choose **New**. Then pick **Python File** from the choices presented:

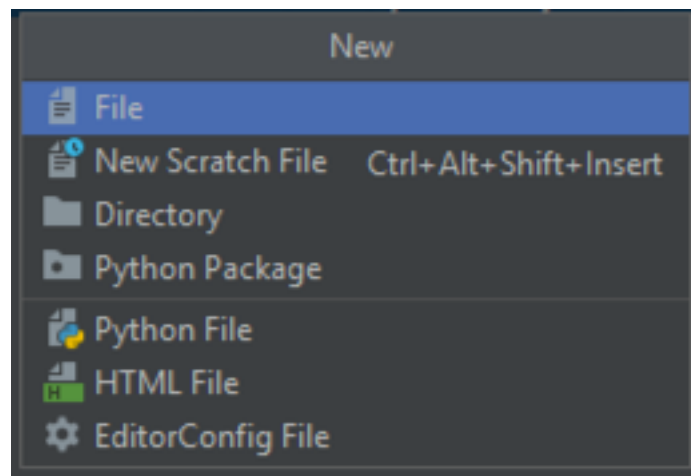


Fig. 2-18: PyCharm New

Give the file a name, such as **hello.py**. Now PyCharm should look like this:

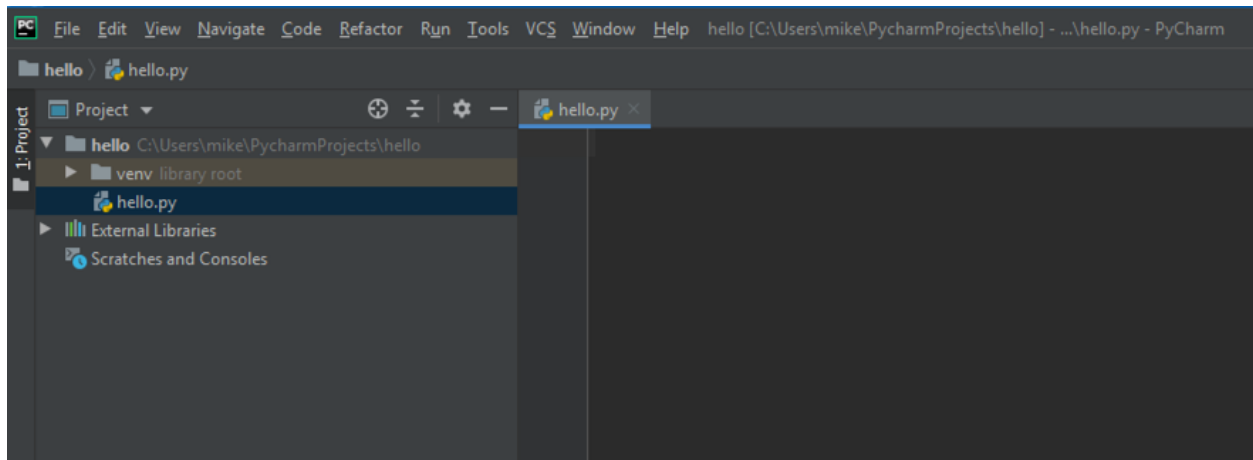


Fig. 2-19: PyCharm Hello World

Running Code in PyCharm

Let's add some code to your file:

```
1 print('Hello PyCharm')
```

To run your code, go to the **Run** menu and choose **Run**. PyCharm might ask you to set up a debug configuration before running it. You can save the defaults and continue.

You should now see the following at the bottom of PyCharm:

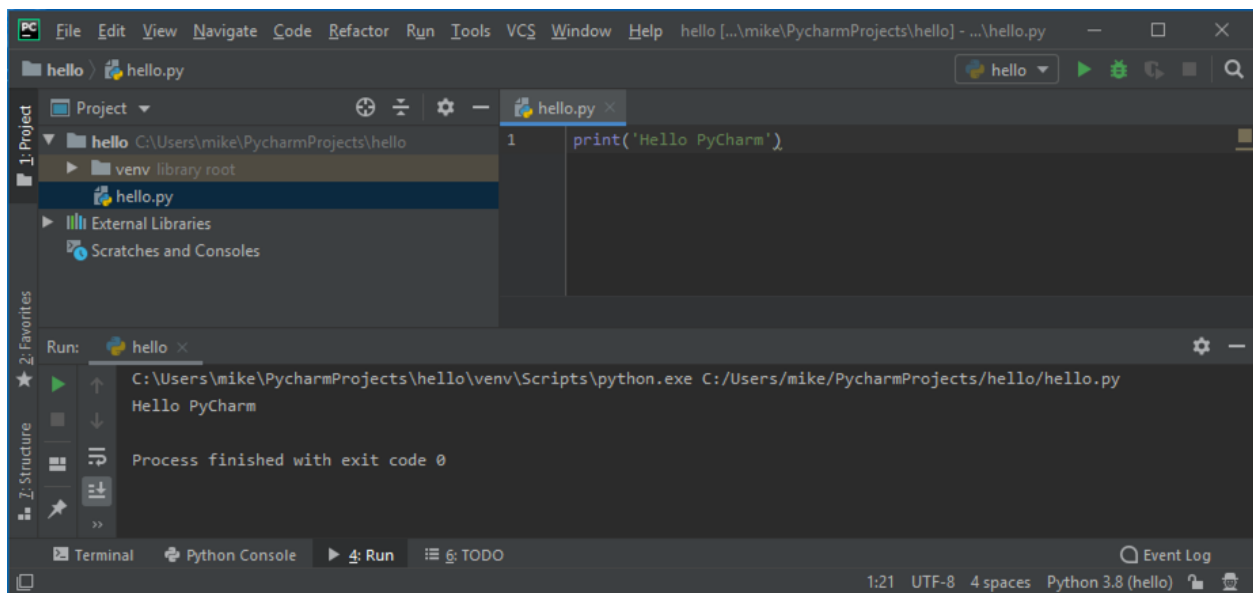


Fig. 2-20: PyCharm Running Code Output

PyCharm Features

PyCharm has tons of features. In fact, it has so many that you could write an entire book on them. For the purposes of this book, you should know that PyCharm will give you suggestions about your code based on PEP8, which is Python's code style guide. You will learn more about that in the next chapter. It will also highlight many other things about your code.

You can usually hover over any code that looks weird to you and a tooltip will appear that will explain the issue or warning.

The debugger that ships with PyCharm is useful for figuring out why your code doesn't work. You can use it to walk through your code line-by-line.

PyCharm's documentation is quite good, so if you get stuck, check their documentation.

Getting Started with Wing Personal

Wingware's Python IDE is written in Python and PyQt. It is my personal favorite IDE for Python. You can get it in Professional (paid), Personal (free) or 101 (really stripped-down version, but also free). Their website explains the differences between the 3 versions.

You can get Wingware here:

<https://wingware.com/>

After you have downloaded and installed the software, go ahead and run it. You will need to accept the License Agreement to load up the IDE.

Once it is fully loaded, you will see something like this:

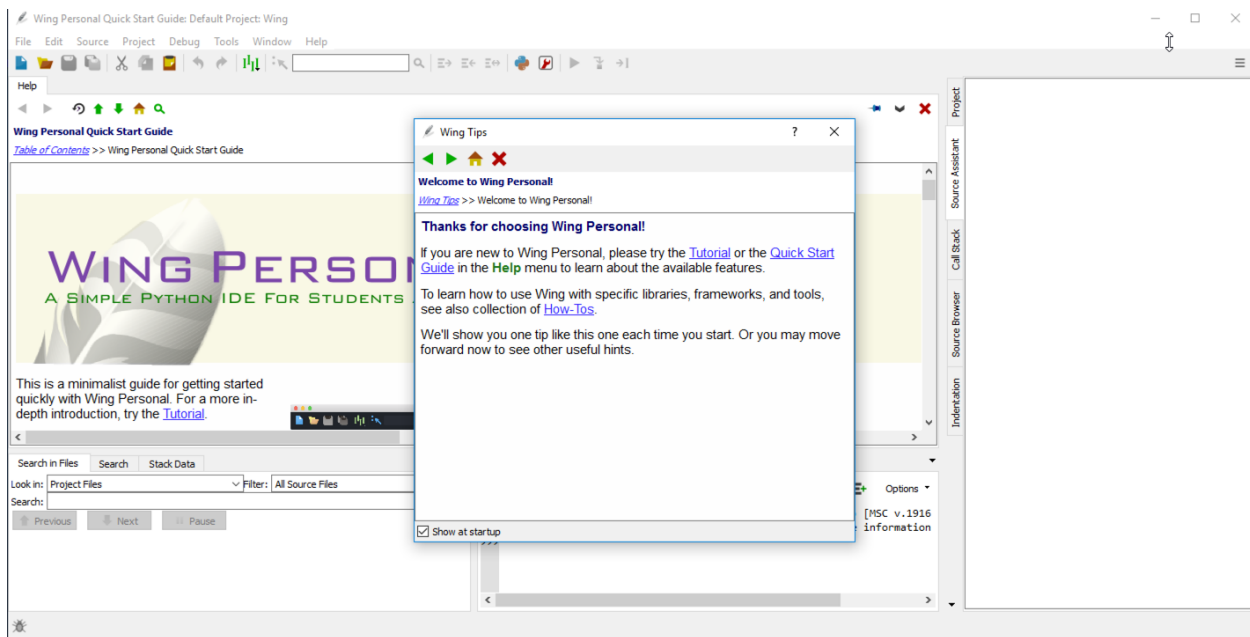


Fig. 2-21: Wingware Python IDE Main Screen

Running Code in Wingware

Let's create some code in Wing. You can open a new file by going to the **File** menu and choosing **New**:

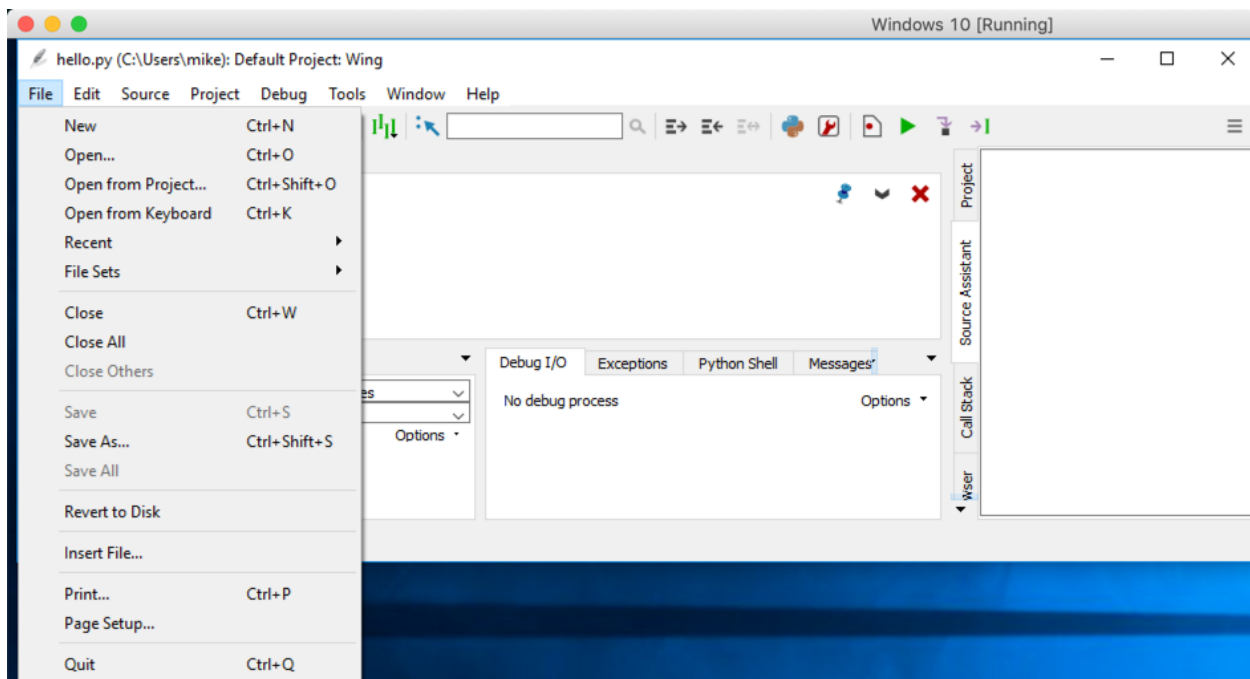


Fig. 2-22: Wingware Python IDE - Adding Code

Now enter the following code:

```
1 print('Hello Wingware')
```

Save the code to disk by going to **File** and then **Save**.

To run this code, you can go to the **Debug** menu, press F5 or click the green “play” button in the toolbar. You will see a debug message dialog:

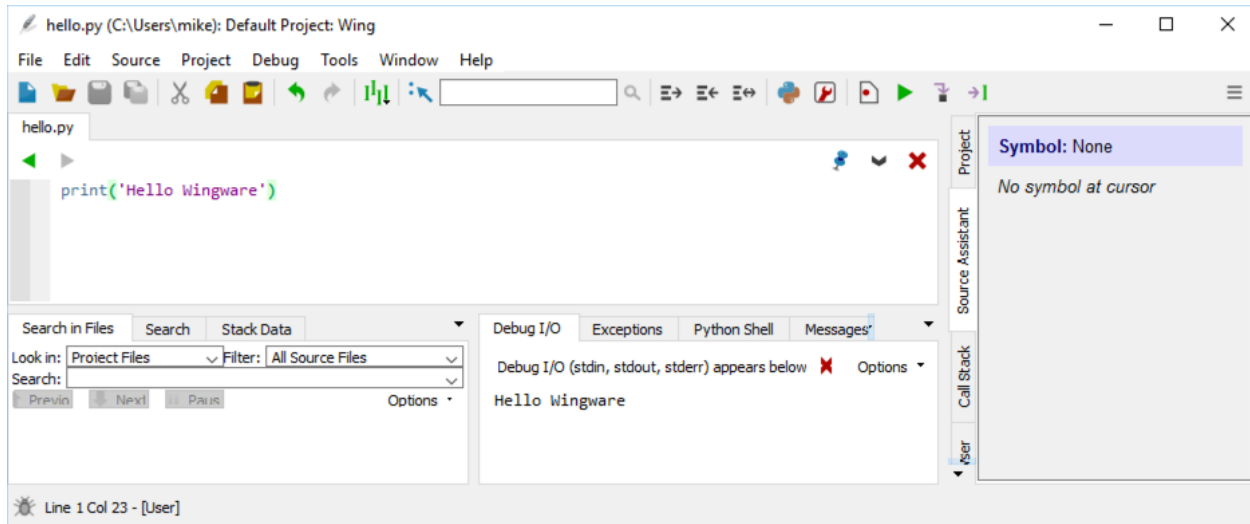


Fig. 2-23: Wingware Python IDE - Code Output

Hit OK and the code will run. You will see the output in the **Debug I/O** tab if there is any.

Note that Wing does not require you to create a project to run a single Python file. You can create projects if you want to though.

Wing Features

Wing has an incredible debugger. However, you cannot use it to its full extent in the free versions of the software. But there is a **Source Assistant** tab in the Personal edition that is very useful. It will show you information about the functions / modules that you have loaded as you use them. This makes learning new modules much easier.

Wing will also show you various issues with your code while you type, although PyCharm seems to do more in this area than Wing does.

Both products have plugins and you can write your own for both IDEs as well.

Getting Started with Visual Studio Code

Visual Studio Code, or VS Code for short, is a general-purpose programming editor. Unlike PyCharm and WingIDE, it is designed to work with lots of languages. PyCharm and WingIDE will let you

write in other languages too, but their primary focus is on Python.

VS Code is made by Microsoft and it is free. You can download it here:

<https://code.visualstudio.com/>

Once you have it downloaded and installed, you will need to install support for Python from the VS Code marketplace.

If you open up VS Code, the screen will look something like this:

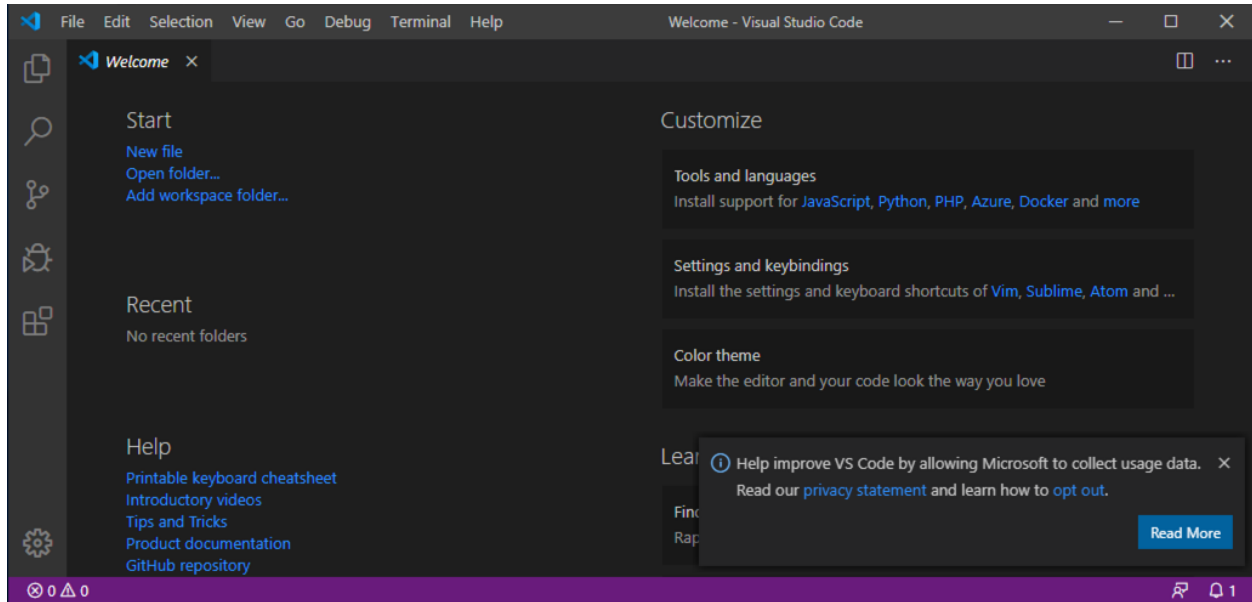


Fig. 2-24: VS Code - Main Screen

Under Customize you can see there is an option for installing Python. If that isn't there, you can click on the **Extensions** button that is on the left and search for Python there:

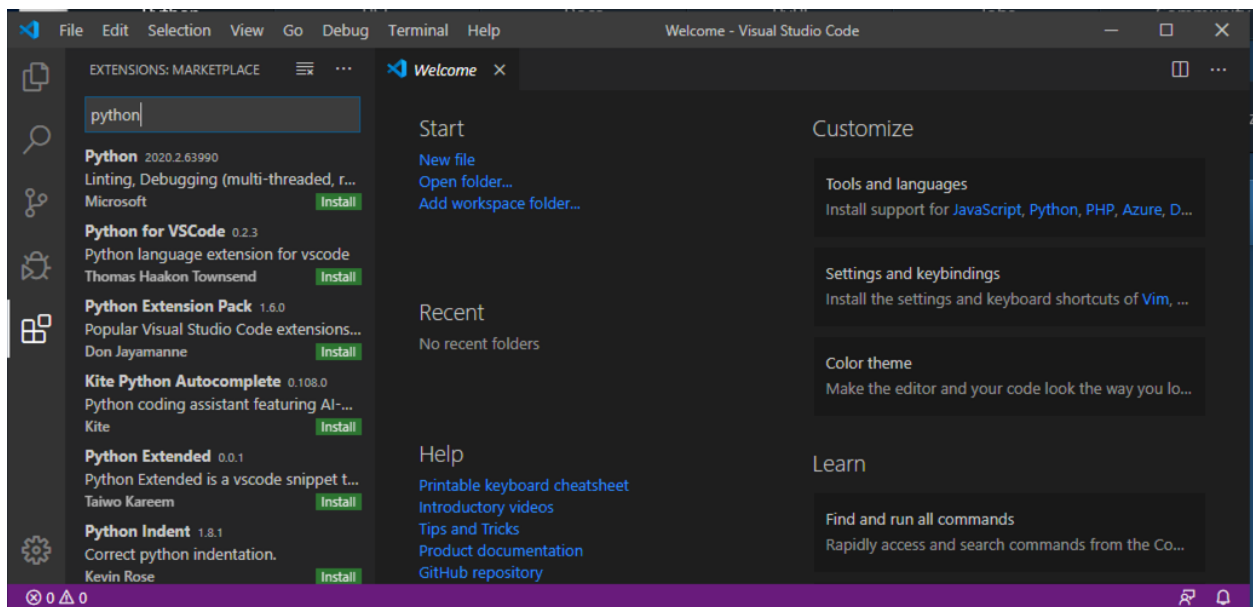


Fig. 2-25: VS Code - Adding the Python Extension

Go ahead and install the Python extension so that VS Code will recognize Python correctly.

Running Code in VS Code

Open a folder in the **File Explorer** tab and then you can right-click in there to create a new file. Alternatively, you can go to the **File** menu and choose **New File** and do it that way.

Once that is done, you can enter the following code and save it:

```
1 print('Hello VS Code')
```

Then right-click anywhere in the editor and select the **Run Python File in Terminal** selection. This will cause your code to run and you will see the following:

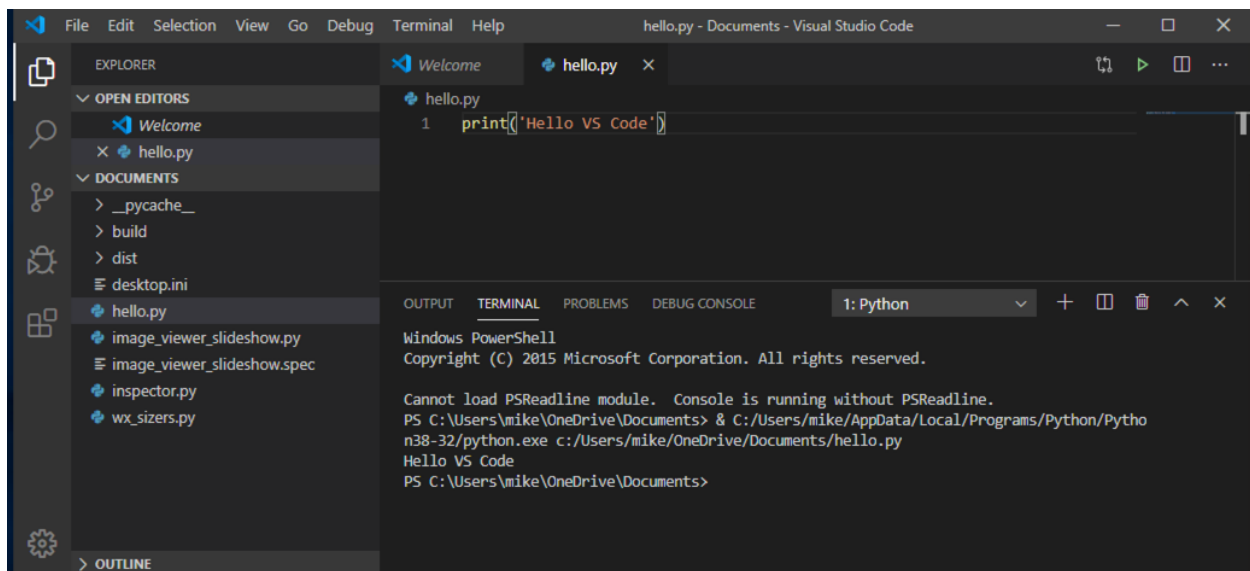


Fig. 2-26: VS Code - Running Code

Note: I didn't have the **PSReadline** module installed when I ran this code which is why you see the error in the console above.

VS Code Features

VS Code can run all kinds of different languages. However, for the purposes of Python, Microsoft has a team of Python developers that are constantly improving this IDE and adding new features. There are tons of extensions that you can install to enhance the editor's functionality.

One of the coolest extensions that you can install is **Live Share**, which lets you do real-time collaboration between developers. It basically shares your coding session with others. Since this IDE is the newest of the bunch and its feature set is changing a lot, you will need to research it on your own time.

Wrapping Up

There are lots of Python code editors to choose from. IDLE is nice in that it comes with Python and is written in Python, so you can actually learn a lot just by looking at its source code. PyCharm and VS Code are very popular right now. Wing IDE used to be more popular than it is today, but I think it is still really great. All of these tools are good, but you should give them a try to see which one works the best for you.

Chapter 3 - Documenting Your Code

Documenting your code early on is quite a bit more important than most new developers realize. Documentation in software development refers to the idea of giving your variables, functions and other identifiers descriptive names. It also refers to adding good comments. When you are immersed in developing your latest creation, it is easy to create variables and functions with non-descriptive names. A month or a year later, when you inevitably come back to your code, you will spend an inordinate amount of time trying to figure out what your code does.

By making your code self-documenting (i.e. using descriptive names) and adding comments when necessary, you will make your code more readable for yourself and for anyone else who may use your code. This will make updating your code and refactoring your code easier too!

In this chapter you will learn about the following topics:

- Comments
- Docstrings
- PEP8 - The Python Style Guide
- Other Tools Useful for Documenting Your Code

Let's get started by learning about comments.

What are Comments?

Comments are code that is for you, not for your computer. What I mean by that is that a comment is basically a note to yourself that explains what is happening in that portion of your code. You use comments to explain why you did something or how a piece of code works. When you are starting out as a new developer, it is good to leave yourself lots of comments to refer back to. But once you learn how to properly name your functions and variables, you will find that you don't need comments as much.

However, comments are still recommended, especially for code that is complex and not easy to understand at first glance. Depending on the company you work for, you may also use comments to document bug fixes. For example, if you are fixing a bug, you might include a comment that mentions which bug you are fixing to help explain why you had to change it.

You can create comments by using the # sign followed by some descriptive text.

Here is an example


```
1  # This is a bad comment
2  x = 10
```

In the code above, the first line demonstrates how to create a simple comment. When Python goes to execute this code, it will see the # symbol and ignore all the text that follows it. In effect, Python will skip that line and try to execute the second line.

This comment is marked as a “bad comment”. While it is good for demonstration purposes, it does not describe the code that follows it at all. That is why it is not a good comment. Good comments describe the code that follows. A good comment may describe the purpose for the Python script, the code line or something else. Comments are your code’s documentation. If they don’t provide information, then they should be removed.

You can also create in-line comments:

```
1  x = 10  # 10 is being assigned to x
```

Here you once again assign 10 to the variable x, but then you add two spaces and the # symbol, which allows you to add a comment about the code. This is useful for when you might need to explain a specific line of code. If you named your variable something descriptive, then you most likely won’t need a comment at all.

Commenting Out

You will hear the term “commenting out code” fairly often. This is the practice of adding the # symbol to the beginning of your code. This will effectively disable your code.

For example, you might have this line of code:

```
1  number_of_people = 10
```

If you want to comment it out, you can do the following:

```
1  # number_of_people = 10
```

You comment code out when you are trying out different solutions or when you’re debugging your code, but you don’t want to delete the code. Python will ignore code that is commented out, allowing you to try something else. Most Python code editors (and text editors) provide a way to highlight multiple lines of code and comment out or uncomment out the entire block of code.

Multiline Comments

Some programming languages, such as C++, provide the ability to create multi-line comments. The Python style guide (PEP8) says that the pound sign is preferred. However, you can use triple quoted strings as a multiline comment.

Here's an example:

```
1 >>> '''This is a
2 multiline comment'''
3 >>> """This is also a
4 multiline comment"""
```

When you create triple quoted strings you may be creating a **docstring**.

Let's find out what docstrings are and how you can use them!

Learning About docstrings

Python has the concept of the PEP, or Python Enhancement Proposal. These PEPs are suggestions or new features for the Python language that get discussed and agreed upon by the Python Steering Council.

PEP 257 (<https://www.python.org/dev/peps/pep-0257/>) describes docstring conventions. You can go read that if you'd like the full story. Suffice to say, a docstring is a string literal that should occur as the first statement in a module, function, class or method definition. You don't need to understand all these terms right now. In fact, you'll learn more about them later on in this book.

A docstring is created by using triple double-quotes.

Here is an example:

```
1 """
2 This is a docstring
3 with multiple lines
4 """
```

Docstrings are ignored by Python. They cannot be executed. However, when you use a docstring as the first statement of a module, function, etc, the docstring will become a special attribute that can be accessed via `__doc__`. You will learn more about attributes and docstrings in the chapter about classes.

Docstrings may be used for one-liners or for multi-line strings.

Here is an example of a one-liner:

```
1 """This is a one-liner"""
```

A one-liner docstring is simply a docstring with only one line of text.

Here is an example of a docstring used in a function:

```
1 def my_function():  
2     """This is the function's docstring"""  
3     pass
```

The code above shows how you can add a docstring to a function. You can learn more about functions in chapter 14. A good docstring describes what the function is supposed to accomplish.

Note: While triple double-quotes are the recommended standard, triple single-quotes, single double-quotes, and single single-quotes all work as well (but single double- and single single-quotes can only contain one line, not multiple lines).

Now let's learn about coding according to Python's style guide.

Python's Style Guide: PEP8

A style guide is a document that describes good programming practices, usually with regard to a single language. Some companies have specific style guides for the company that developers must follow no matter what programming language they are using.

Back in 2001, the Python style guide was created as PEP8 (<https://www.python.org/dev/peps/pep-0008/>). It documents coding conventions for the Python programming language and has been updated several times over the years.

If you plan to use Python a lot, you should really check out the guide. It will help you write better Python code.

Also if you want to contribute to the Python language itself, all your code must conform to the style guidelines or your code will be rejected.

Following a style guide will make your code easier to read and understand. This will help you and anyone else who uses your code in the future.

Remembering all the rules can be hard, though. Fortunately, some intrepid developers have created some utilities that can help!

Tools that can help

There are lots of neat tools that you can use to help you write great code. Here are just a few:

- pycodestyle - <https://pypi.org/project/pycodestyle/> - Checks if your code follows PEP8
- Pylint - <https://www.pylint.org/> - An in-depth static code testing tool that finds common issues with code
- PyFlakes - <https://pypi.org/project/pyflakes/> - Another static code testing tool for Python
- flake8 - <https://pypi.org/project/flake8/> - A wrapper around PyFlakes, pycodestyle and a McCabe script
- Black - <https://black.readthedocs.io/en/stable/> - A code formatter that mostly follows PEP8

You can run these tools against your code to help you find issues with your code. I have found Pylint and PyFlakes / flake8 to be the most useful. Black is helpful if you are working in a team and you want everyone's code to follow the same format. Black can be added to your toolchain to format your code for you.

The more advanced Python IDEs provide some of the checks that Pylint, etc. provide in real-time. For example, PyCharm will automatically check for a lot of the issues that these tools will find. WingIDE and VS Code provide some static code checking as well. You should check out the various IDEs and see which one works the best for you.

Wrapping Up

Python comes with several different ways to document your code. You can use **comments** to explain one or more lines of code. These should be used in moderation and where appropriate. You can also use **docstrings** to document your modules, functions, methods, and classes.

You should also check out Python's style guide that can be found in PEP8. This will help you develop good Python coding practices. There are several other style guides for Python. For example, you might want to look up Google's style guide or possibly NumPy's Python style guide. Sometimes looking at different style guides will help you develop good practices as well.

Finally, you learned about several tools you can use to help you make your code better. If you have the time, I encourage you to check out PyFlakes or Flake8 especially as they can be quite helpful in pointing out common coding issues in your code.

Review Questions

1. How do you create a comment?
2. What do you use a **docstring** for?
3. What is Python's style guide?
4. Why is documenting your code important?

Chapter 4 - Working with Strings

You will be using strings very often when you program. A string is a series of letters surrounded by single, double or triple quotes. Python 3 defines string as a “Text Sequence Type”. You can cast other types to a string using the built-in `str()` function.

In this chapter you will learn how to:

- Creating strings
- String methods
- String formatting
- String concatenation
- String slicing

Let's get started by learning the different ways to create strings!

Creating Strings

Here are some examples of creating strings:

```
1 name = 'Mike'
2 first_name = 'Mike'
3 last_name = "Driscoll"
4 triple = """multi-line
5 string"""
```

When you use triple quotes, you may use three double quotes at the beginning and end of the string or three single quotes. Also, note that using triple quotes allows you to create multi-line strings. Any whitespace within the string will also be included.

Here is an example of converting an integer to a string:

```
1 >>> number = 5
2 >>> str(number)
3 '5'
```

In Python, backslashes can be used to create escape sequences. Here are a couple of examples:

- `\b` - backspace

- `\n` - line feed
- `\r` - ASCII carriage return
- `\t` - tab

There are several others that you can learn about if you read Python's documentation.

You can also use backslashes to escape quotes:

```
1 >>> 'This string has a single quote, \' in the middle'
2 "This string has a single quote, ', in the middle"
```

If you did not have the backslash in the code above, you would receive a `SyntaxError`:

```
1 >>> 'This string has a single quote, ', in the middle'
2 Traceback (most recent call last):
3   Python Shell, prompt 59, line 1
4 invalid syntax: <string>, line 1, pos 38
```

This occurs because the string ends at that second single quote. It is usually better to mix double and single quotes to get around this issue:

```
1 >>> "This string has a single quote, ', in the middle"
2 "This string has a single quote, ', in the middle"
```

In this case, you create the string using double quotes and put a single quote inside of it. This is especially helpful when working with contractions, such as “don’t”, “can’t”, etc.

Now let's move along and see what methods you can use with strings!

String Methods

In Python, everything is an object. You will learn how useful this can be in chapter 18 when you learn about introspection. For now, just know that strings have methods (or functions) that you can call on them.

Here are three examples:

```

1 >>> name = 'mike'
2 >>> name.capitalize()
3 'Mike'
4 >>> name.upper()
5 'MIKE'
6 >>> 'MIke'.lower()
7 'mike'

```

The method names give you a clue as to what they do. For example, `.capitalize()` will change the first letter in the string to a capital letter.

To get a full listing of the methods and attributes that you can access, you can use Python’s built-in `dir()` function:

```

1 >>> dir(name)
2 ['__add__', '__class__', '__contains__', '__delattr__', '__dir__', '__doc__',
3  '__eq__', '__format__', '__ge__', '__getattribute__', '__getitem__',
4  '__getnewargs__', '__gt__', '__hash__', '__init__', '__init_subclass__',
5  '__iter__', '__le__', '__len__', '__lt__', '__mod__', '__mul__', '__ne__',
6  '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__rmod__', '__rmul__',
7  '__setattr__', '__sizeof__', '__str__', '__subclasshook__', 'capitalize',
8  'casefold', 'center', 'count', 'encode', 'endswith', 'expandtabs', 'find',
9  'format', 'format_map', 'index', 'isalnum', 'isalpha', 'isascii', 'isdecimal',
10 'isdigit', 'isidentifier', 'islower', 'isnumeric', 'isprintable', 'isspace',
11 'istitle', 'isupper', 'join', 'ljust', 'lower', 'lstrip', 'maketrans',
12 'partition', 'replace', 'rfind', 'rindex', 'rjust', 'rpartition', 'rsplit',
13 'rstrip', 'split', 'splitlines', 'startswith', 'strip', 'swapcase', 'title',
14 'translate', 'upper', 'zfill']

```

The first third of the listing are special methods that are sometimes called “dunder methods” (AKA double-underscore methods) or “magic methods”. You can ignore these for now as they are used more for intermediate and advanced use-cases. The items in the list above that don’t have double-underscores at the beginning are the ones that you will probably use the most.

You will find that the `.strip()` and `.split()` methods are especially useful when parsing or manipulating text.

You can use `.strip()` and its variants, `.rstrip()` and `.lstrip()` to strip off white space from the string, including tab and new line characters. This is especially useful when you are reading in a text file that you need to parse.

In fact, you will often end up stripping end-of-line characters from strings and then using `.split()` on the result to parse out sub-strings.

Let’s do a little exercise where you will learn how to parse out the 2nd word in a string.

To start, here’s a string:

```
1 >>> my_string = 'This is a string of words'
2 'This is a string of words'
```

Now to get the parts of a string, you can call `.split()`, like this:

```
1 >>> my_string.split()
2 ['This', 'is', 'a', 'string', 'of', 'words']
```

The result is a list of strings. Now normally you would assign this result to a variable, but for demonstration purposes, you can skip that part.

Instead, since you now know that the result is a string, you can use list indexing to get the second element:

```
1 >>> 'This is a string of words'.split()[1]
2 'is'
```

Remember, in Python, lists elements start at 0 (zero), so when you tell it you want element 1 (one), that is the second element in the list.

When doing string parsing for work, I personally have found that you can use the `.strip()` and `.split()` methods pretty effectively to get almost any data that you need. Occasionally you will find that you might also need to use Regular Expressions (regex), but most of the time these two methods are enough.

String Formatting

String formatting or string substitution is where you have a string that you would like to insert into another string. This is especially useful when you need to create a template, such as a form letter. But you will use string substitution a lot for debugging output, printing to standard out and much more.

Standard out (or stdout) is a term used for printing to the terminal. When you run your program from the terminal and you see output from your program, that is because your program “prints” to standard out or standard error (stderr).

Python has three different ways to accomplish string formatting:

- Using the % Method
- Using `.format()`
- Using formatted string literals (f-strings)

This book will focus on f-strings the most and also use `.format()` from time-to-time. But it is good to understand how all three work.

Let’s take a few moments to learn more about string formatting.

Formatting Strings Using %s (printf-style)

Using the % method is Python's oldest method of string formatting. It is sometimes referred to as "printf-style string formatting". If you have used C or C++ in the past, then you may already be familiar with this type of string substitution. For brevity, you will learn the basics of using % here.

Note: This type of formatting can be quirky to work with and has been known to lead to common errors such as failing to display Python tuples and dictionaries correctly. Using either of the other two methods is preferred in that case.

The most common use of using the % sign is when you would use %s, which means convert any Python object to a string using `str()`.

Here is an example:

```
1 >>> name = 'Mike'
2 >>> print('My name is %s' % name)
3 My name is Mike
```

In this code, you take the variable `name` and insert it into another string using the special %s syntax. To make it work, you need to use % outside of the string followed by the string or variable that you want to insert.

Here is a second example that shows that you can pass in an `int` into a string and have it automatically converted for you:

```
1 >>> age = 18
2 >>> print('You must be at least %s to continue' % age)
3 You must be at least 18 to continue
```

This sort of thing is especially useful when you need to convert an object but don't know what type it is.

You can also do string formatting with multiple variables. In fact, there are two ways to do this.

Here's the first one:

```
1 >>> name = 'Mike'
2 >>> age = 18
3 >>> print('Hello %s. You must be at least %i to continue!' % (name, age))
4 Hello Mike. You must be at least 18 to continue!
```

In this example, you create two variables and use %s and %i. The %i indicates that you are going to pass an integer. To pass in multiple items, you use the percent sign followed by a tuple of the items to insert.

You can make this clearer by using names, like this:

```
1 >>> print('Hello %(first_name)s. You must be at least %(age)i to continue!')
2         % {'first_name': name, 'age': age})
3 Hello Mike. You must be at least 18 to continue!
```

When the argument on the right side of the % sign is a dictionary (or another mapping type), then the parenthesized formats in the string must refer to the keys in the dictionary. In other words, if you see %(name)s, then the dictionary to the right of the % must have a name key.

If you do not include all the keys that are required, you will receive an error:

```
1 >>> print('Hello %(first_name)s. You must be at least %(age)i to continue!')
2         % {'age': age})
3 Traceback (most recent call last):
4   Python Shell, prompt 23, line 1
5   KeyError: 'first_name'
```

For more information about using the printf-style string formatting, you should see the following link:

<https://docs.python.org/3/library/stdtypes.html#printf-style-string-formatting>

Now let's move on to using the .format() method.

Formatting Strings Using .format()

Python strings have supported the .format() method for a long time. While this book will focus on using f-strings, you will find that .format() is still quite popular.

For full details on how formatting works, see the following:

<https://docs.python.org/3/library/string.html#formatstrings>

Let's take a look at a few short examples to see how .format() works:

```
1 >>> age = 18
2 >>> name = 'Mike'
3 >>> print('Hello {}. You must be at least {} to continue!'.format(
4     name, age))
5 Hello Mike. You must be at least 18 to continue!
```

This example uses positional arguments. Python looks for two instances of {} and will insert the variables accordingly. If you do not pass in enough arguments, you will receive an error like this:

```
1 >>> print('Hello {}. You must be at least {} to continue!'.format(
2     age))
3 Traceback (most recent call last):
4     Python Shell, prompt 33, line 1
5 IndexError: tuple index out of range
```

This error indicates that you do not have enough items inside the `.format()` call.

You can also use named arguments in a similar way to the previous section:

```
1 >>> age = 18
2 >>> name = 'Mike'
3 >>> print('Hello {first_name}. You must be at least {age} to continue!'.format(
4     first_name=name, age=age))
5 Hello Mike. You must be at least 18 to continue!
```

Instead of passing a dictionary to `.format()`, you can pass in the parameters by name. In fact, if you do try to pass in a dictionary, you will receive an error:

```
1 >>> print('Hello {first_name}. You must be at least {age} to continue!'.format(
2     {'first_name': name, 'age': age}))
3 Traceback (most recent call last):
4     Python Shell, prompt 34, line 1
5 KeyError: 'first_name'
```

There is a workaround for this though:

```
1 >>> print('Hello {first_name}. You must be at least {age} to continue!'.format(
2     **{'first_name': name, 'age': age}))
3 Hello Mike. You must be at least 18 to continue!
```

This looks a bit weird, but in Python when you see a double asterisk (`**`) used like this, it means that you are passing named parameters to the function. So Python is converting the dictionary to `first_name=name, age=age` for you.

You can also repeat a variable multiple times in the string when using `.format()`:

```
1 >>> first_name = 'Mike'
2 >>> print('Hello {first_name}. Why do they call you {first_name}?'.format(
3     first_name=first_name))
4 Hello Mike. Why do they call you Mike?
```

Here you refer to `{first_name}` twice in the string and Python replaces both of them with the `first_name` variable.

If you want, you can also interpolate values using numbers:

```

1 >>> print('Hello {1}. You must be at least {0} to continue!'.format(
2     first_name, age))
3 Hello 18. You must be at least Mike to continue!

```

Because most things in Python start at 0 (zero), in this example you ended up passing the age to {1} and the name to {0}.

A common coding style when working with `.format()` is to create a formatted string and save it to a variable to be used later:

```

1 >>> age = 18
2 >>> first_name = 'Mike'
3 >>> greetings = 'Hello {first_name}. You must be at least {age} to continue!'
4 >>> greetings.format(first_name=first_name, age=age)
5 'Hello Mike. You must be at least 18 to continue!'

```

This allows you to reuse greetings and pass in updated values for `first_name` and `age` later on in your program.

You can also specify the string width and alignment:

```

1 >>> '{:<20}'.format('left aligned')
2 'left aligned'
3 >>> '{:>20}'.format('right aligned')
4 'right aligned'
5 >>> '{:^20}'.format('centered')
6 'centered'

```

Left aligned is the default. The colon (:) tells Python that you are going to apply some kind of formatting. In the first example, you are specifying that the string be left aligned and 20 characters wide. The second example is also 20 characters wide, but it is right aligned. Finally the ^ tells Python to center the string within the 20 characters.

If you want to pass in a variable like in the previous examples, here is how you would do that:

```

1 >>> '{example:^20}'.format(example='centered')
2 'centered'

```

Note that the `example` must come before the `:` inside of the `{}`.

At this point, you should be pretty familiar with the way `.format()` works.

Let's go ahead and move along to f-strings!

Formatting Strings with f-strings

Formatted string literals or f-strings are strings that have an “f” at the beginning and curly braces inside of them that contain expressions, much like the ones you saw in the previous section. These expressions tell the f-string about any special processing that needs to be done to the inserted string, such as justification, float precision, etc.

The f-string was added in Python 3.6. You can read more about it and how it works by checking out PEP 498 here:

<https://www.python.org/dev/peps/pep-0498/>

Let’s go ahead and look at a simple example:

```
1 >>> name = 'Mike'
2 >>> age = 20
3 >>> f'Hello {name}. You are {age} years old'
4 'Hello Mike. You are 20 years old'
```

Here you create the f-string by putting an “f” right before the single, double or triple quote that begins your string. Then inside of the string, you use the curly braces, {}, to insert variables into your string.

However, your curly braces must enclose something. If you create an f-string with empty braces, you will get an error:

```
1 >>> f'Hello {}. You are {} years old'
2 SyntaxError: f-string: empty expression not allowed
```

The f-string can do things that neither %s nor .format() can do, though. Because of the fact that f-strings are evaluated at runtime, you can put any valid Python expression inside of them.

For example, you could increase the displayed value of the age variable:

```
1 >>> age = 20
2 >>> f'{age+2}'
3 '22'
```

Or call a method or function:

```
1 >>> name = 'Mike'
2 >>> f'{name.lower()}'
3 'mike'
```

You can also access dictionary values directly inside of an f-string:

```
1 >>> sample_dict = {'name': 'Tom', 'age': 40}
2 >>> f'Hello {sample_dict["name"]}. You are {sample_dict["age"]} years old'
3 'Hello Tom. You are 40 years old'
```

However, backslashes are not allowed in f-string expressions:

```
1 >>> print(f'My name is {name\n}')
2 SyntaxError: f-string expression part cannot include a backslash
```

But you can use backslashes outside of the expression in an f-string:

```
1 >>> name = 'Mike'
2 >>> print(f'My name is {name}\n')
3 My name is Mike
```

One other thing that you can't do is add a comment inside of an expression in an f-string:

```
1 >>> f'My name is {name # name of person}'
2 SyntaxError: f-string expression part cannot include '#'
```

In Python 3.8, f-strings added support for `=`, which will expand the text of the expression to include the text of the expression plus the equal sign and then the evaluated expression. That sounds kind of complicated, so let's look at an example:

```
1 >>> username = 'jdoe'
2 >>> f'Your {username=}'
3 "Your username='jdoe'"
```

This example demonstrates that the text inside of the expression, `username=` is added to the output followed by the actual value of `username` in quotes.

f-strings are very powerful and extremely useful. They will simplify your code quite a bit if you use them wisely. You should definitely give them a try.

Let's find out what else you can do with strings!

String Concatenation

Strings also allow concatenation, which is a fancy word for joining two strings into one.

To concatenate strings together, you can use the `+` sign:

```
1 >>> first_string = 'My name is'
2 >>> second_string = 'Mike'
3 >>> first_string + second_string
4 'My name isMike'
```

Oops! It looks like the strings merged in a weird way because you forgot to add a space to the end of the `first_string`. You can change it like this:

```
1 >>> first_string = 'My name is '
2 >>> second_string = 'Mike'
3 >>> first_string + second_string
4 'My name is Mike'
```

Another way to merge strings is to use the `.join()` method. The `.join()` method accepts an iterable, such as a list, of strings and joins them together.

```
1 >>> first_string = 'My name is' # no ending space
2 >>> second_string = 'Mike'
3 >>> ''.join([first_string, second_string])
4 'My name isMike'
```

This will make the strings join right next to each other, just like `+` did. However, you can put something inside of the string that you are using for the join, and it will be inserted between each string in the list:

```
1 >>> ' '.join([first_string, second_string]) # a space is in the join string
2 'My name is Mike'
3 >>> '--'.join([first_string, second_string]) # a dash dash is in the join string
4 'My name is--Mike'
```

More often than not, you can use an f-string rather than concatenation or `.join()` and the code will be easier to follow.

String Slicing

Slicing in strings works in much the same way that it does for Python lists. Let's take the string "Mike". The letter "M" is at position zero and the letter "e" is at position 3.

If you want to grab characters 0-3, you would use this syntax: `my_string[0:4]`

What that means is that you want the substring starting at position zero up to but not including position 4.

Here are a few examples:

```
1 >>> 'this is a string'[0:4]
2 'this'
3 >>> 'this is a string'[:4]
4 'this'
5 >>> 'this is a string'[-4:]
6 'ring'
```

The first example grabs the first four letters from the string and returns them. If you want to, you can drop the zero as that is the default and use `[:4]` instead, which is what example two does.

You can also use negative position values. So `[-4:]` means that you want to start at the end of the string and get the last four letters of the string. You will learn more about slicing in **chapter 6**, which is about the `list` data type.

You should play around with slicing on your own and see what other slices you can come up with.

Wrapping Up

Python strings are powerful and useful. They can be created using single, double, or triple quotes. Strings are objects, so they have methods. You also learned about string concatenation, string slicing, and three different methods of string formatting.

The newest flavor of string formatting is the f-string. It is also the most powerful and the currently preferred method for formatting strings.

Review Questions

1. What are 3 ways to create a string?
2. Run `dir("")`. This lists all the string methods you can use. Which of these methods will capitalize each of the words in a sentence?
3. Change the following example to use f-strings:

```
1 >>> name = 'Mike'
2 >>> age = 21
3 >>> print('Hello %s! You are %i years old.' % (name, age))
4 Hello Mike! You are 21 years old.
```

4. How do you concatenate these two strings together?


```
1 >>> first_string = 'My name is'
2 >>> second_string = 'Mike'
```

5. Use string slicing to get the substring, “is a”, out of the following string:

```
1 >>> 'this is a string'
```

Chapter 5 - Numeric Types

Python is a little different than some languages in that it only has three built-in numeric types. A built-in data type means that you don't have to do anything to use them other than typing out their name.

The built-in numeric types are:

- `int`
- `float`
- `complex`

Python 2 also had the `long` numeric type, which was an integer able to represent values larger than an `int` could. In Python 3, `int` and `long` were combined so that Python 3 only has `int`. You can create an `int` by simply typing the number or by using `int()`. `2`, `3`, `int(4)`, and `int("5")` are all integers.

If you are familiar with C++, you probably know that floating-point numbers are defined using the `double` keyword. In Python, you can create a `float` by typing it or by using `float()`. `3.14`, `5.0`, `float(7.9)`, and `float("8.1")` are all floating point numbers.

A complex number has a real and an imaginary part. The real and imaginary parts are accessed using attribute notation: `.real` and `.imag`, respectively. Complex numbers can be created by either typing them or using `complex()`. `2+1j`, `2-1j`, `5j`, `complex(7+2j)`, `complex("7+2j")`, and `complex(7, 2)` are all complex numbers.

There are two other numeric types that are included with Python in its standard library. They are as follows:

- `decimal` - for holding floating-point numbers that allow the user to define their precision
- `fractions` - rational numbers

You can import these libraries using Python's `import` keyword, which you will learn about in **chapter 16**. You might also be interested in checking out Python's `round()` keyword or its `math` module.

Let's go ahead and learn a little bit more about how you can create and use numeric types in Python!

Integers

You can create an integer in two ways in Python. The most common way is to assign an integer to a variable:

```
1 my_integer = 3
```

The equals sign (=) is Python’s **assignment** operator. It “assigns” the value on the right to the variable name on the left. So in the code above, you are *assigning* the value 3 to the variable `my_integer`.

The other way to create an integer is to use the `int` callable, like this:

```
1 my_integer = int(3)
```

Most of the time, you won’t use `int()` to create an integer. In fact, `int()` is usually used for converting a string or other type to an integer. Another term for this is **casting**.

A little known feature about `int()` is that it takes an optional second argument for the base in which the first argument is to be interpreted. In other words, you can tell Python to convert to `base2`, `base8`, `base16`, etc.

Here’s an example:

```
1 >>> int('10', 2)
2 2
3 >>> int('101', 2)
4 5
```

The first argument has to be a string while the second argument is the base, which in this case is 2. Now let’s move on and learn how to create a `float`!

Floats

A `float` in Python refers to a number that has a decimal point in it. For example, 2.0 is a `float` while 2 is an `int`.

You can create a `float` in Python like this:

```
1 my_float = 2.0
```

This code will assign the number, 2.0, to the variable `my_float`.

You can also create a `float` like this:

```
1 my_float = float(2.0)
```

Python’s `float()` built-in will convert an integer or even a string into a `float` if it can. Here’s an example of converting a string to a `float`:

```
1 my_float = float("2.0")
```

This code converts the string, “2.0”, to a `float`. You can also cast string or floats to `int` using the `int()` built-in from the previous section.

Note: The `float` numeric type is inexact and may differ across platforms. You shouldn’t use the `float` type when dealing with sensitive numeric types, such as money values, due to rounding issues. Instead it is recommended that you use Python’s `decimal` module.

Complex Numbers

A complex number has a *real* and an *imaginary* part, which are each a floating-point number. Let’s look at an example with a complex number object named `comp` to see how you can access each of these parts by using `comp.real` and `comp.imag` to extract the real and imaginary parts, respectively, from the number:

```
1 >>> comp = 1 + 2j
2 >>> type(comp)
3 <class 'complex'>
4 >>> comp.real
5 1.0
6 >>> comp.imag
7 2.0
```

In the code sample above, you created a complex number. To verify that it is a complex number, you can use Python’s built-in `type` function on the variable. Then you extract the `real` and `imag` parts from the complex number.

You can also use the `complex()` built-in callable to create a complex number:

```
1 >>> complex(10, 12)
2 (10+12j)
```

Here you created a complex number in the interpreter, but you don’t assign the result to a variable.

Numeric Operations

All the numeric types, with the exception of `complex`, support a set of numeric operations.

Here is a list of the operations that you can do:

Operation	Result
<code>a + b</code>	The sum of a and b
<code>a - b</code>	The difference of a and b
<code>a * b</code>	The product of a and b
<code>a / b</code>	The quotient of a and b
<code>a // b</code>	The floored quotient of a and b
<code>a % b</code>	The remainder of a / b
<code>-a</code>	a negated (multiply by -1)
<code>+a</code>	a unchanged
<code>abs(a)</code>	absolute value of a
<code>int(a)</code>	a converted to integer
<code>float(x)</code>	a converted to a floating-point number
<code>complex(re, im)</code>	A complex number with real and imaginary
<code>c.conjugate()</code>	The conjugate of the complex number c
<code>divmod(a, b)</code>	The pair: (a // b, a % b)
<code>pow(a, b)</code>	a to the power of b
<code>a ** b</code>	a to the power of b

You should check out the full documentation for additional details about how numeric types work (scroll down to the *Numeric Types* section):

- <https://docs.python.org/3/library/stdtypes.html>

Augmented Assignment

Python supports doing some types of arithmetic using a concept called **Augmented Assignment**. This idea was first proposed in PEP 203:

- <https://www.python.org/dev/peps/pep-0203/>

The syntax allows you to do various arithmetic operations using the following operators:

`+= -= *= /= %= **= <<= >>= &= ^= |=`

This syntax is a shortcut for doing common arithmetic in Python. With it you can replace the following code:

```

1  >>> x = 1
2  >>> x = x + 2
3  >>> x
4  3

```

with this:

```
1 >>> x = 1
2 >>> x += 2
3 >>> x
4 3
```

This code is the equivalent of the previous example.

Wrapping Up

In this chapter, you learned the basics of Python's Numeric types. Here you learned a little about how Python handles `int`, `float`, and `complex` number types. You can use these types for working with most operations that involve numbers. However, if you are working with floating-point numbers that need to be precise, you will want to check out Python's `decimal` module. It is tailor-made for working with that type of number.

Review Questions

1. What 3 numeric types does Python support without importing anything?
2. Which module should you use for money or other precise calculations?
3. Give an example of how to use augmented assignment.

Chapter 6 - Learning About Lists

Lists are a fundamental data type in the Python programming language. A `list` is a mutable sequence that is typically a collection of homogeneous items. Mutable means that you can change a `list` after its creation. You will frequently see lists that contain other lists. These are known as nested lists. You will also see lists that contain all manner of other data types, such as dictionaries, tuples, and other objects.

In this chapter, you will learn the following:

- Creating Lists
- List Methods
- List Slicing
- List Copying

Let's find out how you can create a `list`!

Creating Lists

There are several ways to create a `list`. You may construct a list in any of the following ways:

- Using a pair of square brackets with nothing inside creates an empty list: `[]`
- Using square brackets with comma-separated items: `[1, 2, 3]`
- Using a list comprehension (see Chapter 13 for more information): `[x for x in iterable]`
- Using the `list()` function: `list(iterable)`

An iterable is a collection of items that can return its members one at a time; some iterables have an order (i.e. sequences), and some do not. Lists themselves are sequences. Strings are sequences as well. You can think of strings as a sequence of characters.

Let's look at a few examples of creating a `list` so you can see it in action:

```
1 >>> my_list = [1, 2, 3]
2 >>> my_list
3 [1, 2, 3]
```

This first example is pretty straight-forward. Here you create a list with 3 numbers in it. Then you print it out to verify that it contains what you think it should.

The next way to create a `list` is by using Python's built-in `list()` function:

```
1 >>> list_of_strings = list('abc')
2 >>> list_of_strings
3 ['a', 'b', 'c']
```

In this case, you pass a string of three letters to the `list()` function. It automatically iterates over the characters in the string to create a list of three strings, where each string is a single character.

The last example to look at is how to create empty lists:

```
1 >>> empty_list = []
2 >>> empty_list
3 []
4 >>> another_empty_list = list()
5 >>> another_empty_list
6 []
```

The quickest way to create an empty list is by using the square brackets without putting anything inside them. The second easiest way is to call `list()` without any arguments. The nice thing about using `list()` in general is that you can use it to cast a compatible data type to a list, as you did with the string “abc” in the example earlier.

List Methods

You haven’t learned about methods yet, but it is important to cover list methods now. Don’t worry. You will be learning more about methods throughout this book and by the end you will understand them quite well!

A Python list has several methods that you can call. A method allows you to do something to the list.

Here is a listing of the methods you can use with a list:

- `append()`
- `clear()`
- `copy()`
- `count()`
- `extend()`
- `index()`
- `insert()`
- `pop()`
- `remove()`
- `reverse()`
- `sort()`

Most of these will be covered in the following sections. Let's talk about the ones that aren't covered in a specific section first.

You can use `count()` to count the number of instances of the object that you passed in.

Here is an example:

```
1 >>> my_list = list('abcc')
2 >>> my_list.count('a')
3 1
4 >>> my_list.count('c')
5 2
```

This is a simple way to count the number of occurrences of an item in a list.

The `index()` method is useful for finding the first instance of an item in a list:

```
1 >>> my_list = list('abcc')
2 >>> my_list.index('c')
3 2
4 >>> my_list.index('a')
5 0
```

Python lists are zero-indexed, so "a" is in position 0, "b" is at position 1, etc.

You can use the `reverse()` method to reverse a list *in-place*:

```
1 >>> my_list = list('abcc')
2 >>> my_list.reverse()
3 >>> my_list
4 ['c', 'c', 'b', 'a']
```

Note that the `reverse()` method returns `None`. What that means is that if you try to assign the reversed list to a new variable, you may end up with something unexpected:

```
1 >>> x = my_list.reverse()
2 >>> print(x)
3 None
```

Here you end up with `None` instead of the reversed list. That is what *in-place* means. The original list is reversed, but the `reverse()` method itself doesn't return anything.

Now let's find out what you can do with the other list methods!

Adding to a List

There are three `list` methods that you can use to add to a list. They are as follows:

- `append()`
- `extend()`
- `insert()`

The `append()` method will add an item to the end of a pre-existing list:

```
1 >>> my_list = list('abcc')
2 >>> my_list
3 ['a', 'b', 'c', 'c']
4 >>> my_list.append(1)
5 >>> my_list
6 ['a', 'b', 'c', 'c', 1]
```

First you create a list that is made up of four one-character strings. Then you append an integer to the end of the list. Now the list should have 5 items in it with the 1 on the end.

You can use Python's built-in `len()` function to check the number of items in a list:

```
1 >>> len(my_list)
2 5
```

So this tells you that you do in fact have five items in the list. But what if you wanted to add an element somewhere other than the end of the list?

You can use `insert()` for that:

```
1 >>> my_list.insert(0, 'first')
2 >>> my_list
3 ['first', 'a', 'b', 'c', 'c', 1]
```

The `insert()` method takes two arguments:

- The position at which to insert
- The item to insert

In the code above, you tell Python that you want to insert the string, “first”, into the 0 position, which is the first position in the list.

There are two other ways to add items to a list. You can add an iterable to a list using `extend()`:

```
1 >>> my_list = [1, 2, 3]
2 >>> other_list = [4, 5, 6]
3 >>> my_list.extend(other_list)
4 >>> my_list
5 [1, 2, 3, 4, 5, 6]
```

Here you create two lists. Then you use `my_list`'s `extend()` method to add the items in `other_list` to `my_list`.

The `extend()` method will iterate over the items in the passed in list and add each of them to the list.

You can also combine lists using concatenation:

```
1 >>> my_list = [1, 2, 3]
2 >>> other_list = [4, 5, 6]
3 >>> combined = my_list + other_list
4 >>> combined
5 [1, 2, 3, 4, 5, 6]
```

In this case, you create two lists and then combine them using Python's `+` operator. Note that `my_list` and `other_list` have not changed.

You can also use `+=` with Python lists:

```
1 >>> my_list = [1, 2, 3]
2 >>> other_list = [4, 5, 6]
3 >>> my_list += other_list
4 >>> my_list
5 [1, 2, 3, 4, 5, 6]
```

This is a somewhat simpler way to combine the two lists, but it does change the original list in the same way that using the `extend()` method does.

Now let's learn how to access and change elements within a list.

Accessing and Changing List Elements

Lists are made to be worked with. You will need to learn how to access individual elements as well as how to change them.

Let's start by learning how to access an item:

```
1 >>> my_list = [7, 8, 9]
2 >>> my_list[0]
3 7
4 >>> my_list[2]
5 9
```

To access an item in a list, you need to use square braces and pass in the index of the item that you wish to access. In the example above, you access the first and third elements.

Lists also support accessing items in reverse by using negative values:

```
1 >>> my_list[-1]
2 9
```

This example demonstrates that when you pass in -1, you get the last item in the list returned. Try using some other values and see if you can get the first item using negative indexing.

If you try to use an index that does not exist in the list, you will get an `IndexError`:

```
1 >>> my_list[-5]
2 Traceback (most recent call last):
3   Python Shell, prompt 41, line 1
4 builtins.IndexError: list index out of range
```

Now let's learn about removing items!

Deleting From a List

Deleting items from a list is pretty straight-forward. There are 4 primary methods of removing items from a list:

- `clear()`
- `pop()`
- `remove()`
- `del`

You can use `clear()` to remove everything from the list. Let's see how that works:

```
1 >>> my_list = [7, 8, 9]
2 >>> my_list.clear()
3 >>> my_list
4 []
```

After calling `clear()`, the list is now empty. This can be useful when you have finished working on the items in the list and you need to start over from scratch. Of course, you could also do this instead of `clear()`:

```
1 >> my_list = []
```

This will create a new empty list. If it is important for you to always use the same object, then using `clear()` would be better. If that does not matter, then setting it to an empty list will work well too.

If you would rather remove individual items, then you should check out `pop()` or `remove()`. Let's start with `pop()`:

```
1 >>> my_list = [7, 8, 9]
2 >>> my_list.pop()
3 9
4 >>> my_list
5 [7, 8]
```

You can pass an index to `pop()` to remove the item with that specific index and return it. Or you can call `pop()` without an argument, like in the example above, and it will default to removing the last item in the list and returning it. `pop()` is the most flexible way of removing items from a list.

If the list is empty or you pass in an index that does not exist, `pop()` will throw an exception:

```
1 >>> my_list.pop(10)
2 Traceback (most recent call last):
3   Python Shell, prompt 50, line 1
4 builtins.IndexError: pop index out of range
```

Now let's take a look at how `remove()` works:

```
1 >>> my_list = [7, 8, 9]
2 >>> my_list.remove(8)
3 >>> my_list
4 [7, 9]
```

`remove()` will delete the first instance of the passed in item. So in this case, you tell the list to remove the first occurrence of the number 8.

If you tell `remove()` to delete an item that is not in the list, you will receive an exception:

```
1 >>> my_list.remove(4)
2 Traceback (most recent call last):
3   Python Shell, prompt 51, line 1
4 builtins.ValueError: list.remove(x): x not in list
```

You can also use Python's built-in `del` keyword to delete items from a list:

```
1 >>> my_list = [7, 8, 9]
2 >>> del my_list[1]
3 >>> my_list
4 [7, 9]
```

You will receive an error if you try to remove an index that does not exist:

```
1 >>> my_list = [7, 8, 9]
2 >>> del my_list[6]
3 Traceback (most recent call last):
4   Python Shell, prompt 296, line 1
5 builtins.IndexError: list assignment index out of range
```

Now let's learn about sorting a list!

Sorting a List

Lists in Python can be sorted. You can use the built-in `sort()` method to sort a list in-place or you can use Python's `sorted()` function to return a new sorted list.

Let's create a list and try sorting it:

```
1 >>> my_list = [4, 10, 2, 1, 23, 9]
2 >>> my_list.sort()
3 >>> my_list
4 [1, 2, 4, 9, 10, 23]
```

Here you create a list with 6 integers in a pretty random order. To sort the list, you call its `sort()` method, which will sort it *in-place*. Remember that *in-place* means that `sort()` does not return anything.

A common misconception with Python is that if you call `sort()`, you can assign the now-sorted list to a variable, like this:

```
1 >>> sorted_list = my_list.sort()
2 >>> print(sorted_list)
3 None
```

However, when you do that, you will see that `sort()` doesn't actually return the sorted list. It always returns `None`.

Fortunately you can use Python's built-in `sorted()` method for this:

```
1 >>> my_list = [4, 10, 2, 1, 23, 9]
2 >>> sorted_list = sorted(my_list)
3 >>> sorted_list
4 [1, 2, 4, 9, 10, 23]
```

If you use `sorted()`, it will return a new list, sorted ascending by default.

Both the `sort()` method and the `sorted()` function will also allow you to sort by a specified key and you can tell them to sort ascending or descending by setting its `reversed` flag.

Let's sort this list in descending order instead:

```
1 >>> my_list = [4, 10, 2, 1, 23, 9]
2 >>> sorted_list = sorted(my_list, reverse=True)
3 >>> sorted_list
4 [23, 10, 9, 4, 2, 1]
```

When you have a more complicated data structure, such as a nested list or a dictionary, you can use `sorted()` to sort in special ways, such as by key or by value.

List Slicing

Python lists support the idea of slicing. Slicing a list is done by using square brackets and entering a start and stop value. For example, if you had `my_list[1:3]`, you would be saying that you want to create a new list with the element starting at index 1 through index 3 but not including index 3.

Here is an example:

```
1 >>> my_list = [4, 10, 2, 1, 23, 9]
2 >>> my_list[1:3]
3 [10, 2]
```

This slice returns index 1 (10) and index 2 (2) as a new list.

You can also use negative values to slice:

```
1 >>> my_list = [4, 10, 2, 1, 23, 9]
2 >>> my_list[-2:]
3 [23, 9]
```

In this example, you didn't specify an end value. That means you want to start at the second to last item in the list, 23, and take it to the end of the list.

Let's try another example where you specify only the end index:

```
1 >>> my_list = [4, 10, 2, 1, 23, 9]
2 >>> my_list[:3]
3 [4, 10, 2]
```

In this example, you want to grab all the values starting at index 0 up to but not including index 3.

Copying a List

Occasionally you will want to copy a list. One simple way to copy your list is to use the copy method:

```
1 >>> my_list = [1, 2, 3]
2 >>> new_list = my_list.copy()
3 >>> new_list
4 [1, 2, 3]
```

This successfully creates a new list and assigns it to the variable, `new_list`.

However note that when you do this, you are creating what is known as a “shallow copy”. What that means is that if you were to have mutable objects in your list, they can be changed and it will affect both lists. For example, if you had a dictionary in your list and the dictionary was modified, both lists will change, which may not be what you want. You will learn about dictionaries in **chapter 8**.

```
1 >>> my_list = [1, 2, 3]
2 >>> new_list = my_list.copy()
3 >>> my_list
4 [1, 2, 3]
5 >>> new_list
6 [1, 2, 3]
```

You can also copy a list by using this funny syntax:


```
1 >>> my_list = [1, 2, 3]
2 >>> new_list = my_list[:]
3 >>> new_list
4 [1, 2, 3]
```

This example is telling Python to create a slice from the 0 (first) element to the last, which in effect is the whole list.

Finally, you could also use Python's `list()` function to copy a list:

```
1 >>> my_list = [1, 2, 3]
2 >>> new_list = list(my_list)
3 >>> new_list
4 [1, 2, 3]
```

No matter which method you choose though, whether you duplicate a list by using `[:]`, `copy()` or `list()`, all three will create a shallow copy. To avoid running into weird issues where changing one list affects the copied list, you should use the `deepcopy` method from the `copy` module instead.

Wrapping Up

In this chapter, you learned all about Python's wonderful `list` data type. You will be using lists extensively when you are programming in Python.

You learned the following in this chapter:

- Creating Lists
- List Methods
- List Slicing
- List Copying

Now you are ready to move on and learn about tuples!

Review Questions

1. How do you create a list?
2. Create a list with 3 items and then use `append()` to add two more.
3. What is wrong with this code?

```
1 >>> my_list = [1, 2, 3]
2 >>> my_list.remove(4)
```

4. How do you remove the 2nd item in this list?

```
1 >>> my_list = [1, 2, 3]
```

5. Create a list that looks like this: [4, 10, 2, 1, 23]. Use string slicing to get only the middle 3 items.

Chapter 7 - Learning About Tuples

Tuples are another sequence type in Python. Tuples consist of a number of values that are separated by commas. A tuple is immutable whereas a list is not. Immutable means that the tuple has a fixed value and cannot change. You cannot add or delete items in a tuple. Immutable objects are useful when you need a constant hash value. The most popular example of a hash value in Python is the key to a Python dictionary, which you will learn about in **chapter 8**.

In this chapter, you will learn how to:

- Create tuples
- Work with tuples
- Concatenate tuples
- Special case tuples

Let's find out how to create tuples!

Creating Tuples

You can create tuples in several different ways. Let's take a look:

```
1 >>> a_tuple = 4, 5
2 >>> type(a_tuple)
3 <class 'tuple'>
```

One of the simplest methods of creating a tuple is to have a sequence of values separated by commas. Those values could be integers, lists, dictionaries, or any other object.

Depending on where in the code you are trying to create a tuple, just using commas might be ambiguous; you can always use parentheses to make it explicit:

```
1 >>> a_tuple = (2, 3, 4)
2 >>> type(a_tuple)
3 <class 'tuple'>
```

However, parentheses by themselves do *not* make a tuple!

```
1 >>> not_a_tuple = (5)
2 >>> type(not_a_tuple)
3 <class 'int'>
```

You can cast a list into a tuple using the `tuple()` function:

```
1 >>> a_tuple = tuple(['1', '2', '3'])
2 >>> type(a_tuple)
3 <class 'tuple'>
```

This example demonstrates how to convert or cast a Python list into a tuple.

Working With Tuples

There are not many ways to work with tuples due to the fact that they are immutable. If you were to run `dir(tuple())`, you would find that tuples have only two methods:

- `count()`
- `index()`

You can use `count()` to find out how many elements match the value that you pass in:

```
1 >>> a_tuple = (1, 2, 3, 3)
2 >>> a_tuple.count(3)
3 2
```

In this case, you can find out how many times the integer 3 appears in the tuple.

You can use `index()` to find the first index of a value:

```
1 >>> a_tuple = (1, 2, 3, 3)
2 >>> a_tuple.index(2)
3 1
```

This example shows you that the number 2 is at index 1, which is the second item in the tuple. Tuples are zero-indexed, meaning that the first element starts at zero.

You can use the indexing methodology that you learned about in the previous chapter to access elements within a tuple:

```
1 >>> a_tuple = (1, 2, 3, 3)
2 >>> a_tuple[2]
3 3
```

The first “3” in the tuple is at index 2.

Let’s try to modify an element in your tuple:

```
1 >>> a_tuple[0] = 8
2 Traceback (most recent call last):
3     Python Shell, prompt 92, line 1
4 TypeError: 'tuple' object does not support item assignment
```

Here you try to set the first element in the tuple to 8. However, this causes a `TypeError` to be raised because tuples are immutable and cannot be changed.

Concatenating Tuples

Tuples can be joined together, which in programming is called “concatenation”. However, when you do that, you will end up creating a new tuple:

```
1 >>> a_tuple = (1, 2, 3, 3)
2 >>> id(a_tuple)
3 140617302751760
4 >>> a_tuple += (6, 7)
5 >>> id(a_tuple)
6 140617282270944
```

Here you concatenate a second tuple to the first tuple. You can use Python’s `id()` function to see that the variable, `a_tuple`, has changed. The `id()` function returns the id of the object. An object’s ID is equivalent to an address in memory. The ID number changed after concatenating the second tuple. That means that you have created a new object.

Special Case Tuples

There are two special-case tuples. A tuple with zero items and a tuple with one item. The reason they are special cases is that the syntax to create them is a little different.

To create an empty tuple, you can do one of the following:

```
1 >>> empty = tuple()
2 >>> len(empty)
3 0
4 >>> type(empty)
5 <class 'tuple'>
6 >>> also_empty = ()
7 >>> len(also_empty)
8 0
```

You can create an empty tuple by calling the `tuple()` function with no arguments or via assignment when using an empty pair of parentheses.

Now let's create a tuple with a single element:

```
1 >>> single = 2,
2 >>> len(single)
3 1
4 >>> type(single)
5 <class 'tuple'>
```

To create a tuple with a single element, you can assign a value with a following comma. Note the trailing comma after the 2 in the example above.

While the parentheses are usually optional, I highly recommend them for single-item tuples as the comma can be easy to miss.

Wrapping Up

The tuple is a fundamental data type in Python. It is used quite often and is certainly one that you should be familiar with. You will be using tuples in other data types. You will also use tuples to group related data, such as a name, address and country.

In this chapter, you learned how to create a tuple in three different ways. You also learned that tuples are immutable. Finally, you learned how to concatenate tuples and create empty tuples. Now you are ready to move on to the next chapter and learn all about dictionaries!

Review Questions

1. How do you create a tuple?
2. Can you show how to access the 3rd element in this tuple?

```
1 >>> a_tuple = (1, 2, 3, 4)
```

3. Is it possible to modify a tuple after you create it? Why or why not?
4. How do you create a tuple with a single item?

Chapter 8 - Learning About Dictionaries

Dictionaries are another fundamental data type in Python. A dictionary is a (key, value) pair. Some programming languages refer to them as hash tables. They are described as a *mapping* that maps keys (hashable objects) to values (any object). Immutable objects are hashable (*immutable* means unable to change).

Starting in Python 3.7, dictionaries are ordered. What that means is that when you add a new (key, value) pair to a dictionary, it remembers what order they were added. Prior to Python 3.7, this was not the case and you could not rely on insertion order.

You will learn how to do the following in this chapter:

- Creating dictionaries
- Accessing dictionaries
- Dictionary methods
- Modifying dictionaries
- Deleting items from your dictionary

Let's start off by learning about creating dictionaries!

Creating Dictionaries

You can create a dictionary in a couple of different ways. The most common method is by placing a comma-separated list of key: value pairs within curly braces.

Let's look at an example:

```
1 >>> sample_dict = {'first_name': 'James', 'last_name': 'Doe',  
2 'email': 'jdoe@gmail.com'}  
3 >>> sample_dict  
4 {'first_name': 'James', 'last_name': 'Doe', 'email': 'jdoe@gmail.com'}
```

You can also use Python's built-in `dict()` function to create a dictionary. `dict()` will accept a series of keyword arguments (i.e. `1='one', 2='two', etc`), a list of tuples, or another dictionary.

Here are a couple of examples:


```
1 >>> numbers = dict(one=1, two=2, three=3)
2 >>> numbers
3 {'one': 1, 'two': 2, 'three': 3}
4 >>> info_list = [('first_name', 'James'), ('last_name', 'Doe'),
5 ('email', 'jdoes@gmail.com')]
6 >>> info_dict = dict(info_list)
7 >>> info_dict
8 {'first_name': 'James', 'last_name': 'Doe', 'email': 'jdoes@gmail.com'}
```

The first example uses `dict()` on a series of keyword arguments. You will learn more about these when you learn about functions. You can think of keyword arguments as a series of keywords with the equals sign between them and their value.

The second example shows you how to create a list that has 3 tuples inside of it. Then you pass that list to `dict()` to convert it to a dictionary.

Accessing Dictionaries

Dictionaries' claim to fame is that they are very fast. You can access any value in a dictionary via the key. If the key is not found, you will receive a `KeyError`.

Let's take a look at how to use a dictionary:

```
1 >>> sample_dict = {'first_name': 'James', 'last_name': 'Doe',
2 'email': 'jdoe@gmail.com'}
3 >>> sample_dict['first_name']
4 'James'
```

To get the value of `first_name`, you must use the following syntax: `dictionary_name[key]`

Now let's try to get a key that doesn't exist:

```
1 >>> sample_dict['address']
2 Traceback (most recent call last):
3 Python Shell, prompt 118, line 1
4 builtins.KeyError: 'address'
```

Well that didn't work! You asked the dictionary to give you a value that wasn't in the dictionary!

You can use Python's `in` keyword to ask if a key is in the dictionary:

```
1 >>> 'address' in sample_dict
2 False
3 >>> 'first_name' in sample_dict
4 True
```

You can also check to see if a key is **not** in a dictionary by using Python's `not` keyword:

```
1 >>> 'first_name' not in sample_dict
2 False
3 >>> 'address' not in sample_dict
4 True
```

Another way to access keys in dictionaries is by using one of the dictionary methods. Let's find out more about dictionary methods now!

Dictionary Methods

As with most Python data types, dictionaries have special methods you can use. Let's check out some of the dictionary's methods!

`d.get(key[, default])`

You can use the `get()` method to get a value. `get()` requires you to specify a key to look for. It optionally allows you to return a default if the key is not found. The default for that value is `None`. Let's take a look:

```
1 >>> print(sample_dict.get('address'))
2 None
3 >>> print(sample_dict.get('address', 'Not Found'))
4 Not Found
```

The first example shows you what happens when you try to `get()` a key that doesn't exist without setting `get`'s default. In that case, it returns `None`. Then the second example shows you how to set the default to the string "Not Found".

`d.clear()`

The `clear()` method can be used to remove all the items from your dictionary.

```
1 >>> sample_dict = {'first_name': 'James', 'last_name': 'Doe',  
2 'email': 'jdoe@gmail.com'}  
3 >>> sample_dict  
4 {'first_name': 'James', 'last_name': 'Doe', 'email': 'jdoe@gmail.com'}  
5 >>> sample_dict.clear()  
6 >>> sample_dict  
7 {}
```

d.copy()

If you need to create a shallow copy of the dictionary, then the `copy()` method is for you:

```
1 >>> sample_dict = {'first_name': 'James', 'last_name': 'Doe',  
2 'email': 'jdoe@gmail.com'}  
3 >>> copied_dict = sample_dict.copy()  
4 >>> copied_dict  
5 {'first_name': 'James', 'last_name': 'Doe', 'email': 'jdoe@gmail.com'}
```

If your dictionary has objects or dictionaries inside of it, then you may end up running into logic errors as a result of using this method, because changing one dictionary will affect the other. In this case you should use Python's `copy` module, which has a `deepcopy` function that will create a completely separate copy for you.

You may remember this issue being mentioned back in the chapter on lists. These are common problems with creating “shallow” copies.

d.items()

The `items()` method will return a new view of the dictionary's items:

```
1 >>> sample_dict = {'first_name': 'James', 'last_name': 'Doe',  
2 'email': 'jdoe@gmail.com'}  
3 >>> sample_dict.items()  
4 dict_items([('first_name', 'James'), ('last_name', 'Doe'),  
5 ('email', 'jdoe@gmail.com')])
```

This view object will change as the dictionary object itself changes.

d.keys()

If you need to get a view of the keys that are in a dictionary, then `keys()` is the method for you. As a view object, it will provide you with a dynamic view of the dictionary's keys. You can iterate over a view and also check membership via the `in` keyword:

```
1 >>> sample_dict = {'first_name': 'James', 'last_name': 'Doe',
2 'email': 'jdoe@gmail.com'}
3 >>> keys = sample_dict.keys()
4 >>> keys
5 dict_keys(['first_name', 'last_name', 'email'])
6 >>> 'email' in keys
7 True
8 >>> len(keys)
9 3
```

d.values()

The `values()` method also returns a view object, but in this case it is a dynamic view of the dictionary's values:

```
1 >>> sample_dict = {'first_name': 'James', 'last_name': 'Doe',
2 'email': 'jdoe@gmail.com'}
3 >>> values = sample_dict.values()
4 >>> values
5 dict_values(['James', 'Doe', 'jdoe@gmail.com'])
6 >>> 'Doe' in values
7 True
8 >>> len(values)
9 3
```

d.pop(key[, default])

Do you need to remove a key from a dictionary? Then `pop()` is the method for you. The `pop()` method takes a key and an option default string. If you don't set the default and the key is not found, a `KeyError` will be raised.

Here are some examples:

```
1 >>> sample_dict = {'first_name': 'James', 'last_name': 'Doe',
2 'email': 'jdoe@gmail.com'}
3 >>> sample_dict.pop('something')
4 Traceback (most recent call last):
5   Python Shell, prompt 146, line 1
6 builtins.KeyError: 'something'
7 >>> sample_dict.pop('something', 'Not found!')
8 'Not found!'
9 >>> sample_dict.pop('first_name')
```

```
10 'James'
11 >>> sample_dict
12 {'last_name': 'Doe', 'email': 'jdoe@gmail.com'}
```

d.popitem()

The `popitem()` method is used to remove and return a (key, value) pair from the dictionary. The pairs are returned in last-in first-out (LIFO) order, which means that the last item added will also be the first one that is removed when you use this method. If called on an empty dictionary, you will receive a `KeyError`.

```
1 >>> sample_dict = {'first_name': 'James', 'last_name': 'Doe',
2 'email': 'jdoe@gmail.com'}
3 >>> sample_dict.popitem()
4 ('email', 'jdoe@gmail.com')
5 >>> sample_dict
6 {'first_name': 'James', 'last_name': 'Doe'}
```

d.update([other])

Update a dictionary with the (key, value) pairs from *other*, overwriting existing keys. The other can be another dictionary, a list of tuples, etc.

`update()` will return `None` when called.

Let's look at a couple of examples:

```
1 >>> sample_dict = {'first_name': 'James', 'last_name': 'Doe',
2 'email': 'jdoe@gmail.com'}
3 >>> sample_dict.update([('something', 'else')])
4 >>> sample_dict
5 {'first_name': 'James',
6 'last_name': 'Doe',
7 'email': 'jdoe@gmail.com',
8 'something': 'else'}
```

Let's try using `update()` to overwrite a pre-existing key:

```
1 >>> sample_dict = {'first_name': 'James', 'last_name': 'Doe',  
2 'email': 'jdoe@gmail.com'}  
3 >>> sample_dict.update([('first_name', 'Mike')])  
4 >>> sample_dict  
5 {'first_name': 'Mike', 'last_name': 'Doe', 'email': 'jdoe@gmail.com'}
```

Modifying Your Dictionary

You will need to modify your dictionary from time to time. Let's assume that you need to add a new (key, value) pair:

```
1 >>> sample_dict = {'first_name': 'James', 'last_name': 'Doe',  
2 'email': 'jdoe@gmail.com'}  
3 >>> sample_dict['address'] = '123 Dunn St'  
4 >>> sample_dict  
5 {'first_name': 'James',  
6 'last_name': 'Doe',  
7 'email': 'jdoe@gmail.com',  
8 'address': '123 Dunn St'  
9 }
```

To add a new item to a dictionary, you can use the square braces to enter a new key and set it to a value.

If you need to update a pre-existing key, you can do the following:

```
1 >>> sample_dict = {'first_name': 'James', 'last_name': 'Doe',  
2 'email': 'jdoe@gmail.com'}  
3 >>> sample_dict['email'] = 'jame@doe.com'  
4 >>> sample_dict  
5 {'first_name': 'James', 'last_name': 'Doe', 'email': 'jame@doe.com'}
```

In this example, you set `sample_dict['email']` to `jame@doe.com`. Whenever you set a pre-existing key to a new value, you will overwrite the previous value.

You can also use the `update()` method from the previous section to modify your dictionary.

Deleting Items From Your Dictionary

Sometimes you will need to remove a key from a dictionary. You can use Python's `del` keyword for that:

```
1 >>> sample_dict = {'first_name': 'James', 'last_name': 'Doe',  
2 'email': 'jdoe@gmail.com'}  
3 >>> del sample_dict['email']  
4 >>> sample_dict  
5 {'first_name': 'James', 'last_name': 'Doe'}
```

In this case, you tell Python to delete the key “email” from `sample_dict`

The other method for removing a key is to use the dictionary’s `pop()` method, which was mentioned in the previous section:

```
1 >>> sample_dict = {'first_name': 'James', 'last_name': 'Doe',  
2 'email': 'jdoe@gmail.com'}  
3 >>> sample_dict.pop('email')  
4 'jdoe@gmail.com'  
5 >>> sample_dict  
6 {'first_name': 'James', 'last_name': 'Doe'}
```

When you use `pop()`, it will remove the key and return the value that is being removed.

Wrapping Up

The dictionary data type is extremely useful. You will find it handy to use for quick lookups of all kinds of data. You can set the value of the key: value pair to any object in Python. So you could store lists, tuples, and other objects as values in a dictionary.

You learned the following topics in this chapter:

- Creating dictionaries
- Accessing dictionaries
- Dictionary methods
- Modifying dictionaries
- Deleting items from your dictionary

It is fairly common to need a dictionary that will create a key when you try to access one that does not exist. If you have such a need, you should check out Python’s `collections` module. It has a `defaultdict` class that is made for exactly that use case.

Review Questions

1. How do you create a dictionary?
2. You have the following dictionary. How do you change the `last_name` field to ‘Smith’?

```
1 >>> my_dict = {'first_name': 'James', 'last_name': 'Doe', 'email': 'jdoe@gmail.com'}
```

3. Using the dictionary above, how would you remove the `email` field from the dictionary?
4. How do you get just the values from a dictionary?

Chapter 9 - Learning About Sets

A set data type is defined as an “unordered collection of distinct hashable objects” according to the Python 3 documentation. You can use a set for membership testing, removing duplicates from a sequence and computing mathematical operations, like intersection, union, difference, and symmetric difference.

Due to the fact that they are unordered collections, a set does not record element position or order of insertion. Because of that, they also do not support indexing, slicing or other sequence-like behaviors that you have seen with lists and tuples.

There are two types of set built-in to the Python language:

- set - which is mutable
- frozenset - which is immutable and hashable

This chapter will focus on set.

You will learn how to do the following with sets:

- Creating a set
- Accessing set members
- Changing items
- Adding items
- Removing items
- Deleting a set

Let's get started by creating a set!

Creating a Set

Creating a set is pretty straight-forward. You can create them by adding a series of comma-separated objects inside of curly braces or you can pass a sequence to the built-in `set()` function.

Let's look at an example:

```
1 >>> my_set = {"a", "b", "c", "c"}
2 >>> my_set
3 {'c', 'a', 'b'}
4 >>> type(my_set)
5 <class 'set'>
```

A set uses the same curly braces that you used to create a dictionary. Note that instead of key:value pairs, you have a series of values. When you print out the set, you can see that duplicates were removed automatically.

Now let's try creating a set using `set()`:

```
1 >>> my_list = [1, 2, 3, 4]
2 >>> my_set = set(my_list)
3 >>> my_set
4 {1, 2, 3, 4}
5 >>> type(my_set)
6 <class 'set'>
```

In this example, you created a list and then cast it to a set using `set()`. If there had been any duplicates in the list, they would have been removed.

Now let's move along and see some of the things that you can do with this data type.

Accessing Set Members

You can check if an item is in a set by using Python's `in` operator:

```
1 >>> my_set = {"a", "b", "c", "c"}
2 >>> "a" in my_set
3 True
```

Sets do not allow you to use slicing or the like to access individual members of the set. Instead, you need to iterate over a set. You can do that using a loop, such as a `while` loop or a `for` loop.

You won't be covering loops until **chapter 12**, but here is the basic syntax for iterating over a collection using a `for` loop:

```
1 >>> for item in my_set:
2     ...     print(item)
3     ...
4 c
5 a
6 b
```

This will loop over each item in the set one at a time and print it out.

You can access items in sets much faster than lists. A Python `list` will iterate over each item in a `list` until it finds the item you are looking for. When you look for an item in a set, it acts much like a dictionary and will find it immediately or not at all.

Changing Items

While both `dict` and `set` require hashable members, a set has no value to change. However, you can add items to a set as well as remove them. Let's find out how!

Adding Items

There are two ways to add items to a set:

- `add()`
- `update()`

Let's try adding an item using `add()`:

```
1 >>> my_set = {"a", "b", "c", "c"}
2 >>> my_set.add('d')
3 >>> my_set
4 {'d', 'c', 'a', 'b'}
```

That was easy! You were able to add an item to the set by passing it into the `add()` method.

If you'd like to add multiple items all at once, then you should use `update()` instead:

```
1 >>> my_set = {"a", "b", "c", "c"}
2 >>> my_set.update(['d', 'e', 'f'])
3 >>> my_set
4 {'a', 'c', 'd', 'e', 'b', 'f'}
```

Note that `update()` will take any iterable you pass to it. So it could take, for example, a list, tuple or another set.

Removing Items

You can remove items from sets in several different ways.

You can use:

- `remove()`
- `discard()`
- `pop()`

Let's go over each of these in the following sub-sections!

Using `.remove()`

The `remove()` method will attempt to remove the specified item from a set:

```
1 >>> my_set = {"a", "b", "c", "c"}
2 >>> my_set.remove('a')
3 >>> my_set
4 {'c', 'b'}
```

If you happen to ask the set to `remove()` an item that does not exist, you will receive an error:

```
1 >>> my_set = {"a", "b", "c", "c"}
2 >>> my_set.remove('f')
3 Traceback (most recent call last):
4   Python Shell, prompt 208, line 1
5 builtins.KeyError: 'f'
```

Now let's see how the closely related `discard()` method works!

Using `.discard()`

The `discard()` method works in almost exactly the same way as `remove()` in that it will remove the specified item from the set:

```
1 >>> my_set = {"a", "b", "c", "c"}
2 >>> my_set.discard('b')
3 >>> my_set
4 {'c', 'a'}
```

The difference with `discard()` though is that it **won't** throw an error if you try to remove an item that doesn't exist:

```
1 >>> my_set = {"a", "b", "c", "c"}
2 >>> my_set.discard('d')
3 >>>
```

If you want to be able to catch an error when you attempt to remove an item that does not exist, use `remove()`. If that doesn't matter to you, then `discard()` might be a better choice.

Using `.pop()`

The `pop()` method will remove and return an arbitrary item from the set:

```
1 >>> my_set = {"a", "b", "c", "c"}
2 >>> my_set.pop()
3 'c'
4 >>> my_set
5 {'a', 'b'}
```

If your set is empty and you try to `pop()` an item out, you will receive an error:

```
1 >>> my_set = {"a"}
2 >>> my_set.pop()
3 'a'
4 >>> my_set.pop()
5 Traceback (most recent call last):
6   Python Shell, prompt 219, line 1
7   builtins.KeyError: 'pop from an empty set'
```

This is very similar to the way that `pop()` works with the `list` data type, except that with a `list`, it will raise an `IndexError`. Also lists are ordered while sets are not, so you can't be sure what you will be removing with `pop()` since sets are not ordered.

Clearing or Deleting a Set

Sometimes you will want to empty a set or even completely remove it.

To empty a set, you can use `clear()`:

```
1 >>> my_set = {"a", "b", "c", "c"}
2 >>> my_set.clear()
3 >>> my_set
4 set()
```

If you want to completely remove the set, then you can use Python's `del` built-in:

```
1 >>> my_set = {"a", "b", "c", "c"}
2 >>> del my_set
3 >>> my_set
4 Traceback (most recent call last):
5   Python Shell, prompt 227, line 1
6 builtins.NameError: name 'my_set' is not defined
```

Now let's learn what else you can do with sets!

Set Operations

Sets provide you with some common operations such as:

- `union()` - Combines two sets and returns a new set
- `intersection()` - Returns a new set with the elements that are common between the two sets
- `difference()` - Returns a new set with elements that are not in the other set

These operations are the most common ones that you will use when working with sets.

The `union()` method is actually kind of like the `update()` method that you learned about earlier, in that it combines two or more sets together into a new set. However the difference is that it returns a new set rather than updating the original set with new items:

```
1 >>> first_set = {'one', 'two', 'three'}
2 >>> second_set = {'orange', 'banana', 'peach'}
3 >>> first_set.union(second_set)
4 {'two', 'banana', 'three', 'peach', 'orange', 'one'}
5 >>> first_set
6 {'two', 'three', 'one'}
```

In this example, you create two sets. Then you use `union()` on the first set to add the second set to it. However `union` doesn't update the set. It creates a new set. If you want to save the new set, then you should do the following instead:

```
1 >>> united_set = first_set.union(second_set)
2 >>> united_set
3 {'two', 'banana', 'three', 'peach', 'orange', 'one'}
```

The `intersection()` method takes two sets and returns a new set that contains only the items that are the same in both of the sets.

Let's look at an example:

```
1 >>> first_set = {'one', 'two', 'three'}
2 >>> second_set = {'orange', 'banana', 'peach', 'one'}
3 >>> first_set.intersection(second_set)
4 {'one'}
```

These two sets have only one item in common: the string “one”. So when you call `intersection()`, it returns a new set with a single element in it. As with `union()`, if you want to save off this new set, then you would want to do something like this:

```
1 >>> intersection = first_set.intersection(second_set)
2 >>> intersection
3 {'one'}
```

The `difference()` method will return a new set with the elements in the set that are **not** in the other set. This can be a bit confusing, so let's look at a couple of examples:

```
1 >>> first_set = {'one', 'two', 'three'}
2 >>> second_set = {'three', 'four', 'one'}
3 >>> first_set.difference(second_set)
4 {'two'}
5 >>> second_set.difference(first_set)
6 {'four'}
```

When you call `difference()` on the `first_set`, it returns a set with “two” as its only element. This is because “two” is the only string not found in the `second_set`. When you call `difference()` on the `second_set`, it will return “four” because “four” is not in the `first_set`.

There are other methods that you can use with sets, but they are used pretty infrequently. You should go check the documentation for full details on set methods should you need to use them.

Wrapping Up

Sets are a great data type that is used for pretty specific situations. You will find sets most useful for de-duplicating lists or tuples or by using them to find differences between multiple lists.

In this chapter, you learned about the following:

- Creating a set
- Accessing set members
- Changing items
- Adding items
- Removing items
- Deleting a set

Any time you need to use a set-like operation, you should take a look at this data type. However, in all likelihood, you will be using lists, dictionaries, and tuples much more often.

Review Questions

1. How do you create a set?
2. Using the following set, how would you check to see if it contains the string, “b”?

```
1 >>> my_set = {"a", "b", "c", "c"}
```

3. How do you add an item to a set?
4. Remove the letter “c” from the following set using a set method:

```
1 >>> my_set = {"a", "b", "c", "c"}
```

5. How do you find the common items between two sets?

Chapter 10 - Boolean Operations and None

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

The bool() Function

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

What About None?

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Wrapping Up

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Review Questions

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Chapter 11 - Conditional Statements

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Comparison Operators

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Creating a Simple Conditional

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Branching Conditional Statements

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Nesting Conditionals

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Logical Operators

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Special Operators

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Wrapping Up

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Review Questions

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Chapter 12 - Learning About Loops

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Creating a `for` Loop

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Looping Over a String

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Looping Over a Dictionary

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Extracting Multiple Values in a Tuple While Looping

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Using `enumerate` with Loops

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Creating a `while` Loop

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Breaking Out of a Loop

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Using `continue`

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Loops and the `else` Statement

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Nesting Loops

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Wrapping Up

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Review Questions

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Chapter 13 - Python Comprehensions

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

List Comprehensions

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Filtering List Comprehensions

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Nested List Comprehensions

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Dictionary Comprehensions

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Set Comprehensions

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Wrapping Up

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Review Questions

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Chapter 14 - Exception Handling

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

The Most Common Exceptions

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Handling Exceptions

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Raising Exceptions

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Examining the Exception Object

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Using the `finally` Statement

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Using the `else` Statement

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Wrapping Up

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Review Questions

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Chapter 15 - Working with Files

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

The open() Function

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Reading Files

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Reading Binary Files

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Writing Files

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Seeking Within a File

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Appending to Files

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Catching File Exceptions

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Wrapping Up

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Review Questions

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Chapter 16 - Importing

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Using `import`

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Using `from` to Import Specific Bits & Pieces

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Using `as` to assign a new name

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Importing Everything

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Wrapping Up

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Review Questions

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Chapter 17 - Functions

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Creating a Function

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Calling a Function

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Passing Arguments

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Type Hinting Your Arguments

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Passing Keyword Arguments

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Required and Default Arguments

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

What are `*args` and `**kwargs`?

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Positional-only Parameters

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Scope

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Wrapping Up

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Review Questions

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Chapter 18 - Classes

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Class Creation

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Figuring Out `self`

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Public and Private Methods / Attributes

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Subclass Creation

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Polymorphism

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Making the Class Nicer

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Wrapping Up

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Review Questions

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Part II - Beyond the Basics

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Chapter 19 - Introspection

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Using the `type()` Function

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Using the `dir()` Function

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Getting `help()`

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Other Built-in Introspection Tools

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Using `callable()`

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Using `len()`

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Using `locals()`

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Using `globals()`

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Wrapping Up

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Review Questions

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Chapter 20 - Installing Packages with pip

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Installing a Package

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Exploring Command Line Options

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Installing with `requirements.txt`

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Upgrading a Package

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Checking What's Installed

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Uninstalling Packages

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Alternatives to pip

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Wrapping Up

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Review Questions

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Chapter 21 - Python Virtual Environments

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Python's `venv` Library

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

The `virtualenv` Package

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Other Tools

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Wrapping Up

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Review Questions

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Chapter 22 - Type Checking in Python

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Pros and Cons of Type Hinting

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Built-in Type Hinting / Variable Annotation

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Collection Type Hinting

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Hinting Values That Could be None

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Type Hinting Functions

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

What To Do When Things Get Complicated

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Classes

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Decorators

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Aliasing

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Other Type Hints

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Type Comments

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Static Type Checking

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Wrapping Up

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Review Questions

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Chapter 23 - Creating Multiple Threads

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Pros of Using Threads

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Cons of Using Threads

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Creating Threads

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Subclassing Thread

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Writing Multiple Files with Threads

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Wrapping Up

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Review Questions

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Chapter 24 - Creating Multiple Processes

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Pros of Using Processes

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Cons of Using Processes

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Creating Processes with multiprocessing

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Subclassing Process

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Creating a Process Pool

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Wrapping Up

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Review Questions

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Chapter 25 - Launching Subprocesses with Python

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

The `subprocess.run()` Function

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Getting the Output

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

The `subprocess.Popen()` Class

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

The `subprocess.Popen.communicate()` Function

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Reading and Writing with `stdin` and `stdout`

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Wrapping Up

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Review Questions

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Chapter 26 - Debugging Your Code with pdb

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Starting pdb in the REPL

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Starting pdb on the Command Line

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Stepping Through Code

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Adding Breakpoints in pdb

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Creating a Breakpoint with `set_trace()`

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Using the built-in `breakpoint()` Function

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Getting Help

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Wrapping Up

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Review Questions

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Chapter 27 - Learning About Decorators

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Creating a Function

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Creating a Decorator

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Applying a Decorator with @

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Creating a Decorator for Logging

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Stacking Decorators

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Passing Arguments to Decorators

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Using a Class as a Decorator

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Python's Built-in Decorators

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Python Properties

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Wrapping Up

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Review Questions

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Chapter 28 - Assignment Expressions

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Using Assignment Expressions

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

What You Cannot Do With Assignment Expressions

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Wrapping Up

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Review Questions

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Chapter 29 - Profiling Your Code

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Learning How to Profile with `cProfile`

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Profiling a Python Script with `cProfile`

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Working with Profile Data Using `pstats`

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Other Profilers

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

`line_profiler`

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

`memory_profiler`

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

profilehooks

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

scalene

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

snakeviz

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Wrapping Up

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Review Questions

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Chapter 30 - An Introduction to Testing

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Using `doctest` in the Terminal

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Using `doctest` in Your Code

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Using `doctest` From a Separate File

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Using `unittest` For Test Driven Development

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

The Fizz Test

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

The Buzz Test

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

The FizzBuzz Test

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

The Final Test

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Wrapping Up

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Review Questions

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Chapter 31 - Learning About the Jupyter Notebook

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Installing The Jupyter Notebook

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Creating a Notebook

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Naming Your Notebook

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Running Cells

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Learning About the Menus

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Adding Content

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Creating Markdown Content

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Formatting Your Text

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Using Headings

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Adding a Listing

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Highlighting Code Syntax

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Creating a Hyperlink

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Adding an Extension

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Exporting Notebooks to Other Formats

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Wrapping Up

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Review Questions

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Part III - Practical Python

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Chapter 32 - How to Create a Command-line Application with argparse

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Parsing Arguments

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Creating Helpful Messages

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Adding Aliases

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Using Mutually Exclusive Arguments

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Creating a Simple Search Utility

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Wrapping Up

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Review Questions

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Chapter 33 - How to Parse XML

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Parsing XML with `ElementTree`

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Creating XML with `ElementTree`

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Editing XML with `ElementTree`

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Manipulating XML with `lxml`

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Wrapping Up

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Review Questions

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Chapter 34 - How to Parse JSON

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Encoding a JSON String

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Saving JSON to Disk

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Decoding a JSON String

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Loading JSON from Disk

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Validating JSON with `json.tool`

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Wrapping Up

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Review Questions

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Chapter 35 - How to Scrape a Website

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Rules for Web Scraping

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Preparing to Scrape a Website

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Scraping a Website

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Downloading a File

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Wrapping Up

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Review Questions

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Chapter 36 - How to Work with CSV files

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Reading a CSV File

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Reading a CSV File with DictReader

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Writing a CSV File

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Writing a CSV File with DictWriter

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Wrapping Up

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Review Questions

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Chapter 37 - How to Work with a Database Using `sqlite3`

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Creating a SQLite Database

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Adding Data to Your Database

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Searching Your Database

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Editing Data in Your Database

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Deleting Data From Your Database

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Wrapping Up

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Review Questions

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Chapter 38 - Working with an Excel Document in Python

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Python Excel Packages

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Getting Sheets from a Workbook

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Reading Cell Data

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Iterating Over Rows and Columns

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Writing Excel Spreadsheets

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Adding and Removing Sheets

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Adding and Deleting Rows and Columns

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Wrapping Up

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Review Questions

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Chapter 39 - How to Generate a PDF

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Installing ReportLab

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Creating a Simple PDF with the Canvas

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Creating Drawings and Adding Images Using the Canvas

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Creating Multi-page Documents with PLATYPUS

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Creating a Table

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Wrapping Up

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Review Questions

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Chapter 40 - How to Create Graphs

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Installing Matplotlib

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Creating a Simple Line Chart with PyPlot

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Creating a Bar Chart

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Creating a Pie Chart

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Adding Labels

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Adding Titles to Plots

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Creating a Legend

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Showing Multiple Figures

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Wrapping Up

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Review Questions

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Chapter 41 - How to Work with Images in Python

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Installing Pillow

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Opening Images

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Cropping Images

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Using Filters

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Adding Borders

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Resizing Images

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Wrapping Up

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Review Questions

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Chapter 42 - How to Create a Graphical User Interface

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Installing wxPython

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Learning About Event Loops

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

How to Create Widgets

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

How to Lay Out Your Application

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

How to Add Events

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

How to Create an Application

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Wrapping Up

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Review Questions

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Part IV - Distributing Your Code

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Chapter 43 - How to Create a Python Package

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Creating a Module

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Creating a Package

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Packaging a Project for PyPI

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Creating Project Files

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Creating `setup.py`

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Generating a Python Wheel

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Uploading to PyPI

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Wrapping Up

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Review Questions

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Chapter 44 - How to Create an Exe for Windows

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Installing PyInstaller

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Creating an Executable for a Command-Line Application

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Creating an Executable for a GUI

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Wrapping Up

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Review Questions

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Chapter 45 - How to Create an Installer for Windows

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Installing Inno Setup

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Creating an Installer

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Testing Your Installer

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Wrapping Up

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Review Questions

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Chapter 46 - How to Create an “exe” for Mac

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Installing PyInstaller

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Creating an Executable with PyInstaller

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Wrapping Up

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Review Questions

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Afterword

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Appendix A - Version Control

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Version Control Systems

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Distributed vs Centralized Versioning

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Common Terminology

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Branch

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Changelist / Changeset

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Checkout

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Clone

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Commit

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Diff

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Fork

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Head

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Initialize

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Mainline

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Merge

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Pull / Push

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Pull Request

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Repository

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Resolve

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Stream

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Tag

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Trunk

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Python IDE Version Control Support

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Wrapping Up

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Appendix B - Version Control with Git

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Installing Git

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Installing on Windows

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Installing on MacOS

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Installing on Linux

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Configuring Git

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Creating a Project

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Ignoring Files

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Initializing a Repository

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Checking the Project Status

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Adding Files to a Repository

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Committing Files

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Viewing the Log

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Changing a File

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Reverting a File

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Checking Out Previous Commits

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Pushing to Github

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Wrapping Up

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Review Question Answer Key

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Chapter 3 - Documenting Your Code

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

1) How do you create a comment?

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

2) What do you use a docstring for?

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

3) What is Python's style guide?

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

4) Why is documenting your code important?

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Chapter 4 - Working with Strings

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

1) What are 3 ways to create a string?

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

2) Run `dir("")`. This lists all the string methods you can use. Which of these methods will capitalize each of the words in a sentence?

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

3) Change the following example to use f-strings:

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

4) How do you concatenate these two strings together?

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Chapter 5 - Numeric Types

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

1) What 3 numeric types does Python support without importing anything?

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

2) Which module should you use for money or other precise calculations?

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

3) Give an example of how to use augmented assignment.

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Chapter 6 - Learning About Lists

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

1) How do you create a list?

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

2) Create a list with 3 items and then use `append()` to add two more.

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

3) What is wrong with this code?

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

4) How do you remove the 2nd item in this list?

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

5) Create a list that looks like this: `[4, 10, 2, 1, 23]`. Use string slicing to get only the middle 3 items.

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Chapter 7 - Learning About Tuples

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

1) How do you create a tuple?

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

2) Can you show how to access the 3rd element in this tuple?

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

3) Is it possible to modify a tuple after you create it? Why or why not?

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

4) How do you create a tuple with a single item?

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Chapter 8 - Learning About Dictionaries

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

1) How do you create a dictionary?

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

2) You have the following dictionary. How do you change the `last_name` field to 'Smith'?

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

3) Using the dictionary above, how would you remove the `email` field from the dictionary?

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

4) How do you get just the values from a dictionary?

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Chapter 9 - Learning About Sets

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

1) How do you create a `set`?

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

2) Using the following `set`, how would you check to see if it contains the string, `"b"`?

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

3) How do you add an item to a `set`?

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

4) Remove the letter `"c"` from the following `set` using a `set` method:

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

5) How do you find the common items between two sets?

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Chapter 10 - Boolean Operations and None

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

1) What number does `True` equal?

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

2) How do you cast other data types to `True` or `False`?

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

3) What is Python's null type?

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Chapter 11 - Conditional Statements

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

1) Give a couple of examples of comparison operators:

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

2) Why does indentation matter in Python?

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

3) How do you create a conditional statement?

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

4) How do you use logical operators to check more than one thing at once?

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

5) What are some examples of special operators?

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

6) What is the difference between these two?

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Chapter 12 - Learning About Loops

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

1) What two types of loops does Python support?

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

2) How do you loop over a string?

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

3) What keyword do you use to exit a loop?

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

4) How do you “skip” over an item when you are iterating?

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

5) What is the `else` statement for in loops?

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

6) What are the flow control statements in Python?

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Chapter 13 - Python Comprehensions

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

1) How do you create a list comprehension?

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

2) What is a good use case for a list comprehension?

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

3) Create a dictionary using a dict comprehension.

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

4) Create a `set` using a set comprehension.

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Chapter 14 - Exception Handling

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

1) What are a couple of common exceptions?

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

2) How do you catch an exception in Python?

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

3) What do you need to do to raise a run time error?

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

4) What is the `finally` statement for?

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

5) How is the `else` statement used with an exception handler?

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Chapter 15 - Working with Files

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

1) How do you open a file?

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

2) What do you need to do to read a file?

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

3) Write the following sentence to a file named test.txt:

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

4) How do you append new data to a pre-existing file?

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

5) What do you need to do to catch a file exception?

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Chapter 16 - Importing

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

1) How do you include the `math` library from Python's standard library in your code?

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

2) How do you include `cos` from the `math` library in your own code?

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

3) How do you import a module/function/etc. with a different name?

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

4) Python has a special syntax you can use to include everything. What is it?

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Chapter 17 - Functions

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

1) How do you create a function that accepts two positional arguments?

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

2) Create a function named `address_builder` that accepts the following and add type hints:

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

3) Using the function from question 2, give the zip code a default of 55555

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

4) How do you allow an arbitrary number of keyword arguments to be passed to a function?

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

5) What syntax do you use to force a function to use positional-only parameters?

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Chapter 18 - Classes

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

1) How do you create a class in Python?

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

2) What do you name a class initializer?

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

3) Explain the use of `self` in your own words

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

4) What does overriding a method do?

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

5) What is a subclass?

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Chapter 19 - Introspection

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

1) What is introspection?

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

2) What is the `type()` function used for?

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

3) How is `dir()` helpful?

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Chapter 20 - Installing Packages with pip

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

1) How do you install a package with `pip`?

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

2) What command do you use to see the version of the packages you installed?

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

3) How do you uninstall a package?

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Chapter 21 - Python Virtual Environments

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

1) How do you create a Python virtual environment?

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

2) What do you need to do after creating a virtual environment to use it?

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

3) Why would you use a Python virtual environment?

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Chapter 22 - Type Checking in Python

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

1) What is type hinting in your own words?

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

2) Why would you use type hinting?

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

3) Demonstrate your understanding of type hinting by adding type annotations to the variables as well as the function. Don't forget the return type!

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Chapter 23 - Creating Multiple Threads

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

1) What are threads good for?

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

2) Which module do you use to create a thread in Python?

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

3) What is the Global Interpreter Lock?

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Chapter 24 - Creating Multiple Processes

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

1) What are processes good for?

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

2) How do you create a process in Python?

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

3) Can you create a process pool in Python? How?

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

4) What effect, if any, does the Global Interpreter Lock have on processes?

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

5) What happens if you don't use `process.join()`?

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Chapter 25 - Launching Subprocesses with Python

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

1) How would you launch Microsoft Notepad or your favorite text editor with Python?

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

2) Which method do you use to get the result from a process?

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

3) How do you get `subprocess` to return strings instead of bytes?

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Chapter 26 - Debugging Your Code

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

1) What is `pdb`?

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

2) How do you use `pdb` to get to a specific location in your code?

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

3) What is a breakpoint?

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

4) What is a callstack?

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Chapter 27 - Learning About Decorators

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

1) What is a decorator?

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

2) What special syntax do you use to apply a decorator?

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

3) Name at least two of Python's built-in decorators

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

4) What is a Python property?

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Chapter 28 - Assignment Expressions

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

1) What is an assignment expression?

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

2) How do you create an assignment expression?

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

3) Why would you use assignment expressions?

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Chapter 29 - Profiling Your Code

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

1) What does “profiling” mean in a programming context?

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

2) Which Python library do you use to profile your code?

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

3) How do you extract data from saved profile statistics?

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Chapter 30 - An Introduction to Testing

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

1) How do you add tests to be used with doctest?

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

2) Are tests for doctest required to be in the docstring? If not, how would you execute it?

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

3) What is a unit test?

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

4) What is test driven development?

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Chapter 31 - Learning About the Jupyter Notebook

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

1) What is Jupyter Notebook?

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

2) Name two Notebook cell types

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

3) What markup language do you use to format text in Jupyter Notebook?

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

4) How do you export a Notebook to another format?

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Chapter 32 - How to Create a Command Line Application with `argparse`

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

1) Which module in the standard library can you use to create a command-line application?

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

2) How do you add arguments to the `ArgumentParser()`?

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

3) How do you create helpful messages for your users?

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

4) Which method do you use to create a mutually exclusive group of commands?

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Chapter 33 - How to Parse XML

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

1) What XML modules are available in Python's standard library?

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

2) How do you access an XML tag using `ElementTree`?

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

3) How do you get the root element using the `ElementTree` API?

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Chapter 34 - How to Parse JSON

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

1) What is JSON?

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

2) How do you decode a JSON string in Python?

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

3) How do you save JSON to disk with Python?

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Chapter 35 - How to Scrape a Website

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

1) What are some popular Python web scraping packages?

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

2) How do you examine a web page with your browser?

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

3) Which Python module from the standard library do you use to download a file?

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Chapter 36 - How to Work with CSV files

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

1) How do you read a CSV file with Python's standard library?

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

2) If your CSV file doesn't use commas as the delimiter, how do you use the `csv` module to read it?

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

3) How do you write a row of CSV data using the `csv` module?

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Chapter 37 - How to Work with a Database Using `sqlite`

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

1) How do you create a database with the `sqlite3` library?

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

2) Which SQL command is used to add data to a table?

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

3) How do you change a field in a database with SQL?

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

4) What are SQL queries used for?

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

5) By default, how many records in a table will DELETE affect? How about UPDATE and SELECT?

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

6) The `delete_author` function above is susceptible to an SQL Injection attack. Why, and how would you fix it?

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Chapter 38 - Working with an Excel Document in Python

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

1) What Python package can you use to work with Microsoft Excel spreadsheets?

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

2) How do you open an Excel spreadsheet with Python?

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

3) Which class do you use to create an Excel spreadsheet with OpenPyXL?

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Chapter 39 - How to Generate a PDF

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

1) What class in ReportLab do you use to draw directly on the PDF at a low-level?

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

2) What does PLATYPUS stand for?

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

3) How do you apply a stylesheet to a Paragraph?

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

4) Which method do you use to apply a `TableStyle`?

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Chapter 40 - How to Create Graphs

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

1) Which module in Matplotlib do you use to create plots?

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

2) How do you add a label to the x-axis of a plot?

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

3) Which functions do you use to create titles and legends for plots?

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Chapter 41 - How to Work with Images in Python

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

1) How do you get the width and height of a photo using Pillow?

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

2) Which method do you use to apply a border to an image?

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

3) How do you resize an image with Pillow while maintaining its aspect ratio?

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Chapter 42 - How to Create a Graphical User Interface

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

1) What is a GUI?

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

2) What is an event loop?

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

3) How do you layout widgets in your application?

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Chapter 43 - How to Create a Python Package

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

1) What is a module?

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

2) How is a package different from a module?

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

3) Name at least two tools that are required for packaging a project?

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

4) What are some of the files you need to include in a package that are not Python files?

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Chapter 44 - How to Create an Exe for Windows

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

1) Name 3 different tools you can use to create a Windows executable out of Python code.

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

2) What command do you use with PyInstaller to create an executable?

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

3) How do you create a single file executable with PyInstaller?

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

4) Which flag do you use with PyInstaller to suppress the console window?

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Chapter 45 - How to Create an Installer for Windows

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

1) Name two programs you can use to create installers for Windows.

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

2) How do you modify an Inno Setup compiler script?

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

3) Why should you test your installers?

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

Chapter 46 - How to Create an “exe” for Mac

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

1) What tools can you use to create executables for Python on MacOS?

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.

2) What is an executable called on MacOS?

This content is not available in the sample book. The book can be purchased on Leanpub at <http://leanpub.com/py101>.