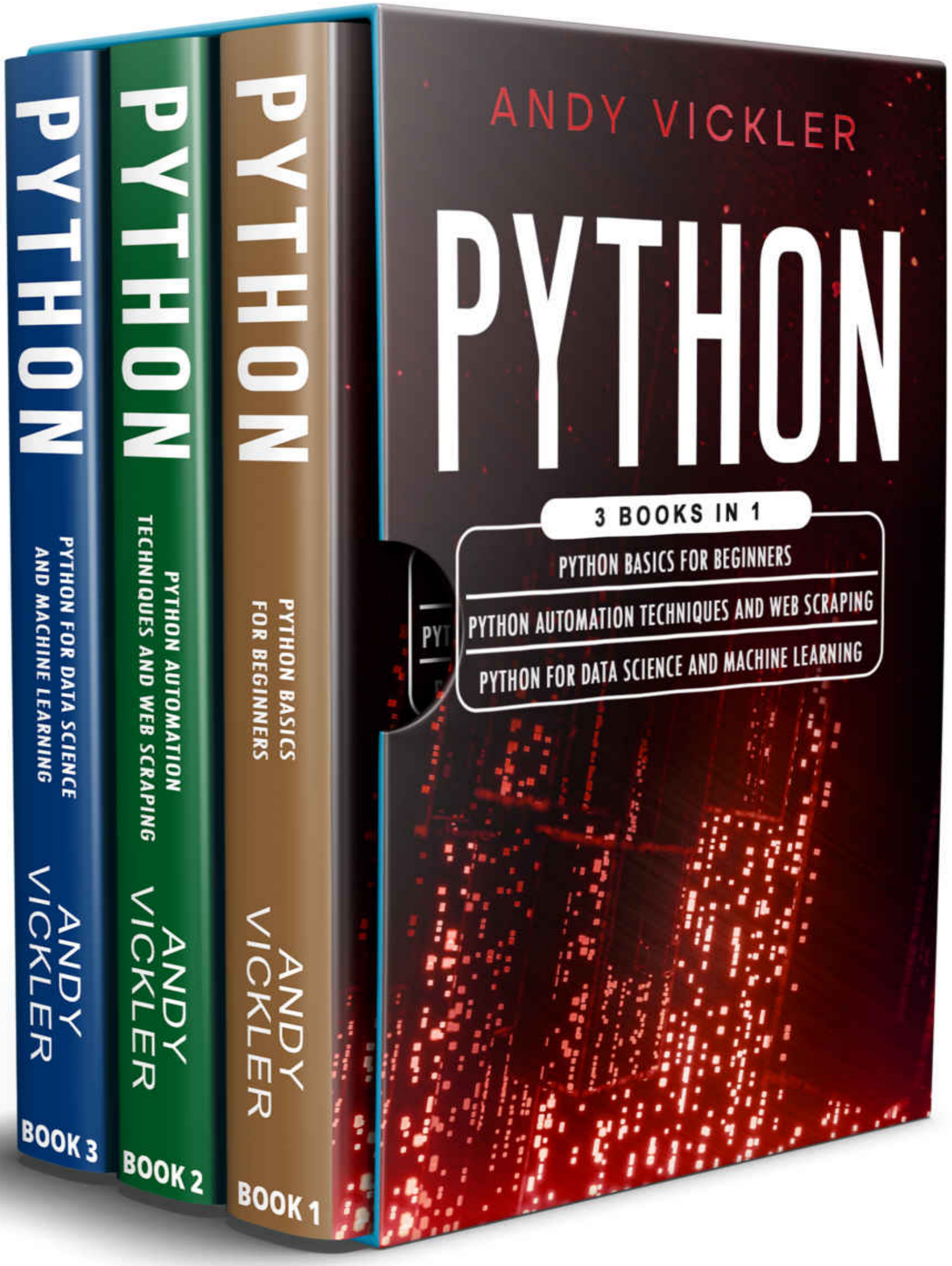

PYTHON
PYTHON FOR DATA SCIENCE AND MACHINE LEARNING
ANDY VICKLER
BOOK 3
PYTHON
PYTHON AUTOMATION TECHNIQUES AND WEB SCRAPING
ANDY VICKLER
BOOK 2
PYTHON
PYTHON BASICS FOR BEGINNERS
ANDY VICKLER
BOOK 1
ANDY VICKLER
PYTHON
3 BOOKS IN 1
PYTHON BASICS FOR BEGINNERS
PYTHON AUTOMATION TECHNIQUES AND WEB SCRAPING
PYTHON FOR DATA SCIENCE AND MACHINE LEARNING

PYTHON

Andy Vickler

Table of Contents

PYTHON
Python Automation Techniques and Web Scraping

PYTHON
Python for Data Science and Machine Learning

PYTHON
PYTHON BASICS FOR BEGINNERS

Andy Vickler

Introduction

This book on Python's basics contains proven steps and strategies on how to be master the basics of Python programming. I recommend that you keep your laptop open in front of you while reading this book. This book is more practical than theoretical. It will challenge you to try coding yourself. It is easy enough for any beginner to start coding right away.

I have given functional codes in the book and comprehensive explanations to avoid any difficulty when practicing the codes.

There is no prior knowledge of Python required to read the book. I started the book from the basics and took it over to object-oriented programming, which falls into the advanced level domain. I just touched the concept slightly to give you a flavor of how you can use Python in the practical world.

With a good grasp of the Python basics, you can automate simple tasks, which can help you with simple home or work chores. After mastering the basics, you can use them to build machine learning models. Different Python libraries like Keras and TensorFlow can help you build Artificial Intelligence and Machine Learning models. Both are in demand and are highly useful in solving real-life problems. For example, artificial intelligence models can power robotic machines, customer services centers, retail outlets, the health sector for diagnosing diseases, etc.

The applications of Python are endless. Artificial Intelligence systems tend to learn from the data that you feed to them. This is similar to the human brain. Machine learning models also work on data. Although they are not as efficient as artificial intelligence models, they also have wider applications. They are used in the manufacturing industry where human labor cannot work. They

can handle bigger robotic machines and wrap up complicated tasks in a simple and fast manner.

I hope you enjoy your codes and the time you will spend on debugging your codes as well. Please don’t forget to keep a notebook and a pen by your side. Best of luck to you for your reading ventures!

Chapter One

A Brief History of Python

Python is a programming language that is highly focused on bringing ease to the users and simplicity to the programmers at the beginner's level. It allows programmers to write the code in structural, object-oriented, and functional styles. It is presently widely used in several fields like machine learning and developing websites. After the highly famous and used JavaScript, Python is the second most used programming language worldwide.

The founder and creator of Python is Guido van Rossum. Until 2018, he played a key role in the development of Python, making decisions that would affect the changes and the updates to the programming language.

In 1989, he was working on a microkernel-based system called Amoeba. For that system, he developed certain system utilities. While he was working on them, he realized that program development in C used to consume too much time and energy. He decided that he would spend his free time building a dedicated language to help him finish his work quickly.

He hit upon an idea to script a language that would fall somewhere between the shell script and C in terms of functionality and nature. Simply put, he wanted an interpreted language. However, he also wanted it to be easily programmable and readable compared to shell scripts.

Therefore, he developed Python. As the popular belief might be, Python is in no way named after the poisonous snake species. It is named after a British comedy troupe, namely Monty Python.

The word Python also turned out to be catchy and eye-popping. Not only is it a bit edgy, but it also fulfills the tradition of choosing saucy names and naming after highly famous people.

Python's First Release

The language Python was first released at the institute at which Guido had been working. He agreed with the manager to publish Python as an open-source language.

In February 1991, Van Rossum published the source code for the interpreter of Python to alt.sources. Nowadays, almost all programming languages are open-source and can be easily traced on GitHub. At the point of the release, it was not clear what the business model of the people who were developing those languages would be.

There were different proprietary languages, but it was a bit difficult for them to gain popularity. Guido knew that the idea of open-sourcing was the only viable way to make Python a success.

Sharing the open-source code was never easy. Originally, the developers faced some difficulties. They had to split up the source code for the Python interpreter into 21 uuencoded messages for sharing with different newsgroups. The idea worked and proved to be a better replacement for carrying along with a physical version of the source code.

The first release of Python had classes, functions, exception handling, and different core datatypes like str, list, dict, etc. It also was heavily inspired by the ABC language that Guido implemented at CWI. While he was creating Python, his goal was to take up all the good parts of ABC and fix the rest of them. In January 1994, he released the 1.0 version, and a milestone in the history of Python was achieved.

At that time, the market also welcomed many other interpreted languages similar to Python, two of them being Ruby and Perl.

This shows that the market definitely needed an easily interpreted language.

The US National Institute for Standards and Technology invited Guido in 1994. NIST was interested in the use of Python for many standards-related projects. They also needed somebody to boost Python skills, so they decided to get Guido, the creator of Python, onboard.

With the support of NIST, Guido could run several workshops and would participate in different conferences. He managed to spread Python and attract certain key contributors that would play an important role in shaping the future of this new programming language.

This resulted in different job offers for Guido from CNRI that was a non-profit research laboratory. This specific position helped Guido gather a team of enthusiastic people who would later support him to release Python versions.

The Python language started shaping up nicely with perfect principles and priorities. A fine example of the principles of Python is Python's Zen, a certain set of aphorisms from a software engineer, namely Tom Peters. The Zen carries Python's core philosophy.

In 1996, the Python language was used to build different products such as Microsoft Server, a part of Windows NT. Python 2.0 that was released in October 2000, brought list comprehensions. List comprehensions are a common Python feature present in several programming languages such as Haskell. It also added many features like Unicode support and a garbage can. Python was gradually inching toward the future as a reliable language with comfortable development experience.

Python 3.0

Starting from the year 200, the core developers began thinking about releasing Python 3.0. They desired to streamline different languages like cutting unnecessary language functions and constructs that Python sought to accrue in around 20 years. Their efforts resulted in Python 3.0, a backward-incompatible version of the Python language that came out in 2008. However, the release of the language was not as smooth as it was thought to be. It was filled up with several complications.

The developers did not realize how much Python code depended on the libraries of Python. So, while it was really easy to shift your scripts to Python 3, it was harder to move the programs that relied on the third-party libraries since they could not upgrade fast.

While Python was growing steadily, from 2010, it began on the trajectory of growth that enabled it to compete with top computer programming languages, such as JavaScript and Java.

With exponential growth in machine learning, a rise in the number of developers, and big data, the popularity of Python shot to the sky. The machine learning journey of Python got to the inflection point around September 2016 as per Google Trends, following the release of TensorFlow. This also is similar to the rise in the worldwide interest in the field of machine learning.

The Future

In the future of Python frameworks like Flask and Django, it is a great option for simple and quick web development. Python enjoys top support for machine learning programs across different programming languages because of its heavyweight libraries such as Keras and TensorFlow. The easy syntax makes it the best programming language of choice for data scientists and ML experts.

Python is equipped with some wonderful tools to do data analysis and visualization, making it the best choice at different

points in the data pipeline. One of the best things you should know about Python is that it enables specialists in different domains to start quick programming. The popularity of Python is gradually growing because it is highly versatile and applicable to different fields that are boosted with the help of automation, big data, and machine learning. This makes Python one of the most popular programming languages.

How to Install Python

Python is considered the most widely used programming language. It has gained popularity and is also considered one of the most flexible and highly popular server-side programming languages. Unlike most of the distributions of Linux, Windows does not have any pre-installed version of Python. However, you can install it on your local machine or Windows server in a few easy steps.

Installing Python Windows 10

There are some prerequisites to installing Python on your system. The first is a system that has Windows 10 installed on it. You also should have admin privileges. The second prerequisite is the command prompt that comes with Windows by default. The third prerequisite is a remote desktop app. You can use it if you are about to install Python on a remote Windows server.

Step 1

The first step is downloading the official Python .exe installer. After you have installed it, you need to run it on your system. The version depends on what you need to do with Python. If you have been working on a certain project coded in Python version 2.6, you might need this version. If you are just starting a project from scratch, you have the freedom to choose.

I recommend that if you are learning coding in Python, you should look out for the latest version of Python to enjoy the latest features. However, if you have any older test projects stored on

your system, you may opt for downloading Python 2 to have backward compatibility for those older projects. Python offers Remote Desktop Protocol (RDP) as well. Once you log into the remote server, you have to follow the same installation process to install it on a local Windows system.

Step 2

To complete the second step, open up your web browser. Go to python.org. Once you are on the website, find the Download Python for Windows button. You will find multiple versions of Python on the website. Download the one that suits you. You can either download the x86 or 64 or the x86 installer. Choose and click.

If you are running 32-bit Windows, you should download the x86 installer. Otherwise, download the x-86-64 installer. If, by mistake, you install the wrong version, uninstall it and then reinstall the right version.

Step 3

Now that you have downloaded the installer, you should run it. It will install Python on your system. Make sure to add Python to the path and enable Python for all users on the system. Adding Python to the path allows the interpreter to access the execution path.

Now select *Install Now* .

For all the versions of Python, the options for recommended installation include IDLE and Pip. The older versions may not include any additional features.

The next dialog box will ask you to disable the path length limit. Disabling it will help Python use lengthy path names. The disable path length limit option does not affect the other system settings. If you turn it on, it will resolve the issues regarding name length that pop up in Python projects developed in Linux.

When you have installed Python, you can go to the Start Menu on your Windows or click on the Windows sign to search where Python is located. Find it and run Python IDLE. I have got Python 3 on my computer system. When I run it for the first time, here is what I see on the screen.

```
Python 3.8.5 (tags/v3.8.5:580fbb0, Jul 20 2020, 15:57:54) [MSC
v.1924 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more
information.
>>>
```

Step 4

Now you need to verify whether Python is successfully and properly installed on Windows or not. You need to navigate to the directory where you have installed Python on your system. Double-click python.exe. You will see the installed version of Python. You also can check if the installation had been successful by typing −V in the Command Prompt.

Step 5

The next step is to verify whether Pip was installed on the system or not. If you opted to install the older version of Python, it might not come with a preinstalled Pip. Pip is an extremely powerful package management system to create software packages in Python, so make sure it is installed. I recommend that you use Pip for different Python packages when you are working inside certain virtual environments.

For verification of Pip, you need to do the following:

1. First of all, you need to open the Start menu and enter ‘cmd.’
2. You need to open the Command Prompt application.
3. Enter pip −V inside the console. The output will verify whether Pip was installed on your system or not.

Step 6

You need to open the Start menu and then initiate the Run app. Here, type sysdm.cpl and then click OK. This will open the System Properties window. Now navigate to the Advanced lab and then select the Path variable. Now click Edit. Here you need to select the Variable value field. At this point, you can add path to the python.exe file that is preceded with a semicolon(;).

Click Ok and shut down all the windows. When you have set it up, you can execute certain Python scripts such as Python script.py. This script is cleaner and can be easily managed.

Step 7

Now you have installed Python and Pip to manage different packages, you only need one final software package - virtualnv. It will enable you to create certain isolated local virtual environments for several Python projects.

Python software packages come pre-installed by default. As a result, whenever any specific package faces change, it tends to change for all the Python projects. You may need to avoid this. Having separate virtual environments for different projects probably is the easiest solution you can have.

For installing virtualnv, you need to open the Start menu and type “cmd.” Then you need to select the Command Prompt application. After that, you should type the pip command inside the console. When it is completed, virtualnv is installed on your system.

Installing Python on Mac OS

The recommended approach on macOS is to install Python using the official installer at Python.org. Previously, Homebrew's package manager was the recommended option because it made installations and updates easy in most cases. However, it doesn't work too well anymore for Python, so Homebrew is the best option.

Step 1

Opening your browser, go to www.python.org/downloads and click the button to download the latest version of Python.

Step 2

A popup message will ask if you want to allow the site to download; click Yes. Then open a Finder window and click Downloads on the sidebar. Double-click the Python package to start the installation.

The Python Installer opens, click the Continue button.

On the Read Me page, click on Continue and then click Continue on the Licence page. This will bring up another popup, asking you to agree to the terms and conditions. Click on Agree.

Click Install on the next screen, and Python will be saved to your hard drive.

Enter the password when asked for it and click on Install Software.

Click Close on the Summary window, and a new popup will ask you if you want the installer placed in your trash bin. Click Move to Trash because you don't need it anymore.

A new Finder window will also open, showing you the Python package you just installed.

Click the link for IDLE, as this is the easiest way to use Python. A new Shell opens where you can type your commands. You can also use the command line to access Python. Open Applications>Utilities>Terminal and type python3 – the interpreter will open, and if you see the command prompt - >>> - it is working properly.

Installing Python on Linux

Most of today's Linux distributions have Python already installed by default. To check your current version, open the terminal and run the command below. Do not include the $ (dollar sign) as this is the command prompt:

```
$ python --version
Python 3.9.5
```

If your version comes up as 2.xxx, repeat the above command using python3:

```
$ python3 --version
Python 3.9.5
```

At the time of writing, v3.9 is the most up-to-date Python version, but anything from 3.6 onwards will be good enough.

Linux provides a lot of different installation options, but the easiest one to use is called deadsnakes. Open your terminal and enter these commands:

```
$ sudo add-apt-repository ppa:deadsnakes/ppa
$ sudo apt-get update
$ sudo apt install python3.9
```

Python 3.9 will be installed, and you can start a new session by typing the following at the command prompt:

```
$ python3.9
>>>
```

Troubleshooting Installation Problems

I hope that your installation venture is successful on the first attempt. However, if you are unable to do that, you may need to fix some installation problems. Here are a few remedies you may try.

When a certain program contains a prominent error, Python will display a traceback error. Python will look through the file and then try to report the problem. The traceback might offer you a

clue as to what is wrong with the code and what prevents your program from running.

The best practice is to step away from your computer for a while. Take a break and try again. Syntax is extremely important in the world of programming, so even if there is a missing colon, mismatched parentheses, or a mismatched quotation mark, it will prevent your program from running properly. You should reread all the parts of the code, look at what you have done, and remove any errors.

Chapter Two

Python Datatypes

This chapter will walk you through various types of data that you can work with to create your Python programs. You will learn how you can store data inside different variables, and you will also learn how you can use these variables inside the programs. Variables can be defined as containers that are used to store different values. You can, when the need arises, use these variables to perform different tasks. Let's write the first code in Python. I will write a print statement that will display what I will write in the code.

```
print("I am learning Python from scratch.")
I am learning Python from scratch.
```

I wrote the code with the print command in Python editor. You can open this editor by clicking on the File in the menu line and then New. A new Editor window will pop up on your screen. You can start writing code here. The output that you can see after the demarcating line is displayed in the IDLE.

The editor runs the file through a Python interpreter or IDLE in simple words. The interpreter reads through the Python program and then determines what each word in the editor means. When the editor finds the word print, it prints whatever exists in the parentheses that follow the print keyword.

As you write programs, the editor will highlight different parts of the program in several ways. For example, it will recognize that the print keyword is the name of a Python function. That's why you will see that the word print will appear in blue. It will immediately display the statement in the parenthesis. The

coloring feature of Python is known as syntax highlighting. This feature is very useful, especially when you begin writing your programs.

Python Variables

In the following section, I will explain how you can use Python variables. You can add a new line at the start of the file and then modify the second line. See how I will do that.

```
mg1 = ("Python is an easy and interpreted language.")
print(mg1)
Python is an easy and interpreted language.
```

You can see that the output is the same as we had before. So, what is different here? I have added a variable named mg1 in the code. Every variable in Python holds a specific value. This is the information that is associated with the same variable. In this particular case, the value is a statement.

The addition of a variable causes a bit more work for the Python interpreter. When the interpreter processes the first line of code, it tends to associate the text with the message of the variable. When it gets to the second line, it usually prints the value associated with the message toward your computer screen. Let's expand this program by adding more lines of code.

```
mg1 = ("Python is an easy and interpreted language.")
print(mg1)

mg1 = ("You can learn Python easily by practice.")
print(mg1)

Python is an easy and interpreted language.
You can learn Python easily by practice.
```

This means that you can always change the value of the variable in your program at any time. Also, Python keeps track of every change in the current value of the variable.

Naming Variables

When you are working on variables in Python, you have to adhere to several guidelines and rules. Breaking these rules will definitely cause certain errors. Different guidelines will help you write code that's easier to understand and read. However, you need to be sure that you follow a set of rules when naming them.

The first rule is that the names of variables should only contain numbers, letters, and underscores. They may start with an underscore and a letter but never with a number. For example, you can write mg1 but not 1mg. You cannot add spaces to the names of variables, but you can use underscores to create separation between words in the names of variables. For example, you can write mg_morning, but you cannot write mg morning. It will trigger an immediate error.

You should also avoid using the reserved keywords of Python and the names of functions to name your variables. For example, you cannot use print, a Python keyword, as a name for a variable.

The names of variables need to be short but highly descriptive. For example, the name student is better than st. A descriptive name helps you understand the code better. You need to be extremely careful about not confusing lowercase l with 1 and uppercase O with 0. Never make this mistake, or your code will never run smoothly.

Just like coding itself, writing a standard variable name also takes some practice. You need to keep the rules and traditions in mind before starting to produce standard names when you practice.

Python variables that you may be using at the moment might be lowercase. If you write uppercase names, you will not see any error. Still, it is a good idea to write names in lower case letters.

If you write a variable name and make a mistake, don't worry because it is normal. Every programmer takes time to grasp the traditions of naming variables. Before they become a naming

expert, they make a lot of mistakes. The point is that you must know how to respond to the errors that keep popping up from time to time.

The first error you may make when writing a variable's name is writing the wrong spelling. In the following example, I will write wrong spellings and see what kind of error I will have. See how it is done.

```
mg1 = ("Python is an easy and interpreted language.")
print(msg1)
```

Traceback (most recent call last):

```
  File "C:/Users/saifia computers/Desktop/python.py", line 2, in
<module>
   print(msg1)
NameError: name 'msg1' is not defined
>>>
```

The output says that you have incurred an error. It clearly returns that the name msg1 is not defined. By this phrase, you should understand that Python cannot define a variable because it cannot match its name with any in the code. You might have written the wrong name, or the variable does not exist at all. This is a name error. I added an additional letter ‘s’ to the variable's name, which triggered an error. The Python interpreter usually does not spellcheck the code but ensures that the variables' names must be spelled consistently.

Computers disregard good and bad spelling. They only know the spelling that you use for the names of the variables. As a result, you need not consider the grammar rules or English spelling when creating the names of variables or writing the code. Several programming errors are quite simple. They are like single-character typos. Many experienced programmers have to spend hours hunting down these tiny errors to run their code successfully. The best method to understand the new

programming concepts is to attempt to use the same when you write your own programs.

Python Datatypes

In the world of programming, the data type is a very important concept. Different data types do different things and Python has different data types to help programmers achieve their programming objectives. Let us explore different data types in the following example.

```
>>> #This is Python string
>>> a = str (" I am learning Python from scratch.")
>>> print(a)
I am learning Python from scratch.
>>> #This is Python int datatype
>>> a = int(40)
>>> print(a)
40
>>> #This is Python float
>>> a = float(30.777)
>>> print(a)
30.777
#This is Python list
>>> a = list(("Great expectations", "God of flies", "God of small
things", "Tempest", "Othello"))
>>> print(a)
['Great expectations', 'God of flies', 'God of small things', 'Tempest',
'Othello']
>>> #This is Python tuple
>>> a = tuple(("Great expectations", "God of flies", "God of small
things", "Tempest", "Othello"))
>>> print(a)
('Great expectations', 'God of flies', 'God of small things', 'Tempest',
'Othello')
>>> #This is Python range
>>> a = range(7)
>>> print(a)
range(0, 7)
>>> #This is Python set
```

```
>>> a = set (("Great expectations", "God of flies", "God of small
things", "Tempest", "Othello"))
>>> print(a)
{'Othello', 'Tempest', 'God of small things', 'Great expectations',
'God of flies'}
>>> #This is Python frozenset
>>> a = frozenset(("Great expectations", "God of flies", "God of
small things", "Tempest", "Othello"))
>>> print(a)
frozenset({'Othello', 'Tempest', 'God of small things', 'Great
expectations', 'God of flies'})
```

Strings

Most programs gather data or define something. Then they use the same data to produce something useful. It helps in classifying various types of data. The first data type that I will dig into in this section is strings. Strings are simple if you take a look at them. However, when you start using them, they may feel more complicated than you had initially thought.

A simple string usually is a bundle of characters. Anything that is inside the quotes is considered a string. You can either use single or double quotes around the strings like the following.

```
“Python is giving me a tough time to learn.”
‘Python is giving me a tough time to learn.’
```

This is how you can integrate apostrophes into your code without risking errors to the code. See the following example.

```
‘I said to my father, “You will fail if you start losing friends.”’
“The programming language ‘Python’ is the best computer
language I ever got to learn.”
“My uncle’s best habit is that he never stops learning and gaining
new knowledge.”
```

So, this is how you can use the single and double quotes to your benefit. You can use different string methods to perform some amazing things. See how you can change the case of letters through string methods.

```
mg1 = ("Python is an interpreted language.")
print(mg1.title())

Python Is An Interpreted Language.
```

In this example, I have taken a lowercase string that was stored in a variable. The method title() that I have included in the code after the print() statement will convert the case of the letters. A method is generally any action that Python performs on a given set of data. The dot (.) that comes after the variable's name in the method tells Python that the title() method acts on the variable's name. Parentheses follow each method in Python because methods often demanded additional information to perform their work. That piece of information is filled up inside the parentheses. The title() function does not require additional information so, the parentheses are generally empty.

The method title() displays all the words in the title case, where each word starts with a capital letter. This is highly useful because you may need to think of a name as an additional piece of information.

Many other useful methods are available to deal with cases. You can switch the string to all lowercase or all uppercase letters such as the following. I will use the same string to apply the uppercase and lowercase methods. See the following example.

```
mg1 = ("Python is an interpreted language.")
print(mg1.upper())
print(mg1.lower())


PYTHON IS AN INTERPRETED LANGUAGE.

python is an interpreted language.
```

The lowercase method is highly useful for storing sets of data. You won’t trust the capitalization that the users provide often, so you will have to convert the strings into lowercase before you can

store them in the database. When you display information, you will have to use the case that makes the best sense for a string.

String Concatenation

It is highly useful if you start combining strings. For example, you may want to store the first and last name in separate variables. Then you will have to combine them when you have to display the full name of a person.

```
f_name = "John"
l_name = "Wick"
complete_name = f_name + " " + l_name + " " + "is the best action movie."

print(complete_name)
```

```
John Wick is the best action movie.
```

Python uses the plus symbol for combining two or more strings. In the above example, I have used the + operator for creating a complete name. The most important thing in this regard is the inclusion of space that you can find between two strings. If we exclude the space between the strings, we will have joined strings that are neither good-looking nor comprehensible.

The method of joining two or more strings through the plus operator is called concatenation. You can use this method for composing messages as I did in the example. Suppose you have a program that demands the first and last name of the users and return greetings when they have given their information to the system.

```
f_name = "John"
l_name = "Wick"
complete_name = "Hello, " + f_name + " " + l_name + "! How are you doing today?"

print(complete_name)
```

```
Hello, John Wick! How are you doing today?
```

Whitespaces

In programming, whitespace means any nonprinting character like tabs, spaces, and end-of-line symbols. You may use whitespace for organizing the output so that it is quite easier for the users to understand.

Let's see how you can add tabs to a piece of text. Here is the character combination \t.

```
print("Python")
print ("\tPython")

Python
Python
```

If you want to add a newline to the string, you can use the character \n.

```
print ("I can play the following
games:\nCricket\nFootball\nSoccer\nBasketball")
```

I can play the following games:

```
Cricket
Football
Soccer
Basketball
```

You can combine different newlines and tabs into a single string. The string \"\n\" instructs Python to shift to a new line and then start the next line with the help of a new tab. The following example will show how you can use the one-line string for generating different types of output.

Stripping Whitespace

While whitespace is desirable and highly useful in Python programs, it can be confusing in some programs. In programming, the words 'python' and 'python ' may appear to be

the same to the programmers but are not to the programs. They are two completely different strings. Python immediately detects the extra space in the second one and considers it of significance unless you instruct it otherwise.

You must consider whitespace as an important section of Python programming because you may come across situations in which you may have to compare two strings to determine if they are inherently the same or not. One important instance may involve checking the usernames of the people when they login to your website. Extra whitespace may turn out to be confusing in simpler situations. Fortunately, Python eases off the elimination of extraneous whitespace from the data that people fill into the program.

Python looks out for extra whitespace to the left and right sides of the string. In order to ensure that there is no whitespace at the right end of the string, you can use the rstrip() method in the code.

Syntax Errors

One most common type of error that you may see in the Python codes with a hint of regularity is a syntax error. A syntax error happens when Python fails to recognize a particular section of a program as the valid Python code. For example, if you use an apostrophe inside the single quotes, you may strike an error. This happens because Python generally interprets everything between the first single quote and apostrophe in the form of a string. After that, it attempts to interpret the remaining piece of text as the Python code. This results in errors.

Here is how you can use single and double quotation marks accurately.

```
best_movie = "Lewis Carrol's Alice In Wonderland is the best
fantasy movie."
print(best_movie)
```

```
Lewis Carrol's Alice In Wonderland is the best fantasy movie.
```

The above program has an apostrophe inside the quotation marks, but there is no problem running them because I have put the apostrophe inside double quotation marks. However, if you use single quotes, Python will not identify at what point the string will end.

```
best_movie = 'Lewis Carrol's Alice In Wonderland is the best fantasy movie.'
print(best_movie)
```

However, if I run the above-mentioned code, it will trigger a syntax error. This syntax error indicates that the Python interpreter does not recognize the code. Syntax errors are the least specific type of error, which is why they can be hard and frustrating to correct if you are stuck on a stubborn error.

The highlighting feature of the editor will help you find out a bunch of syntax errors fast as you write your programs.

Numbers

You can use numbers quite often when you are programming to add scores to different games. You can also use numbers to represent sets of data in visualizations and store information in different web applications. Python treats different sets of numbers in several ways, depending on how they are used. Let us take a look at how Python handles integers. Integers are the simplest form of numbers in Python.

Integers

You can perform different functions with integers. Here is a summary of them.

```
>>> #Adding integers
>>> 56 + 4
60
>>> #Subtracting numbers
>>> 345 - 45
```

```
300
>>> #Subtracting bigger number from smaller number
>>> 345 - 445
-100
>>> # Multiplying numbers
>>> 34 * 45
1530
>>> # Dividing Numbers
>>> 555 / 5
111.0
>>> #Double multiplication
>>> 5 ** 5
3125
>>> 3 ** 2
9
>>> 11 ** 5
161051
>>>
```

Python calls numbers that have a decimal point a float. The term is generally used in several programming languages. A decimal point may appear at a given position inside a number. Each programming language needs to be carefully designed to manage the numbers with points so the numbers behave appropriately regardless of the point the decimal point appears.

For the most part, you may use decimals without worrying about how they tend to behave. You may simply enter all the numbers that you want to use. Python will do what you expect from it.

```
>>> 1.5 + 1.6
3.1
>>> 555.5 - 55.9
499.6
>>> 34.56 * 45.5
1572.48
>>> 34 * 0.8
27.200000000000003
>>> 345 / 5.6
61.60714285714286
```

```
>>> 345.5 / 54.6
6.327838827838828
>>>
```

When you are working with datatypes, you are most likely to cause unexpected errors due to the variation in the types of data. For example, you may want to use the value of a variable inside a message. If you are telling someone about the sequel number of the movie, you may need to write the code like the following.

```
sequel = 1
best_movie = "Lewis Carrol's Alice In Wonderland novel has " +
sequel + "sequel as well."
print(best_movie)
Traceback (most recent call last):
  File "C:/Users/saifia computers/Desktop/python.py", line 2, in
<module>
    best_movie = "Lewis Carrol's Alice In Wonderland novel has " +
sequel + "sequel as well."
TypeError: can only concatenate str (not "int") to str
>>>
```

You can see that the interpreter returned a type error. It means that Python cannot recognize what kind of information you might be using. In the above-mentioned example, Python sees that you have been using a variable that comes with an integer value (int). However, Python does not know how to interpret that value. When you use integers inside strings in this way, you have to specify that Python should use this integer as a string. You can achieve this goal by wrapping up the variable inside the str() function, which instructs Python to represent the non-string values in the form of strings. Here is the right way to write this code.

```
sequel = 1
best_movie = "Lewis Carrol's Alice In Wonderland novel has " +
str(sequel) + " sequel as well."
print(best_movie)

Lewis Carrol's Alice In Wonderland novel has 1 sequel as well.
```

Working with numbers is the most straightforward thing. However, if you start getting unexpected results, you need to check whether Python interprets the numbers the way you want it to.

The Zen of Python

For many years, the Perl programming language was considered the mainstay of the Internet. Most of the interactive websites in the early days had Perl as a power pack. The moto of the Perl community was to find more than one way to do a task. People loved the approach for some time because the flexibility of the language made it possible to solve complex problems. Ultimately, people realized that the focus on the flexibility factor made it hard to maintain some big projects over a considerably longer period of time. It was tedious, difficult, and extremely time-consuming to review the code and figure out what others had been thinking when solving complex problems.

Experienced programmers of Python would ask you to avoid complexity and use a simple approach wherever possible. The philosophy of Python's community can be traced in the Zen of Python. You can find it out by writing the following code and importing the document.

```
import this

The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
```

The document is long. I have just included a few sentences to show you what you will find after running the code in Python editor. Python programmers like the code to be beautiful. It must

look elegant. Programmers solve different problems that people face, and they respect efficient, well-designed, and beautiful solutions to different problems. As you start learning more about Python and use it to write more code, the on-looker should praise the beauty of the code.

If you have a choice between a simple and complex solution, try using a simple solution. Even if your code is complex, you should make a conscious effort to make it readable and elegant. When creating a project that involves complex codes, you need to focus on including informative comments to the code.

Your solutions should be compatible and creative. Most programming consists of small and common approaches to fairly simple situations for creating a larger and creative project.

Chapter Three

Python Lists

This chapter will walk you through Python lists. I will explain what lists are and how you can start working with different elements in the list. Python lists allow you to store different sets of information in a centralized place, whether you have one item or one million to deal with. Lists are considered as one of Python's most robust features. They have the power to tie together several important concepts into programming.

Defining Lists

A Python list is, in simple words, a collection of different items that are organized in a specific order. You can create a list that may include different alphabet letters, a set of digits from 0-9, and the names of the people in the family. You can put anything that you want on the list. The items in your list don't have to be related to one another. It is the best way to make your list plural, including letters, names, or digits.

In Python, you can use square brackets to indicate a list. Individual elements in a list are usually separated with the help of commas. See the following example of a list that consists of different types of guns.

```
guns = ['shotgun', 'pistol', 'revolver', 'bazooka', 'machine gun']
print(guns)
['shotgun', 'pistol', 'revolver', 'bazooka', 'machine gun']
```

Accessing Elements

Lists are highly ordered collections. You can access elements in a list by instructing Python about the position or index of the

desired item. You should write the name of the list and then the item's index number inside square brackets. See how I access elements from the list that I have just produced.

```
guns = ['shotgun', 'pistol', 'revolver', 'bazooka', 'machine gun']
print(guns)
print(guns[0])
print(guns[2])
print(guns[4])

['shotgun', 'pistol', 'revolver', 'bazooka', 'machine gun']
shotgun
revolver
machine gun
```

This is the result you will have. You will see a neat and clean formatted output on the interpreter screen. You also can use the string methods that I have already explained in the datatype section to format the output furthermore. Here is what happens when you enter the index number that never exists, like an out-of-range index number.

```
guns = ['shotgun', 'pistol', 'revolver', 'bazooka', 'machine gun']
print(guns[8])
```

Traceback (most recent call last):

```
  File "C:/Users/saifia computers/Desktop/python.py", line 2, in
<module>
    print(guns[8])
IndexError: list index out of range
```

Python clearly tells you that you have made an index error while writing the code. You also can integrate different string methods in the code like the title(), lower(), or upper() method to alter the formatting of the output.

```
guns = ['shotgun', 'pistol', 'revolver', 'bazooka', 'machine gun']
print(guns[0].title())
print(guns[1].lower())
print(guns[2].upper())
```

```
print(guns[3].title())
print(guns[4].upper())

Shotgun
pistol
REVOLVER
Bazooka
MACHINE GUN
```

Another important thing is that Python considers the first item as index 0, not index 1. If you make this mistake and consider it as index 1, the index of the entire list will be affected. Most programming languages have indices that start at 0. Python also offers negative indexing. You can display items from the end of the list by using negative indexing. Negative indexing is different from positive indexing as it starts with -1 and not 0. See how you can use it.

```
guns = ['shotgun', 'pistol', 'revolver', 'bazooka', 'machine gun']
print(guns[-1])
print(guns[-2])
print(guns[-3])
print(guns[-4])
print(guns[-5])

machine gun
bazooka
revolver
pistol
shotgun
```

This syntax is highly useful because sometimes, you need to access the items at the end of the list without knowing the length of the list. As you can see, the first index returns the first item from the end of the list, the second index returns the second item from the end of the list, and the third index returns the third item from the list.

You also can use individual values from the list, just like you would do with other variables. You can use concatenation for

creating messages based on the value from the list. Let us see how to use a value from the list and create a message through the same works.

```
guns = ['shotgun', 'pistol', 'revolver', 'bazooka', 'machine gun']
mg1 = "I am going to use " + guns[2].title() + " in my next battle."
print(mg1)
```

```
I am going to use Revolver in my next battle.
```

You can also use multiple values in the same sentence by the following method.

```
guns = ['shotgun', 'pistol', 'revolver', 'bazooka', 'machine gun']
mg1 = "I am going to use " + guns[2].title() + " in my next battle."
print(mg1)
mg1 = "I am going to use " + guns[4].title() + " in my next battle."
print(mg1)
mg1 = "I am going to use " + guns[0].title() + " in my next battle."
print(mg1)
```

```
I am going to use Revolver in my next battle.
I am going to use Machine Gun in my next battle.
I am going to use Shotgun in my next battle.
```

List Modification

Most lists are dynamic lists, so you can build a list and then add or remove different elements from the same as your program runs. For example, if a player is in the game, he is using guns to fight battles. You can store the initial number and type of guns in a list. Once the player empties the guns, you can remove them from the list. Each time the player finds a new gun off a dead soldier, you can add the same to the list. The list of your guns will keep decreasing and increasing throughout the game. Similarly, you can create other games that need addition and removal to lists constantly.

You can modify a list element in the same way you can access elements from a list. If you seek to change an element, you need

to use the name of the list and the index number of the item that should be changed. After that, you should fill in the new value that you want that item to have.

I will use the list of guns that I have created earlier on and modify different items in the list.

```
guns = ['shotgun', 'pistol', 'revolver', 'bazooka', 'machine gun']
print(guns)

guns[0] = 'Ak-47'
print(guns)

guns[2] = 'knife'
print(guns)

['shotgun', 'pistol', 'revolver', 'bazooka', 'machine gun']
['Ak-47', 'pistol', 'revolver', 'bazooka', 'machine gun']
['Ak-47', 'pistol', 'knife', 'bazooka', 'machine gun']
```

I defined the list at the first line of code. Then I changed the values at index 0 and index 2, displaying the modified output accordingly. That's how you can change the output of any item in the list.

Adding elements to Python lists is also quite easy. You might need to add new elements to a specific list for different reasons. As I have explained earlier, if you are developing a game with a player who fights a battle, you need to know how to add elements to allow the player to pick up or buy different guns.

Append

The simplest form of addition of elements to lists in Python is through the append method. When you try to append an item to a Python list, the new element is immediately added to the tail of the list. I will use the same list and try to add new elements to it through the append method.

```
guns = ['shotgun', 'pistol', 'revolver', 'bazooka', 'machine gun']
print(guns)
```

```
guns.append('Ak-47')
print(guns)

guns.append('sten gun')
print(guns)
```

```
['shotgun', 'pistol', 'revolver', 'bazooka', 'machine gun']
['shotgun', 'pistol', 'revolver', 'bazooka', 'machine gun', 'Ak-47']
['shotgun', 'pistol', 'revolver', 'bazooka', 'machine gun', 'Ak-47',
'sten gun']
```

What is remarkable about the append method is that it adds different items to the end of the list without producing any changes in the order of the elements in the list. This makes the append method the most favorite method to build lists dynamically. You can take an empty list and add items to it gradually. This will take a series of append statements. In the following example, I will create an empty list and add elements to the same through the append() method.

```
guns = []
guns.append('Ak-47')
guns.append('sten gun')
guns.append('shotgun')
guns.append('pistol')
guns.append('revolver')
guns.append('bazooka')
guns.append('machine gun')
print(guns)
```

```
['Ak-47', 'sten gun', 'shotgun', 'pistol', 'revolver', 'bazooka',
'machine gun']
```

Building lists by this method is quite common because you will never know what data your users need to store in a specific program until after the start of the program. By the append() method, you can put your users in complete control. Just start by defining the list that is going to hold the values of users. After that, they can go on and fill up the list.

Another method to perform addition is the insert() method. You can add as many new items to the list through the insert() method as you need. All you need is to specify the index number of the new element and then the value of the new item.

```
guns = ['shotgun', 'pistol', 'revolver', 'bazooka', 'machine gun']
print(guns)
guns.insert(0, 'Ak-47')
print(guns)
guns.insert(2, 'sten gun')
print(guns)
```

```
['shotgun', 'pistol', 'revolver', 'bazooka', 'machine gun']
['Ak-47', 'shotgun', 'pistol', 'revolver', 'bazooka', 'machine gun']
['Ak-47', 'shotgun', 'sten gun', 'pistol', 'revolver', 'bazooka',
'machine gun']
```

You can see that I added different items at the locations I wanted to by using the insert() method. The insert() method pops open space at positions 0 and 2 to add elements that I proposed. This operation shifts the position of other values in the list to the right of the list.

Removing Elements

You also can remove elements from the list. For example, when your player empties a gun and wants to get rid of it, you need to have a method that removes that item from the list.

If you know exactly what position the item holds in the list, you can use the del statement and use the index number to slice out the element from the list. This helps create a dynamic game where the player can see the list and throw away a specific gun from his list of guns.

```
guns = ['shotgun', 'pistol', 'revolver', 'bazooka', 'machine gun']
print(guns)

del guns[0]
print(guns)
```

```
del guns[2]
print(guns)

['shotgun', 'pistol', 'revolver', 'bazooka', 'machine gun']
['pistol', 'revolver', 'bazooka', 'machine gun']
['pistol', 'revolver', 'machine gun']
```

You can see that you can remove items from any positions in the list. However, what is remarkable about the del() method is that you can no longer access the deleted values in any way.

Another method to remove elements from the list is the pop() method. You might want to use the elements once you have removed them from the list in a game. For example, you may want to allow the players to see the list of guns that have been emptied and thrown away. At this point, you need to use the pop() method to achieve the objective that you want to.

The pop() method removes items from the end of the list. It also lets you work with popped-out items. The term pop is derived from the idea of comparing lists with stacks of different items and popping those items off the top. In this analogy, the top of the stack is akin to the end of the list.

```
guns = ['shotgun', 'pistol', 'revolver', 'bazooka', 'machine gun']
print(guns)

popped_guns = guns.pop()
print(guns)
print(popped_guns)

popped_guns = guns.pop()
print(guns)
print(popped_guns)

popped_guns = guns.pop()
print(guns)
print(popped_guns)

['shotgun', 'pistol', 'revolver', 'bazooka', 'machine gun']
```

```
['shotgun', 'pistol', 'revolver', 'bazooka']
machine gun
['shotgun', 'pistol', 'revolver']
bazooka
['shotgun', 'pistol']
revolver
```

You can see that not only was I able to delete items from the list, but I was also able to use the deleted items to display them in the interpreter.

I defined the list and printed the items in the list. Then I popped one value and stored it in the variable popped_guns. Then I printed the revised list to show that the item is no longer on the list. After that, I printed the popped value to prove that I could still access it. I repeated the removal three times, and it worked perfectly.

Let us see how you can use the pop method in the game you have been developing to remove values. In the following example, I will show how you can use the value of the removed item. In the example, I will use the popped value in a statement to display a message to the player.

```
guns = ['shotgun', 'pistol', 'revolver', 'bazooka', 'machine gun']

popped_guns = guns.pop()
print("The last emptied gun was a " + popped_guns.title() + ".")

popped_guns = guns.pop()
print("The last emptied gun was a " + popped_guns.title() + ".")

popped_guns = guns.pop()
print("The last emptied gun was a " + popped_guns.title() + ".")

The last emptied gun was a Machine Gun.
The last emptied gun was a Bazooka.
The last emptied gun was a Revolver.
```

The output comes in the form of a simple sentence about the recent gun that the player owns.

You also can mention the index number from a designated position in the list. You can use the pop() method to remove different items from any positions in a list. See how it is done.

```
guns = ['shotgun', 'pistol', 'revolver', 'bazooka', 'machine gun']

popped_guns = guns.pop(0)
print("The " + popped_guns.title() + " is emptied and thrown
away.")

popped_guns = guns.pop(2)
print("The " + popped_guns.title() + " is emptied and thrown
away.")

The Shotgun is emptied and thrown away.
The Bazooka is emptied and thrown away.
```

I popped an item from the list and then used it to print a message that told the player about the gun that was emptied and thrown away. Each time you use the pop() method, the item you have been working with will no longer be stored inside the list.

If you cannot decide whether you have to use the pop() method or the del statement, you can decide based on the fact if you have to use the item later on or not.

Remove() Method

You may not know the exact position of a value at a point, which is why you have to use the value itself to remove it from the list. If you have got all the values in the list, you can use the remove() method. I will use values to delete different items from the list in the following example.

```
guns = ['shotgun', 'pistol', 'revolver', 'bazooka', 'machine gun']
print(guns)
```

```
guns.remove('revolver')
print(guns)

guns.remove('shotgun')
print(guns)

['shotgun', 'pistol', 'revolver', 'bazooka', 'machine gun']
['shotgun', 'pistol', 'bazooka', 'machine gun']
['pistol', 'bazooka', 'machine gun']
```

This method is considered one of the most efficient methods to remove items from a list. You can also use the remove() method to operate on a value removed from the list. I will take the removed value and use it in a list.

```
guns = ['shotgun', 'pistol', 'revolver', 'bazooka', 'machine gun']
print(guns)

emptied_gun = 'pistol'
guns.remove(emptied_gun)
print(guns)
print("\nThe " + emptied_gun.title() + " is empty and is being
thrown away.")

emptied_gun = 'bazooka'
guns.remove(emptied_gun)
print(guns)
print("\nThe " + emptied_gun.title() + " is empty and is being
thrown away.")

['shotgun', 'pistol', 'revolver', 'bazooka', 'machine gun']
['shotgun', 'revolver', 'bazooka', 'machine gun']

The Pistol is empty and is being thrown away.
['shotgun', 'revolver', 'machine gun']

The Bazooka is empty and is being thrown away.
```

After I had defined the list, I stored the value inside a variable, namely emptied_gun. Then I used the same value to instruct Python on how it should be used in a print statement. The

remove() method only deletes the first occurrence of the value that you have specified.

List Organization

Your lists, more often, will be created in a highly unpredictable order because you cannot control the order in which the users will enter data on the website. Although this may appear to be unavoidable in many circumstances, you may need to present the information in perfect order. Sometimes you may need to preserve the original order in which the list was first developed. At other times, you may need to change the order. Python provides multiple ways for organizing lists.

Permanent Method

The sort() method in Python makes it easy to sort out the list. I will use the same to explain how to use the sort() method. I am assuming that the values that are stored in the list are in lowercase.

```
guns = ['shotgun', 'pistol', 'revolver', 'bazooka', 'machine gun']
guns.sort()
print(guns)

['bazooka', 'machine gun', 'pistol', 'revolver', 'shotgun']
```

The order will come out in alphabetical order. The order has permanently changed. You cannot revert it to the original order. You also can sort the list in the reverse alphabetical order by passing the argument reverse=True to sort() method. In the following example, I will sort the lists in reverse alphabetical order.

Sorted() Function

To maintain the original order of the list but still keep it sorted, you may use the sorted() function. The sorted() function will let you display the list in a specific order, but it will not affect the original order.

```
fruits = ['orange', 'dragon fruit', 'jackfruit', 'strawberry', 'apple']
print("This is the original order:")
print(fruits)

print("\nThis list is sorted by the sorted() function.")
print(sorted(fruits))

print("\nThis again is the original list.")
print(fruits)
```

```
This is the original order:
['orange', 'dragon fruit', 'jackfruit', 'strawberry', 'apple']

This list is sorted by the sorted() function.
['apple', 'dragon fruit', 'jackfruit', 'orange', 'strawberry']

This again is the original list.
['orange', 'dragon fruit', 'jackfruit', 'strawberry', 'apple']
```

You can see that the list retains the original order. The sorted() function also accepts the reverse=True argument if you are looking to display a list that is in the reverse alphabetical order.

Sorting a list in alphabetical order is a complicated job if the values are not in lowercase. There are many ways to interpret capital letters when you have to sort the order of the list.

Reverse Order

If you want to reverse the order of a list, you have to use the reverse() method. It will reverse the chronological order of the list. It does not reorder the list in alphabetical order. It simply reverses the original order of the list. This change, too, is permanent and cannot be restored. But you can revert the order of the list by using the reverse() method the second time on the changed list.

```
fruits = ['orange', 'dragon fruit', 'jackfruit', 'strawberry', 'apple']
print(fruits)

fruits.reverse()
```

```
print(fruits)

['orange', 'dragon fruit', 'jackfruit', 'strawberry', 'apple']
['apple', 'strawberry', 'jackfruit', 'dragon fruit', 'orange']
```

If I apply the same reverse method to the changed list, I will have the originally ordered list back.

```
fruits = ['orange', 'dragon fruit', 'jackfruit', 'strawberry', 'apple']
print(fruits)

fruits.reverse()
print(fruits)

fruits.reverse()
print(fruits)

['orange', 'dragon fruit', 'jackfruit', 'strawberry', 'apple']
['apple', 'strawberry', 'jackfruit', 'dragon fruit', 'orange']
['orange', 'dragon fruit', 'jackfruit', 'strawberry', 'apple']
```

List Length

There is a dedicated function to find out the length of the list. The function is named as the len() function. See how it works.

```
>>> fruits = ['orange', 'dragon fruit', 'jackfruit', 'strawberry', 'apple']
>>> len(fruits)
5
```

This function is useful when identifying the total number of fruits or guns in the list. For example, in your game, if the player wants to know how many guns he has to play with, he can use the function to display the total number on the screen. This simple function will add more charm to your game or application. If you are building a grocery app, you can let the user count the number of fruits that he has added to the cart by this function.

Looping Through Lists

You may often need to run through the list entries while performing the same tasks with each item. For example, if you are building a game, you may need to move every element on your screen in a list of numbers, and you may need to perform the statistical operation on each element.

If you are developing a battle-oriented game, you may allow the user to display the list of guns on the screen to know what kind of guns he has before he jumps into the battle. This will also allow him to formulate an attack strategy for which gun is suitable to which point of the battle. A simple loop can help you make your game and application more dynamic and user-friendly. Python loops with lists are also used when you have to perform the same action with all the items in the list.

Let us say that we have a complete list of fruits, and we want to use all the items in the list. I can do this by retrieving all the names of the fruits from the list individually. However, this approach may cause several problems. Instead of writing separate codes for each item, I will use a loop to perform the same action repeatedly. A for loop avoids repetition in writing codes to perform an action on lists. I will now use the for loop to print all the names in the list.

```
fruits = ['orange', 'dragon fruit', 'jackfruit', 'strawberry', 'apple']
for fruit in fruits:
   print(fruit)

orange
dragon fruit
jackfruit
strawberry
apple
```

I defined the list in the first step, and then I created the for loop that would iterate through each item in the list and display it neatly in the output. The concept of looping in the world of Python is important because it is the most common way to

execute repetitive tasks. The first line that contains the for keyword tells Python to retrieve the first value in the list and then store it onwards into the variable, namely fruit. Then it reads the next line and prints the name of the value. After it prints the first value, it returns to the first line that has the for keyword because it finds out that the list has more values left on which it has to work.

You should keep in mind that the set of steps are repeated for each item of the list, no matter how many items exist in the list. If the list has a million items, Python will repeat all the steps a million times. Also, the speed of the execution of tasks is important. You can do many things in the loop. I will use the same list of fruits and print a message through the for loop.

```
fruits = ['orange', 'dragon fruit', 'jackfruit', 'strawberry', 'apple']
for fruit in fruits:
   print("The " + fruit.title() + " is ripe for the taking.")
```

```
The Orange is ripe for the taking.
The Dragon Fruit is ripe for the taking.
The Jackfruit is ripe for the taking.
The Strawberry is ripe for the taking.
The Apple is ripe for the taking.
```

In this code, I have added a message that included all items from the list. The first item to loop through is the value of the fruit Orange. The second is Dragon Fruit, and the last to display is Apple. I have added only five items to the list. You can add more items to test the ability of the loop and lists. In the output, you can see a personalized message.

You can write as many codes as you like in the for loop. In the next example, I will add two more lines of code to the same sample. These will be the second and third lines of code to display a broader message.

```
fruits = ['dragon fruit', 'jackfruit', 'strawberry']
for fruit in fruits:
```

```
    print("The " + fruit.title() + " is ripe for the taking.")
    print("I will start picking " + fruit.title() + " next Monday.")
    print("The packing of " + fruit.title() + " for export will start next
month.")

The Dragon Fruit is ripe for the taking.
I will start picking Dragon Fruit next Monday.
The packing of Dragon Fruit for export will start next month.
The Jackfruit is ripe for the taking.
I will start picking Jackfruit next Monday.
The packing of Jackfruit for export will start next month.
The Strawberry is ripe for the taking.
I will start picking Strawberry next Monday.
The packing of Strawberry for export will start next month.
```

As each print statement is indented, each code line must be executed once for all the fruits in the list. However, there still is one problem. There is no black line in between the lines of code. This is not neat and needs to change.

```
fruits = ['dragon fruit', 'jackfruit', 'strawberry']
for fruit in fruits:
    print("The " + fruit.title() + " is ripe for the taking.")
    print("I will start picking " + fruit.title() + " next Monday.")
    print("The packing of " + fruit.title() + " for export will start next
month.\n")

The Dragon Fruit is ripe for the taking.
I will start picking Dragon Fruit next Monday.
The packing of Dragon Fruit for export will start next month.

The Jackfruit is ripe for the taking.
I will start picking Jackfruit next Monday.
The packing of Jackfruit for export will start next month.

The Strawberry is ripe for the taking.
I will start picking Strawberry next Monday.
The packing of Strawberry for export will start next month.
```

Once the loop has ended, you can end it creatively by displaying a neat message to the users. You may want to summarize what has

happened in the loop. You can easily summarize the block of output or shift to the other work that the program needs to accomplish.

> The lines of code that come after the for loop that are no longer indented will be executed without any repetition. This makes indentation highly important. It can produce serious errors if you don't do it right.
> ```
> fruits = ['dragon fruit', 'jackfruit', 'strawberry']
> for fruit in fruits:
>    print("The " + fruit.title() + " is ripe for the taking.")
>    print("I will start picking " + fruit.title() + " next Monday.")
>    print("The packing of " + fruit.title() + " for export will start next month.\n")
>
> print("All the fruits need to be double-checked before export.")
> ```
>
> ```
> The Dragon Fruit is ripe for the taking.
> I will start picking Dragon Fruit next Monday.
> The packing of Dragon Fruit for export will start next month.
>
> The Jackfruit is ripe for the taking.
> I will start picking Jackfruit next Monday.
> The packing of Jackfruit for export will start next month.
>
> The Strawberry is ripe for the taking.
> I will start picking Strawberry next Monday.
> The packing of Strawberry for export will start next month.
>
> All the fruits need to be double-checked before export.
> ```

When you process data by using the for loop, you will find out that this is the best way to summarize a specific operation usually performed on a complete data set. For an app that deals with groceries, you can let the users go through all the fruits and vegetable items to see what is available to buy.

Indentation

Python is strict about indentation. It uses indentation to sense the connection between different lines of code. In the above-mentioned examples, the lines that printed messages about different fruits became part of the loop because of their indentation. Python's indentation makes code smoother to read. It basically uses whitespace to compel you to write code in a neatly formatted way with a clear structure. In lengthy Python programs, you may notice certain blocks of code that are indented at various levels.

Once you realize how to keep up proper indentation in your code, you will make fewer indentation errors. Sometimes, people indent blocks of code that are not supposed to be indented. Once you see these errors, you will immediately realize how to fix them.

```
>>> fruits = ['dragon fruit', 'jackfruit', 'strawberry']
>>> for fruit in fruits:
print("The " + fruit.title() + " is ripe for the taking.")
SyntaxError: expected an indented block
Just indent the last line of code to fix this error.
```

You may have indented the first line of code after the for statement, but you forgot to indent the subsequent lines. This kind of error is possible when you try to execute multiple tasks at the same time. Here is the result of a forgotten indentation for the second and third lines.

```
fruits = ['dragon fruit', 'jackfruit', 'strawberry']
for fruit in fruits:
   print("The " + fruit.title() + " is ripe for the taking.")
print("I will start picking " + fruit.title() + " next Monday.")
print("The packing of " + fruit.title() + " for export will start next
month.\n")

The Dragon Fruit is ripe for the taking.
The Jackfruit is ripe for the taking.
The Strawberry is ripe for the taking.
I will start picking Strawberry next Monday.
The packing of Strawberry for export will start next month.
```

The loop did not work on the second and third print statements. It only worked on the first value in the item and left out the result of them. So, those that are not indented will be automatically left out of the iterations of the loop. This is called a logical error. The syntax is a valid Python code and what is left in the code is logic. If you find out in your code that a certain action should have been repeated, but it is not, you need to determine whether there are some lines of code that should be simply indented or not.

Unnecessary Indentation

If you add an unnecessary indent, it will also produce an error.

```
>>> fruits = ['dragon fruit']
>>> print(fruits)
SyntaxError: unexpected indent
```

Post-Loop Unnecessary Indentation

If you have accidentally indented a code that you have written as a finisher after the loop, the code will also be repeated just like other codes do. It will be repeated for each item in the list. Sometimes, this will prompt Python to produce an error, but you will often see a logical error. Let us see how it behaves in Python 3.8.

```
fruits = ['dragon fruit', 'jackfruit', 'strawberry']
for fruit in fruits:
   print("The " + fruit.title() + " is ripe for the taking.")
   print("I will start picking " + fruit.title() + " next Monday.")
   print("The packing of " + fruit.title() + " for export will start next
month.\n")

   print("All the fruits need to be double-checked before export.")
```

```
The Dragon Fruit is ripe for the taking.
I will start picking Dragon Fruit next Monday.
The packing of Dragon Fruit for export will start next month.

All the fruits need to be double-checked before export.
The Jackfruit is ripe for the taking.
```

```
I will start picking Jackfruit next Monday.
The packing of Jackfruit for export will start next month.

All the fruits need to be double-checked before export.
The Strawberry is ripe for the taking.
I will start picking Strawberry next Monday.
The packing of Strawberry for export will start next month.

All the fruits need to be double-checked before export.
```

Instead of producing an error, Python 3.8 has repeated the statement for all the items in the list. This is known as the logical error. Python does not know what to do with a wrongful indentation in the code. It runs all the written code in the form of valid syntax. If the action is repeated several times when it needs to be executed once, look out for whether you need to unindent a portion of the code that is being unnecessarily repeated.

Numerical Lists

There are several reasons for storing different sets of numbers. You may need to keep track of the enemies in a battle game. You may need to point out the number of targets in the game. Lists are perfect for storing numbers, and Python offers a set of tools to help you work efficiently with numerical lists. Once you have understood how to use the tools correctly, your code will work perfectly when the lists you are working on contains millions of items.

The range() Function

The first function for a numerical list is the range() function, which makes it easy to generate different series of numbers. You can use this range() function for printing different series of numbers such as the following:

```
for num in range(1,7):
  print(num)

1
```

```
2
3
4
5
6
```

You can see that the range() function prints only numbers from 1 to 6. This is due to its off-by-one style that you will more often see in many programming languages. The range() function causes Python to initiate counting at the very first value. It stops when it finally reaches the second value that you provide to it. Because it seizes at the second value, the output will never contain the last value. If you want to include the digit 7 in the output, you need to extend the range to 8.

```
for num in range(1,8):
   print(num)

1
2
3
4
5
6
7
```

You can use the range() function to create a list of numbers. You can convert the result of the range() function directly into the list() function. When you wrap up list() around the range() function, the output is a list of numbers. In the next example, I will shift the result of the range() function into a list.

```
num = list(range(1,8))
print(num)

[1, 2, 3, 4, 5, 6, 7]
```

You also can use the range() function to tell Python what numbers to skip and what to include. See the demonstration in the following example.

```
even_num = list(range(4, 22, 4))
print(even_num)

[4, 8, 12, 16, 20]
```

You can use the range() function to produce the square of each integer from 1 to 22. Two asterisks ** in Python represent exponents. Here is how you can do that.

```
square_num = []
for num in range(1, 22):
   square = num**2
   square_num.append(square)

print(square_num)

[1, 4, 9, 16, 25, 36, 49, 64, 81, 100, 121, 144, 169, 196, 225, 256, 289,
324, 361, 400, 441]
```

I started the code with an empty list known as square_num. Then I told Python to initiate a loop through each value between 1 and 22 using the range function. The present value is raised to the second power in the loop and stored inside the variable square. When the loop completes its run, the list of squares comes out as printed.

List Slicing

You can produce slices of a list and use them at will. To create a slice of a list, you need to specify the index of the starting and ending elements in the list. Like the range() function, Python will stop one item short of the second index. If you need to include three elements, you have to set the index at 0 and 3. See how it works.

```
fruits = ['orange', 'dragon fruit', 'jackfruit', 'strawberry', 'apple']
print(fruits[0:3])

['orange', 'dragon fruit', 'jackfruit']
```

You can generate different subsets of the list.

```
fruits = ['orange', 'dragon fruit', 'jackfruit', 'strawberry', 'apple']
print(fruits[1:3])

['dragon fruit', 'jackfruit']
```

If you omit the first index number, Python will consider it to be the start of the list.

```
fruits = ['orange', 'dragon fruit', 'jackfruit', 'strawberry', 'apple']
print(fruits[:4])

['orange', 'dragon fruit', 'jackfruit', 'strawberry']
```

You can see that Python started the sliced list from the start. If you leave out the last index, it will run its course to the end of the list.

```
fruits = ['orange', 'dragon fruit', 'jackfruit', 'strawberry', 'apple']
print(fruits[1:])

['dragon fruit', 'jackfruit', 'strawberry', 'apple']
```

You can also use negative indexing to slice the lists.

```
fruits = ['orange', 'dragon fruit', 'jackfruit', 'strawberry', 'apple']
print(fruits[-4:])

['dragon fruit', 'jackfruit', 'strawberry', 'apple']
```

Looping Through Slices

You can use integrate the for loop in the code to create a loop through the slice that you produce.

```
fruits = ['dragon fruit', 'jackfruit', 'strawberry', 'banana', 'peach',
'guava']
for fruit in fruits[:5]:
   print("The " + fruit.title() + " is ripe for the taking.")
   print("I will start picking " + fruit.title() + " next Monday.")

The Dragon Fruit is ripe for the taking.
I will start picking Dragon Fruit next Monday.
The Jackfruit is ripe for the taking.
```

```
I will start picking Jackfruit next Monday.
The Strawberry is ripe for the taking.
I will start picking Strawberry next Monday.
The Banana is ripe for the taking.
I will start picking Banana next Monday.
The Peach is ripe for the taking.
I will start picking Peach next Monday.
```

Instead of looping through the entire list, Python will only loop through the elements present in the slice. Slicing can be highly useful in several situations. For example, when you have to create a game, you can add a player's final score to the list when a player finishes playing.

Copying Lists

You may often want to start the existing list and then create a completely new list based on the original one. Let us see how the copying function in a list works and then examine the situation in which copying the list remains useful. If you want to copy a list, create a slice that will include the original list. You have to omit the first and the last indexes like this ([:]). This will tell Python to create a slice that will start with the first item and ends with the last. Here is a list of books.

```
books = ['great expectations', 'god of flies', 'tempest', 'middle
march', 'bleak house', 'hard times']
favorite_books = books[:]

print("These are the books I have in my library:")
print(books)

print("These are my favorite books:")
print(favorite_books)
```

These are the books I have in my library:

```
['great expectations', 'god of flies', 'tempest', 'middle march',
'bleak house', 'hard times']
```

These are my favorite books:

```
['great expectations', 'god of flies', 'tempest', 'middle march',
'bleak house', 'hard times']
```

So, we have a perfect copy of the list that we created at the start. If you are still in doubt whether they are two separate lists or not, you can add a new book to each list.

```
books = ['great expectations', 'god of flies', 'tempest', 'middle
march', 'bleak house', 'hard times']
favorite_books = books[:]

books.append('othello')
favorite_books.append('heart of darkness')

print("These are the books I have in my library:")
print(books)

print("These are my favorite books:")
print(favorite_books)
```

These are the books I have in my library:

```
['great expectations', 'god of flies', 'tempest', 'middle march',
'bleak house', 'hard times', 'othello']
```

These are my favorite books:

```
['great expectations', 'god of flies', 'tempest', 'middle march',
'bleak house', 'hard times', 'heart of darkness']
```

First, I copied the original list into a new list and then added new items to each list. If you copy a list without a slice, here is what happens.

```
books = ['great expectations', 'god of flies', 'bleak house', 'hard
times']
favorite_books = books

books.append('othello')
favorite_books.append('heart of darkness')

print("These are the books I have in my library:")
```

```
print(books)

print("\nThese are my favorite books:")
print(favorite_books)
```

These are the books I have in my library:

```
['great expectations', 'god of flies', 'bleak house', 'hard times',
'othello', 'heart of darkness']
```

These are my favorite books:

```
['great expectations', 'god of flies', 'bleak house', 'hard times',
'othello', 'heart of darkness']
```

You can see that both items have been added to both lists. They behaved not like two but like one list.

Chapter Four

Python Tuples

Lists are useful to store different sets of items that change throughout a program. Their tendency to modify is highly important, especially when working with a certain list of users on a particular website or different characters inside a game. However, sometimes you may feel the desire to create a list that must not be changed. Tuples allow programmers to do exactly that. Python refers to different values that cannot be changed as immutable values. An immutable list is named a tuple.

> A tuple looks exactly like a list except that it has square brackets instead of parentheses to enclose the items. Once you have defined a tuple, you can access the elements inside it, but you cannot modify them in any sense. Python tuples are one of the four built-in data types in Python. The other three are lists, sets, and dictionaries, and all of these data types have different qualities and usage capacities. Tuples are enclosed inside round brackets. Here is how a tuple is created and how it looks like.
> mytuple = ("cars, bikes, motorbikes, buses, tanks, ships")
> print(mytuple)
>
> cars, bikes, motorbikes, buses, tanks, ships

The items in a tuple are in an ordered form. They are unchangeable, and they also allow duplicate values. The items in the tuple are indexed. The first item is indexed as [0]. This is how the number goes on.

When you say that tuples are in ordered form, this alludes to the fact that all the items inside a tuple fall into a defined order. That order does not change. You can add or remove any items from a tuple once you have created them.

Since tuples come in indexed form, you can add duplicate values inside a tuple. See the following example.

```
mytuple = ("cars, bikes, motorbikes, buses, tanks, ships, bikes")
print(mytuple)

cars, bikes, motorbikes, buses, tanks, ships, bikes
```

Length

You can calculate the length of a tuple by the following function.

```
mytuple = ("cars, bikes, motorbikes, buses, tanks, ships, bikes")
print(len(mytuple))
51
```

If you want to create a tuple with just one item, you need to add a comma after that one item. Otherwise, Python will never recognize it as a tuple.

This is how the result looks like if you don’t add a comma after the first item in the tuple.

```
mytuple = ("cars")
print(mytuple)

cars
```

Here is how the result will look like after you add a comma to the code.

```
mytuple = ("cars",)
print(mytuple)

('cars',)
```

Changing Elements

Let us see what happens when you try to change an item in the tuple.

```
mytuple = ("cars, bikes, motorbikes, buses, tanks, ships, bikes")
mytuple[0] = "jets"
```

Traceback (most recent call last):

```
  File "C:/Users/saifia computers/Desktop/python.py", line 2, in
<module>
   mytuple[0] = "jets"
```

TypeError: 'str' object does not support item assignment

When the code tries to change the value of the item at the first index, Python returns a type error. As you are trying to change a tuple, you cannot do that. That's why you run an error. Python tells you that you cannot assign a new value to the item in the tuple.

Creating Loop

You can build and run a loop through a tuple just like you did with lists. Let us see how to write the code for building a loop and running it through the tuple.

```
mytuples = ("cars", "bikes", "motorbikes", "buses", "tanks",
"ships", "bikes")
for mytuple in mytuples:
   print(mytuple)

cars
bikes
motorbikes
buses
tanks
ships
bikes
```

Overwriting a Tuple

You cannot modify the tuple, but you can overwrite it by assigning it new values. You can assign new values to the variables that carry a tuple. Therefore, if you need to change the values of a tuple, you can simply redefine the entire tuple.

```
mytuples = ("cars", "bikes", "motorbikes", "buses", "tanks",
"ships", "bikes")
print("This is the original tuple:")
for mytuple in mytuples:
  print(mytuple)

mytuples = ("jets", "bikes", "motorbikes", "buses", "tanks",
"ships", "bikes")
print("\nThis is the modified tuple:")
for mytuple in mytuples:
  print(mytuple)
```

This is the original tuple:

```
cars
bikes
motorbikes
buses
tanks
ships
bikes
```

This is the modified tuple:

```
jets
bikes
motorbikes
buses
tanks
ships
bikes
```

The code defines the original tuple, prints the initial values, and then stores the new tuple inside the modified values. Python does not return any error because it allows overwriting a tuple.

If you compare tuples with lists, you will understand that these are simple data structures. You can use them when you have to store different sets of values that need not be changed throughout a program's life.

Code Styling

When you are writing a long program, you should know how to style your code. You need to take the time to make sure that your code is easy to read and understand. Writing easily comprehensible code can help you track down what your programs do, and it also helps you understand the code.

Python programmers have largely agreed upon different styling traditions to ensure that every person's code is properly structured in the same manner. Once you have learned to write a neat and clean code, you will understand the structure of other's code as well. If you hope to become a professional programmer at any point in your life, you need to begin following the guidelines as soon as possible.

If someone tries to change the Python language, they start writing Python Enhancement Proposal (PEP). One of the oldest PEPs is the PEP8. It instructs Python programmers on how they can style their code. PEP 8 is fairly lengthy. However, most can relate to a highly complex coding structure than what you might have seen until now.

The Python style guide was produced with the aim that code is easily read than it is written. You will write your code once and start reading it as you begin the process of debugging. When you add new features to a Python program, you will need to spend time reading the code. When you share your code with other programmers, they will be able to read the code easily.

Between the choice of writing code that is easier to read or write, Python programmers encourage you to write code that can be easily readable by others.

PEP8 recommends that you should use four spaces for each indentation level. Using four spaces will improve the readability while leaving the room for multiple indentation levels on each line of code. In a word processing document, you might have

used tabs instead of spaces. This works well because it makes the paragraphs easily readable. The Python interpreter does not easily distinguish between tabs and spaces. It looks at each differently. Every text editor has a setting that provides you the opportunity to use the TAB key. You should use the TAB key, making sure that the editor is all set to insert several spaces rather than the tabs in the document.

Most senior and professional programmers recommend that each line in Python code should not exceed 80 characters. This guideline has historically developed because most computers could fit only 79 characters on a single line inside a terminal window. Presently, people can fit longer lines on the screens, but other reasons exist to adhere to 79 characters. As professional programmers have several files opened on the screen, using the standard length of lines helps them see complete lines in two to three files that can be opened on side-by-side screens. PEP8 also recommends that you should limit the comments to 72 characters per line as some of the tools that would generate automatic documentation for bigger projects would add the formatting characters at the start of commented lines.

However, these guidelines are not final. You can change the character limit at will. You need not worry about the length of the line. Most editors allow you to view the lines in a vertical format on the screen. It will help you know the limit as well.

You can use blank lines to group different parts of the program in a visual format. You need to see blank lines for organizing the files, but you should not do so excessively. Blank lines do not affect how your code runs. The Python interpreter uses horizontal indentation for interpreting the deep meaning in the code. However, it utterly disregards any kind of vertical spacing.

Chapter Five

Python Conditionals

Programming involves the examination of different conditions and making decisions based on those conditions. The Python if statement will allow you to examine the present state of a Python program and also respond to the same most appropriately to the same state.

This chapter will walk you through different conditional tests that allow you to test any condition. You will learn how to write the if statements, and you will also learn how to build a complex series of conditional statements when you can sense that the exact conditions are present. After that, you will be able to apply the concept to lists.

I will start with a short example that shows how if tests can allow you to respond to special situations properly. Just take a list of different types of bike companies and print the name of each of them. The names are proper names, and they should be printed in the title case. Some of them also need to be printed in upper case as well. See how the code works.

```
bikes = ("toyota", "bmw", "suzuki", "yamaha", "harley davidson",
"honda")
for bike in bikes:
   if bike == 'bmw':
      print(bike.upper())
   else:
      print(bike.title())

Toyota
BMW
Suzuki
```

```
Yamaha
Harley Davidson
Honda
```

The for loop in this example will check if the current value of the bike is bmw. If it finds it true, the value will be neatly printed in uppercase. If the value of the bike is not bmw and something else, it will print it in the title case.

In this example, I have combined several concepts that you will learn in the incoming sections. Let us start by looking at the types of tests that you may use to examine the program's conditions.

The core of if statements is the conditional test. This rests at the heart of the if statement. It is an expression that may be evaluated as False or True and is known as the conditional test. Python uses True and False values to decide whether the code in the if-statement needs to be executed or not. If the conditional test is true, Python will execute code that follows the if statement. If the test turns out to be false, Python will ignore the code and the if statement.

Most conditional tests tend to compare the values of a specific variable to the value of interest. The simplest of the conditional tests will check if the variable's value is equal to the value of interest.

```
>>> bike = 'bmw'
>>> bike == 'bmw'
True
>>>
```

At the first line of the code, the value of the bike is set at ‘bmw.’ I used the single equal sign in the code to. The second line of code tests if the value of the bike is bmw by using the double equal signs. This is known as the equality operator. It will return True if the values to the right and left sides of the operator match. It will return False if the values to the right and left do not match. As the

values to the right and left sides match in this code, the result is True. Let us see another example.

```
>>> bike = 'bmw'
>>> bike == 'honda'
False
>>>
```

When you check for equality and ignore the value's case, you will see a different result than you had expected. Testing for equality is extremely case-sensitive in the world of Python. For example, two values that have different capitalization will never be considered equal.

```
>>> bike = 'bmw'
>>> bike == 'Bmw'
False
>>>
```

If case is important, the behavior is advantageous. When it does not matter, and you need to test the value of a variable, you may be able to convert the value of a variable into lowercase before running the comparison.

```
>>> bike = 'BMW'
>>> bike.lower() == 'bmw'
True
>>>
```

Different websites enforce a set of rules for the data that the users will enter in a similar manner. For example, a website may use a conditional test like this to ensure that each user carries a unique username and not simply a variation on the capitalization of that person's username. When a person submits their username, the new username is immediately converted into lowercase and then compared to the lowercase versions of usernames. The username Adam will be immediately rejected if the system is already using any variation of ‘adam.’

If you want to determine if the two values are equal or not, you may combine an equal sign with an exclamation point. The exclamation denotes a ‘not’ as it does in most of the programming languages. Now I will use another if statement to test how we can use the inequality operator.

```
ordered_dessert = ('chocolate cake')
if ordered_dessert != 'fruit salad':
   print("Please serve the fruit salad.")

Please serve the fruit salad.
```

On the first line, I compared the values of ordered_dessert to fruit salad. If the values don’t match, Python declares it true and executes the code. If the values match, Python returns False and stops running the code. As the value of ordered_dessert is not fruit salad, the code is immediately executed. Most conditional expressions that you write will test for equality. However, sometimes you will find it highly efficient if you test it for inequality.

Numerical Testing

You also can do numerical testing. See the following statements to learn how Python tests numbers and returns results.

```
the_answer = 45
if the_answer != 55:
   print("Your answer is incorrect. Please try to run the code again.")

Your answer is incorrect. Please try to run the code again.
```

The conditional test passes because the value 45 is not equal to 55. Because the test is meant to pass, the indented block of code is executed. Let us see what happens if the test does not pass.

```
the_answer = 45
if the_answer != 45:
   print("Your answer is incorrect. Please try to run the code again.")
```

If you run the code like this, you will see nothing on the interpreter screen.

Testing Multiple Conditions

You may need to check multiple conditions simultaneously. For example, you may need a couple of conditions to turn out to be true for taking action. Other times, you may be satisfied with one condition turning out to be true. The keywords *or* and *and* will aid you in these tricky situations.

If you want to check if two conditions are true, you need to use the *and* keyword to combine the conditional tests. If each test passes, the overall expression will be evaluated to be True. If either or both tests fail, the expression will be evaluated to be False.

For example, you may check if two persons are over 18 or not by using the following example.

```
>>> age1 = 25
>>> age2 = 18
>>> age1 >= 22 and age2 >= 17
True
Here is the second code.
>>> age1 = 25
>>> age2 = 18
>>> age1 >= 22 and age2 >= 20
False
```

You can also use *or* to check multiple conditions in Python. The *or* keyword will allow you to check different conditions, but it also passes when both or either of the individual tests will pass. The *or* expression will only fail when both tests fail. Let us use the *or* keyword in the following example.

```
>>> age1 = 22
>>> age2 = 18
>>> age1 >= 20 or age2 >= 20
True
```

```
>>> age1 = 18
>>> age1 >= 22 or age2 >= 21
False
>>>
```

The conditionals have some other important jobs to do as well. Sometimes you feel the need to check if a list contains a specific value before you take an action. For example, you might need to check if a certain bike vegetable exists in a list of vegetables. For example, if you develop a grocery app with a list of vegetables, you can use the conditionals to let users know if a certain vegetable exists in the list or not.

To find out if a specific value exists in a list, you have to use the word in. Let us consider a piece of code that you might use for a grocery app. I will create a list of vegetables from which a customer may select a vegetable to buy.

```
>>> veggies = ['potato', 'tomato', 'lettuce', 'cilantro', 'cabbage',
'pumpkin']
>>> 'tomato' in veggies
True
>>> 'pepper' in veggies
False
>>>
```

This method is extremely powerful because you can build a list of vegetables and then check if a certain value exists in the list while you match items.

Example

Sometimes it is important to know whether a specific value appears in a list or not. You can use the keyword, *not* in a specific situation. See how I use the vegetable list in a program.

```
veggies = ['potato', 'tomato', 'lettuce', 'cilantro', 'cabbage',
'pumpkin']
veg_item = 'pepper'
veg_item1 = 'bell pepper'
```

```
if veg_item not in veggies:
   print(veg_item.title() + " is not available at the store right now.")
   print(veg_item1.title() + " is not available at the store right now.")

Pepper is not available at the store right now.
Bell Pepper is not available at the store right now.
```

The first line of code tells Python to return True if the requisite value is not in the list. Then it goes on to execute the second line of code. The vegetables, pepper, and bell pepper are in the vegetables list, so Python displays the message neat and formatted.

The if Statement

When you get to know the conditional tests, you may start writing the *if* statements. There are different types of if statements in Python. Your choice of which you should use will depend on the total number of conditions that you are looking forward to testing. You have seen multiple examples of the *if* statements. Let us what the structure of each of them is and how they function.

Simple if

The simplest type of if statement has to perform one action. You can put this conditional test inside the first line and around any action in the indented block of code after the test. If the conditional test returns the True value, Python will execute the code that fails after the if statement. Let us say a variable represents the age of a person who seeks to join a golf club. To join the club, the person must be over the age of 18. The following lines of code will test if the person is more than 18 or not.

```
entry_age = 20
if entry_age >= 18:
   print("You have the minimum required age to join the golf club.")

You have the minimum required age to join the golf club.
```

Python will check to see if the value of the variable entry_age is bigger than or equal to 18. If it is, Python will execute the print statement that is indented in the code. Indentation has the same role to play in the if statements as it does in the for loops. All the indented lines that follow the if statement must be executed if the test passes. The entire block of the indented lines will be ignored if your test does not pass.

You may have multiple lines of code that you need in the block after the execution of the if statement. I will now add another line of code. Let us see how it will change the code.

```
entry_age = 20
if entry_age >= 18:
   print("You have the minimum required age to join the golf club.")
   print("Have you submitted your application to join the club?")

You have the minimum required age to join the golf club.
Have you submitted your application to join the club?
```

The conditional test passes. As both print statements are indented, both are printed logically. If you change the value to less than 18, the program will produce no output at all.

if-else Statements

You will more often need to take one action upon passing one conditional test and another action in the other cases. The if-else syntax will make it extremely possible. The if-else block is extremely similar to the if statement. The only difference is that the else statement allows you to define a set of actions or a single action that should be executed whenever a particular conditional test fails.

I will display the same message here, but I will add a new message for the person who cannot join the club.

```
entry_age = 17
if entry_age >= 18:
   print("You have the minimum required age to join the golf club.")
```

```
    print("Have you submitted your application to join the club?")
else:
    print("I am sorry, you are not old enough to join our golf club.")
    print("Please file your application when you are 18 years old.")

I am sorry, you are not old enough to join our golf club.
Please file your application when you are 18 years old.
```

If the conditional test had passed, the first block of code would be executed. Now that the first conditional test has failed, the second one has been executed. The test returns a false value, so the second block of code is executed. As the requisite age is less than 18, the test fails, and you see the second line of statements.

The if-elif-else Chain

More often, you have to test multiple possible situations. In this kind of scenario, you have to use the if-elif-else syntax. Python will execute just one block of code in the if-elif-else chain. It will run the conditional test until one test passes. When one passes, the code linked to that test will pass. Python will skip everything else.

There are a lot of real-world situations where you need to test more than two conditions. For example, sometimes, you need to allow people of multiple ages to enter a place after scanning each of them.

The following code will test the age of different people and print a statement as to where they can enter or not.

```
entry_age = 31
if entry_age > 30:
    print("You have the minimum required age to join the golf club
for the mature.")
    print("Have you submitted your application to join the club?")
elif entry_age < 18:
    print("I am sorry, you are not old enough to join our golf club.")
    print("Please file your application when you are 18 years old.")
else:
```

```
    print("You are eligible only to play golf in a restricted practice
area.")
    print("Please process your application at booth no 7 and move
in")
```

You have the minimum required age to join the golf club for the mature.

Have you submitted your application to join the club?

Now I will change the value and the else statement to bring them into action.

```
entry_age = 25
if entry_age > 30:
    print("You have the minimum required age to join the golf club
for the mature.")
    print("Have you submitted your application to join the club?")
elif entry_age < 18:
    print("I am sorry, you are not old enough to join our golf club.")
    print("Please file your application when you are 18 years old.")
else:
    print("You are eligible only to play golf in a restricted practice
area.")
    print("Please process your application at booth no 7 and move
in")

You are eligible only to play golf in a restricted practice area.
Please process your application at booth no 7 and move in
```

If you study the code deeply, you will realize that the code can handle whatever age number you fill in. There is a statement for each category of age. There is a message for each category of age. All you need to fill in the age number and know how you have to move on.

Another Example

You can use the if statements to find out the entry fee of a person. The following code will test different age groups and then print a message as to the fee of each age category.

```
entry_age = 18

if entry_age < 5:
   print("Your entry fee will be $1.")

elif entry_age < 18:
   print("Your entry fee will be $5.")
else:
   print("Your entry fee will be $15.")
```

```
Your entry fee will be $15.
```

Multiple elif Statements

```
entry_age = 45
if entry_age < 30:
   print("You have the minimum required age to join the golf club
for the mature.")
   print("Have you submitted your application to join the club?")
elif entry_age < 18:
   print("I am sorry, you are not old enough to join our golf club.")
   print("Please file your application when you are 18 years old.")
elif entry_age < 50:
   print("You can play gold in the elite club with professional
players.")
   print("Please submit the fee and move on.")

else:
   print("You are eligible only to play golf in a restricted practice
area.")
   print("Please process your application at booth no 7 and move
in")
```

```
You can play gold in the elite club with professional players.
Please submit the fee and move on.
```

Let us change the value and print another elif statement.

```
entry_age = 17
```

```
if entry_age > 18:
   print("You have the minimum required age to join the golf club
for the mature.")
   print("Have you submitted your application to join the club?")
elif entry_age < 18:
   print("I am sorry, you are not old enough to join our golf club.")
   print("Please file your application when you are 18 years old.")
elif entry_age < 50:
   print("You can play gold in the elite club with professional
players.")
   print("Please submit the fee and move on.")

else:
   print("You are eligible only to play golf in a restricted practice
area.")
   print("Please process your application at booth no 7 and move
in")
```

```
I am sorry, you are not old enough to join our golf club.
Please file your application when you are 18 years old.
```

Omitting else Block

Python doesn't always require the else block at the tail of the if-elif chain. The else block is sometimes useful. However, in some cases, the elif statement is the one that is of primary interest and benefit.

```
entry_age = 65
if entry_age < 18:
   print("I am sorry, you are not old enough to join our golf club.")
   print("Please file your application when you are 18 years old.")
elif entry_age < 30:
   print("You have the minimum required age to join the golf club
for the mature.")
   print("Have you submitted your application to join the club?")
elif entry_age < 65:
   print("You can play gold in the elite club with professional
players.")
   print("Please submit the fee and move on.")
```

```
elif entry_age >= 65:
   print("You are eligible only to play golf in a restricted practice
area.")
   print("Please process your application at booth no 7 and move
in")
```

```
You are eligible only to play golf in a restricted practice area.
Please process your application at booth no 7 and move in
```

The else block is known as a catchall statement. It will match any condition that is not matched by a specific elif or if test. It can sometimes include malicious or even invalid data as well. If you have a particular final condition that you are testing, you need to consider the elif block and eliminate the else block. This will ensure that your code will only run under correct situations.

Multiple Conditions

The if-elif-else chain is robust, and it is quite appropriate for usage when you need just a single test to pass. As soon as Python finds a test will pass, it will skip the other tests. This behavior is quite beneficial because it is quite efficient and will allow you to test a specific condition.

Sometimes, you need to check the conditions that interest you. You need to use simple if statements without any else or elif blocks. The technique makes perfect sense when multiple conditions are true, and you need to act on the condition that is True.

```
veggies = ['potato', 'tomato', 'lettuce', 'cilantro', 'cabbage',
'pumpkin']
if 'tomato' in veggies:
   print("I am adding 'tomato' to the cart.")
if 'lettuce' in veggies:
   print("I am adding 'lettuce' to the cart.")
if 'cabbage' in veggies:
   print("I am adding 'cabbage' to the cart.")
if 'cilantro' in veggies:
   print("I am adding 'cilantro' to the cart.")
```

```
print("\nThe cart is ready. Please pay through credit card and check out.")
```

```
I am adding 'tomato' to the cart.
I am adding 'lettuce' to the cart.
I am adding 'cabbage' to the cart.
I am adding 'cilantro' to the cart.

The cart is ready. Please pay through credit card and check out.
```

I started with the list of vegetables and kept adding my favorite to my cart. Each time I added a vegetable to the cart, a message was printed on the screen. The addition of vegetables is the first test to pass. Therefore, vegetables are added to the cart of the e-commerce grocery store.

If Statement & Lists

You can pair up if statements with lists to create amazing outputs. You can produce special values that must be treated differently from the other values inside the list. You can also look out for any special values that must be treated differently from the list's other values. You may manage efficiently changing the conditions like the availability of different items inside a restaurant across the shift. You can start to prove that the code you are writing works as you expect it to be in different conditions.

Special Items

I will continue the example of the grocery app in this section as well. I have already explained the conditional test so let navigate around to see if you can check for any special values inside the list and handle the values appropriately.

The veggies example will print a message where a vegetable is added to the cart. The code for the action should be written

efficiently. I will create a list of the toppings that a customer is about to buy and add to the cart. Then I will use a loop to display a message to the user.

```
veggies = ['potato', 'tomato', 'lettuce', 'cilantro', 'cabbage',
'pumpkin']
for veggie in veggies:
   print("I am adding " + veggie + ".")


print("\nThe cart is ready. Please pay through credit card and check
out.")
```

```
I am adding potato.
I am adding tomato.
I am adding lettuce.
I am adding cilantro.
I am adding cabbage.
I am adding pumpkin.

The cart is ready. Please pay through credit card and check out.
```

The output is quite straightforward as the code is a simple for loop. Suppose a person is shopping at the grocery app, and the store runs out of lettuce. What happens next? If you add an if statement inside the for loop, it can handle this typical situation very well.

```
veggies = ['potato', 'tomato', 'lettuce', 'cilantro', 'cabbage',
'pumpkin']
for veggie in veggies:
   if veggie == 'lettuce':
      print("We are sorry as we have run out of lettuce.")
   else:
      print("I am adding " + veggie + ".")


print("\nThe cart is ready. Please pay through credit card and check
out.")
```

```
I am adding potato.
```

```
I am adding tomato.
We are sorry as we have run out of lettuce.
I am adding cilantro.
I am adding cabbage.
I am adding pumpkin.

The cart is ready. Please pay through credit card and check out.
```

This time, the code behaved differently. The code checks each requested item before it is added to the cart. The code in the first line checks if the person has requested the lettuce or some other vegetable. If the person requests lettuce, the program will print a message explaining that they have run out of lettuce. The else block at the second line of code ensures that the other vegetables the person ordered are added to the cart for check out. The output shows that each vegetable that is requested is properly handled.

You also can check if the list you are working on is empty or not. We have made an assumption about each list we have worked with. Let us assume that each list has at least one item. You can let users add information to the lists. It is quite useful to see if the list is empty before the start of the loop or not. As an example, I will check first if the list of the veggies is empty before the user adds something to the cart. If the list is empty, I will ask the user whether he likes to come back again to visit at a later time. If the list is not empty, the process will go on smoothly as it did in the previous examples.

```
veggies = []

if veggies:
   for veggie in veggies:
      print("I am adding " + veggie + ".")
   print("\nThe cart is ready. Please pay through credit card and
check out.")
else:
   print("We would appreciate if you could visit at our store at a
later time.")
```

We would appreciate if you could visit at our store at a later time.

I started out with an empty list of veggies. Instead of moving straight to the for loop, I quickly checked the second line of code. When the name of the list is included in the if statement, Python will return True even if the list has just one value. If veggies passes the conditional test, I will run the same for loop that I had used in the previous example. If the test fails, I will print a message asking a customer if they would like to come back again later because the store is presently empty.

Multiple Lists

People may ask for anything in the store when it is about a grocery store app. What if one customer wants fruits alongside vegetables. You may use lists and the if statement to ensure that the input makes perfect sense before you start acting on it.

```
veggies = ['strawberry', 'apple', 'peach', 'potato', 'tomato', 'lettuce',
'pumpkin', 'radish']

requested_fruits = ['strawberry', 'orange', 'apple', 'peach', 'guava']

for requested_fruit in requested_fruits:
   if requested_fruit in veggies:
      print("I am adding " + requested_fruit + ".")
   else:
     print("We don't have " + requested_fruit + ". We would
appreciate if you could visit at our store at a later time.")

print("\nThe cart is ready. Please pay through credit card and check
out.")
```

```
I am adding strawberry.
```

We don't have orange. We would appreciate if you could visit at our store at a later time.

I am adding apple.
I am adding peach.

We don't have guava. We would appreciate if you could visit at our store at a later time.

The cart is ready. Please pay through credit card and check out.

In the first line of code, I defined a list of veggies. You also can use a tuple here if the grocery app has a stable list of vegetables. In the second line of code, I made a list of fruits that the users request through the app. In the third line of code, I looped through the list of requested_fruits. In the loop, I checked to see if the grocers already include each fruit into the veggies list or not. If it is, the fruit will be added to the cart. If not, the user will have a message on display through the else block. The else block will run and print a message explaining to the user which fruits are not available right now.

Chapter Six

Python Dictionaries

This chapter will walk you through Python dictionaries. These allow you to connect different pieces of information that are related to one another. You will also learn how you can access information inside the dictionary. You will also learn how you can modify the requisite information. Python dictionaries can store unlimited values. The amount of information that you can add to has no limit. Just like lists, you can create a loop through the dictionaries. You will also learn how to nest one dictionary into lists, nest lists inside dictionaries, and nest dictionaries inside dictionaries.

If you develop a good understanding of dictionaries, you can accurately create models of various real-world objects. You can create a dictionary that represents a person and stores as much information as you need about the same. You may store their name, location, age, profession, married life information, and other information. At a given time, you can create two kinds of information that may be matched up like gender and profession or name and gender. That's how you can do anything with Python dictionaries. You can create a list of cities and their capitals or a list of mountains and their heights. Everything should be in pairs. That's the top requirement.

A Dictionary

In this section, I will create a simple dictionary featuring different eatable items and their types.

```
veggie_fruit = {'apple' : 'fruit', 'radish' : 'vegetable'}

print(veggie_fruit['apple'])
```

```
print(veggie_fruit['radish'])

fruit
vegetable
```

The dictionary veggie_fruit stores the name of the fruit and vegetables and their types. The print statements at the end of the code display the information as printed below. Dictionaries may look simple, but they need practice if you want to master them. Once you know how to use them, you will be better positioned to model some real-world situations through coding.

Python dictionaries are built in the form of key-value pairs where each key has a value. If you know the key, you can access its value. The value can be anything like a list, a number, a word, or another dictionary. If it were not for the dictionaries, Python would be so popular. It is the dictionaries that allow programmers to build objects in Python. By storing information in a dictionary about a real-life object, programmers model those objects in the computer world. You have to use curly braces to enclose a dictionary and add the information in key-value pairs.

The simplest dictionary has a single key-value pair. Once a dictionary has been created, you can modify it any time with more information. There is no limit to the maximum stored information in a dictionary.

Accessing Values

It is simple to access values in a dictionary. To get a value linked to a key, you need to enter the name of a dictionary and then fill it up with the key in square brackets. The following example contains the practical example for accessing values.

```
veggie_fruit = {'apple' : 'fruit', 'radish' : 'vegetable'}

print(veggie_fruit['apple'])
print(veggie_fruit['radish'])
```

```
fruit
vegetable
```

I have accessed values by entering the keys in the print statement. This is one method to access values in a dictionary. However, to use this method effectively, you need to know what your key is. You can keep a list of keys to access the related values, and you can have unlimited key-value pairs in a dictionary. You can use the information after you access it through a print statement.

```
veggie_fruit = {'fruit' : 'apple' , 'vegetable' : 'radish'}

buy1 = veggie_fruit['fruit']
print(buy1.title() + " is just added to your cart.")

Apple is just added to your cart.
```

Once you have defined the dictionary, the first line of code will pull the value linked to the key fruit in the dictionary. The value goes to the variable buy1.

Dictionaries are considered dynamic structures. You can add many new key-value pairs to a specific dictionary at a given time. For example, you can add key-value pairs to the dictionary. In the following example, I will add new items to the dictionary and see how they are added and how they work.

```
veggie_fruit = {'fruit' : 'apple' , 'vegetable' : 'radish'}
print(veggie_fruit)

veggie_fruit['fruit1'] = 'orange'
veggie_fruit['vegetable1'] = 'pumpkin'
veggie_fruit['fruit2'] = 'peach'
veggie_fruit['vegetable2'] = 'carrot'

print(veggie_fruit)

{'fruit': 'apple', 'vegetable': 'radish'}
```

```
{'fruit': 'apple', 'vegetable': 'radish', 'fruit1': 'orange', 'vegetable1':
'pumpkin', 'fruit2': 'peach', 'vegetable2': 'carrot'}
```

You can see that I added four key-value pairs to the dictionary, and then I printed the dictionary. I defined the dictionary first that I had to modify by adding new key-value pairs. I defined the keys and then the values for each pair by using the equal sign and the dictionary's name. When I printed the modified dictionary, all the new key-value pairs were added to the dictionary.

It is pertinent to note that the new key-value pairs may not match the order in which you add them. Python is not concerned about the order of the key-value pairs. The only thing it cares about is the connection that each key has with its value.

Empty Dictionary

You can start with an empty dictionary and fill it up with key-value pairs. You can add as many items to the dictionary as you want. The first step in this regard is to define the dictionary with the help of empty braces. Then you can add key-value pairs on the new lines. See the following example to learn how you can build a dictionary.

```
veggie_fruit = {}

veggie_fruit['fruit'] = 'apple'
veggie_fruit['vegetable'] = 'radish'
veggie_fruit['fruit1'] = 'orange'
veggie_fruit['vegetable1'] = 'pumpkin'
veggie_fruit['fruit2'] = 'peach'
veggie_fruit['vegetable2'] = 'carrot'

print(veggie_fruit)
```

```
{'fruit': 'apple', 'vegetable': 'radish', 'fruit1': 'orange', 'vegetable1':
'pumpkin', 'fruit2': 'peach', 'vegetable2': 'carrot'}
```

You will be using empty dictionaries when you start working with the datasets that must be supplied and filled up by the data provided by users, or when you start writing the code that will generate a great number of key-value pairs automatically.

Modification

You can easily modify a dictionary. Just write the name of the dictionary with the key inside the square brackets and the new value that you need to associate with the key.

```
veggie_fruit = {'fruit' : 'apple' , 'vegetable' : 'radish'}
print("The fruit I have to buy is " + veggie_fruit['fruit'] + ".")


veggie_fruit['fruit'] = 'peach'
print("The fruit I have to buy is " + veggie_fruit['fruit'] + ".")

veggie_fruit['fruit'] = 'orange'
print("The fruit I have to buy is " + veggie_fruit['fruit'] + ".")

The fruit I have to buy is apple.
The fruit I have to buy is peach.
The fruit I have to buy is orange.
```

I defined the dictionary. Then I changed the value that is associated with the key fruit. I did that twice. The output shows the new fruit that the buyer will buy is different from the original one.

Key-Pair Removal

When you do not need a certain piece of information that is stored in the dictionary, you can use the del statement to eliminate it from the key-value pair. The del statement requires the name of the dictionary and its related key. Let’s see how we can use the del statement to remove key-value pairs from the dictionary.

```
veggie_fruit = {'fruit' : 'apple' , 'vegetable' : 'radish', 'fruit1' :
'orange', 'vegetable1' : 'pumpkin', 'fruit2' : 'peach', 'vegetable2' :
```

```
'carrot'}
print(veggie_fruit)

del veggie_fruit['fruit']
print(veggie_fruit)

del veggie_fruit['fruit1']
print(veggie_fruit)

del veggie_fruit['fruit2']
print(veggie_fruit)
```

```
{'fruit': 'apple', 'vegetable': 'radish', 'fruit1': 'orange', 'vegetable1':
'pumpkin', 'fruit2': 'peach', 'vegetable2': 'carrot'}
{'vegetable': 'radish', 'fruit1': 'orange', 'vegetable1': 'pumpkin',
'fruit2': 'peach', 'vegetable2': 'carrot'}
{'vegetable': 'radish', 'vegetable1': 'pumpkin', 'fruit2': 'peach',
'vegetable2': 'carrot'}
{'vegetable': 'radish', 'vegetable1': 'pumpkin', 'vegetable2': 'carrot'}
```

You can see that I have deleted all the fruit key-value pairs from the dictionary. The del keyword tells Python to delete the key from your dictionary. It also removes the value that is associated with the key. The output shows that the key and its related value are deleted from the dictionary. As you can see, for each item I applied the del statement, one key-value pair got deleted from the dictionary. The only thing you need to care about is that the del statement permanently deletes a key-value pair from the dictionary. You cannot get the pair back in any case.

Similar Objects Dictionary

In this example, I will store a lot of information about a single object. For example, you can store information about a house that you need to put on sale. See how you can add multiple pieces of information about an object in one dictionary.

```
house = {'swimming pool' : 'fiber glass' , 'bathrooms' : 5,
'bedrooms' : 6, 'gym' : 'loaded with modern equipment', 'kitchen' :
'open & furnished with cooking range', 'TV lounge' : 'furnished with
cinema-sized screen'}
print(house)
```

```
{'swimming pool': 'fiber glass', 'bathrooms': 5, 'bedrooms': 6,
'gym': 'loaded with modern equipment', 'kitchen': 'open &
furnished with cooking range', 'TV lounge': 'furnished with cinema-
sized screen'}
```

Here is how you can create a dictionary of a house that you need to put on sale. On a similar pattern, you can create different kinds of dictionaries and use them to conduct day-to-day operations.

The dictionary has been broken down from a larger dictionary into small chunks. Each key in the dictionary denotes the name of a section of the house and its description. When you realize that you need multiple lines of information to define a dictionary, you can simply hit ENTER after one brace. Then you need to indent the next line and write the next key-value pair, followed by a comma. Once you have defined the entire dictionary, you need to fill in a closing brace on the new line of the editor. Then you must indent that line as well so that it is perfectly aligned with a set of keys inside the dictionary. You can add a comma after the final key-value pair so that you are ready to include a new key-value pair in the next line of code. Most Python editors will help you format extended lists as well as dictionaries. It is their built-in feature.

Let us see how we can use the above-mentioned dictionary in a program.

```
house = {'swimming pool' : 'fiber glass' , 'bathrooms' : 5,
'bedrooms' : 6, 'gym' : 'loaded with modern equipment', 'kitchen' :
'open & furnished with cooking range', 'TV lounge' : 'furnished with
cinema sized screen'}
```

```
print("The swimming pool of the house is made of " +
house['swimming pool'].title() + ".")
print("The kitchen of the house is " + house['kitchen'].title() + ".")
```

```
The swimming pool of the house is made of Fiber Glass.
```

The kitchen of the house is Open & Furnished With Cooking Range.

You can see that you have to break up the print statement to ensure that the code works properly. The word print is mostly shorter than the dictionaries' names, which is why it makes sense that you need to find the most appropriate spot to break up the print statement.

Looping a Dictionary

Python dictionaries are humongous stores of data. You can create a dictionary and fill it up with a few pairs or a million pairs. It is easy to navigate through a short dictionary and really hard to do that with a large dictionary. That’s why when you are working with a longer dictionary, you should add a loop to the code. You can create a loop through a dictionary in several ways, like looping through all the key-value pairs or looping through only the keys or only the values.

The integration of loops into a dictionary is an easy way to handle a dictionary, especially if the dictionary belongs to a corporate firm where the data is literally in millions of key-value pairs. Looping is the only way to manage large-sized data effectively.

I will use the same dictionary about a house. You can access single pieces of information. What if you need to see everything in the dictionary? If you want to do that, you can use the for loop to print all the keys and values of the dictionary. Let us see how you can build the for loop and print the requisite values.

```
house = {'swimming pool' : 'fiber glass' , 'bathrooms' : 'maple wood
flooring', 'bedrooms' : 'Iranian carpeting', 'gym' : 'loaded with
modern equipment'}
```

```
for key, value in house.items():
   print("\nThe house features a " + key)
   print("Here is the description: " + value)

The house features a swimming pool
Here is the description: fiber glass

The house features a bathrooms
Here is the description: maple wood flooring

The house features a bedrooms
Here is the description: Iranian carpeting

The house features a gym
Here is the description: loaded with modern equipment
```

I wrote the for loop for the dictionary and passed it through the keys and values of the dictionary.

The second half of the for statement includes the names of a dictionary. This will return the list of all the key-value pairs. I have created two variables, namely key and value, to store the information about the keys and values of the dictionary. You can create other variable names as suit you.

There is another way to use the information in a dictionary. You can use the information to display a neatly formatted message.

```
house = {'swimming pool' : 'fiber glass' , 'bathroom' : 'maple wood
flooring', 'bedroom' : 'Iranian carpeting', 'gym' : 'modern
equipment'}

for feature, desc in house.items():
   print("The " + feature.title() + " of the house on sale has " +
desc.title() + ".")

The Swimming Pool of the house on sale has Fiber Glass.
The Bathroom of the house on sale has Maple Wood Flooring.
The Bedroom of the house on sale has Iranian Carpeting.
The Gym of the house on sale has Modern Equipment.
```

The program iterates through key-value pairs in the dictionary. As it runs through each pair, Python keys are forwarded to a variable named ‘feature.’ The values are forwarded and stored in the variable named as ‘desc.’ The information was displayed in a neat and formatted style. Descriptive variables make it easy to write and read the code. For practice, you should change the names of variables and try your own names.

Looping Through Keys

The keys() of the dictionary allow you to create a loop only through the keys of the dictionary. This time, I will loop through the dictionary and only print the house's features, which are also the dictionary's keys.

```
house = {'swimming pool' : 'fiber glass' , 'bathroom' : 'maple wood
flooring', 'bedroom' : 'Irani carpeting', 'gym' : 'modern equipment'}

for feature in house.keys():
   print("Feature of the house: " + feature.title() + ".")

Feature of the house: Swimming Pool.
Feature of the house: Bathroom.
Feature of the house: Bedroom.
Feature of the house: Gym.
```

Looping through keys of a dictionary is considered the default behavior in Python. Therefore, when creating a loop through a dictionary, the following code will have the same output as the above-mentioned code.

```
house = {'swimming pool' : 'fiber glass' , 'bathroom' : 'maple wood
flooring', 'bedroom' : 'Iranian carpeting', 'gym' : 'modern
equipment'}

for feature in house:
   print("Feature of the house: " + feature.title() + ".")

Feature of the house: Swimming Pool.
Feature of the house: Bathroom.
Feature of the house: Bedroom.
Feature of the house: Gym.
```

You can apply the keys() method if you think it will make your code easier to read. You have the option to access the values with the help of keys by manipulating the current key. In the following example, I will change the program to display a neat message to buyers interested in buying your house. The program will include a loop that will iterate through the features that I have packed up in the dictionary. Whenever a feature matches the requirements of a buyer, the buyer will see a message on the screen.

```
house = {'swimming pool' : 'fiber glass' , 'bathroom' : 'maple wood
flooring', 'bedroom' : 'Iranian carpeting', 'gym' : 'modern
equipment'}

buyer_demand = ['bathroom', 'gym']
for feature in house.keys():
   print(feature.title())

   if feature in house:
      print("Most buyers are inquiring about " + feature.title() + "
which has "+ house[feature].title() + ".")
```

```
Swimming Pool
Most buyers are inquiring about Swimming Pool which has Fiber
Glass.
Bathroom
Most buyers are inquiring about Bathroom which has Maple Wood
Flooring.
Bedroom
Most buyers are inquiring about Bedroom which has Iranian
Carpeting.
Gym
Most buyers are inquiring about Gym which has Modern
Equipment.
```

I created a list of features that buyers are asking about to create a message to show buyers what the feature has. I also checked if the feature that the buyer is asking about is in the dictionary or not. If it is, the buyer gets a detailed message that also describes the feature to make the buying decision easier.

You can use the loops to display all the keys in perfect order. A dictionary usually maintains a formal connection between the keys and the values. However, it is unlikely that you get the items from the dictionary in a predictable order. This is not a problem because you will need to obtain the accurate value that is linked with each key. One way to get everything in perfect order is to sort out the keys as they are returned through the loop.

I will use the sorted function to bring everything in perfect order as they are returned in the loop.

house = {'swimming pool' : 'fiber glass' , 'bathroom' : 'maple wood flooring', 'bedroom' : 'Irani carpeting', 'gym' : 'modern equipment'}

```
for feature in sorted(house.keys()):
   print("You will have a " + feature.title() + " in the house.")
```

```
You will have a Bathroom in the house.
You will have a Bedroom in the house.
You will have a Gym in the house.
You will have a Swimming Pool in the house.
```

As I created loops through the keys, I will now create loops through the values in the dictionary. If you are interested in displaying the values, you can use the values() method to return the values without keys.

```
house = {'swimming pool' : 'fiber glass' , 'bathroom' : 'maple wood
flooring', 'bedroom' : 'Iranian carpeting', 'gym' : 'modern
equipment'}

print("Here is the list of values of the dictionary: ")
for desc in house.values():
   print(desc.title())
```

```
Here is the list of values of the dictionary:
```

```
Fiber Glass
Maple Wood Flooring
Iranian Carpeting
Modern Equipment
```

You can see that I pulled all the values in the dictionary. If you repeat some values, this code won't double-check it. This may work fine with a small bunch of values, but if the number of respondents is large, this will create incredible repetition. To confirm if each value is without repetition, you can use the set. Here is how the repetition of the code works.

```
house = {'swimming pool' : 'fiber glass' , 'bathroom' : 'maple wood
flooring', 'bedroom' : 'Iranian carpeting', 'gym' : 'modern
equipment', 'kitchen' : 'modern equipment'}

print("Here is the list of values of the dictionary: ")
for desc in house.values():
  print(desc.title())

Here is the list of values of the dictionary:
Fiber Glass
Maple Wood Flooring
Iranian Carpeting
Modern Equipment
Modern Equipment
```

The following code will eliminate the repetition from the code.

```
house = {'swimming pool' : 'fiber glass' , 'bathroom' : 'maple wood
flooring', 'bedroom' : 'Iranian carpeting', 'gym' : 'modern
equipment', 'kitchen' : 'modern equipment'}

print("Here is the list of values of the dictionary: ")
for desc in set(house.values()):
  print(desc.title())

Here is the list of values of the dictionary:
Fiber Glass
Iranian Carpeting
Maple Wood Flooring
```

Modern Equipment

Nesting

You may sometimes need to store a set of dictionaries inside a list or a list of items as one value inside a dictionary. The process is known as nesting. You can nest one set of dictionaries in a list or a list in a dictionary or a dictionary in a dictionary. Nesting is known as a robust feature of dictionaries. I will demonstrate the process in the following example.

I will create a dictionary for a battle game in the following example. The dictionary players is as under,

```
player_1 = {'gun' : 'revolver', 'enemies gunned' : 6}
player_2 = {'gun' : 'bazooka', 'enemies gunned' : 12}
player_3 = {'gun' : 'ak-47', 'enemies gunned' : 3}
player_4 = {'gun' : 'shot gun', 'enemies gunned' : 15}

players = [player_1, player_2, player_3]

for player in players:
   print(player)
```

```
{'gun': 'revolver', 'enemies gunned': 6}
{'gun': 'bazooka', 'enemies gunned': 12}
{'gun': 'ak-47', 'enemies gunned': 3}
```

First of all, I created four dictionaries, where each of them represented a player in the battle game. I packed up all of them into a list, namely players. Then I created a loop through the list and printed all the values of each dictionary on the screen.

You can create a fleet of players in the game if you are looking out to build an army. Players can use this dynamic piece of code to build an army and fight a large-scale war. Let’s see how it goes.

```
players = []
```

```
for player_num in range(70):
   player_army = {'gun': 'revolver', 'color': 'white', 'GPS' : 'enabled'}
   players.append(player_army)

   for player in players[:7]:
      print(player)
   print("....")

print("The total number of soldiers in the army: " +
str(len(players)))

{'gun': 'revolver', 'color': 'white', 'GPS': 'enabled'}
....
{'gun': 'revolver', 'color': 'white', 'GPS': 'enabled'}
{'gun': 'revolver', 'color': 'white', 'GPS': 'enabled'}
....
{'gun': 'revolver', 'color': 'white', 'GPS': 'enabled'}
{'gun': 'revolver', 'color': 'white', 'GPS': 'enabled'}
{'gun': 'revolver', 'color': 'white', 'GPS': 'enabled'}
....
{'gun': 'revolver', 'color': 'white', 'GPS': 'enabled'}
{'gun': 'revolver', 'color': 'white', 'GPS': 'enabled'}
{'gun': 'revolver', 'color': 'white', 'GPS': 'enabled'}
{'gun': 'revolver', 'color': 'white', 'GPS': 'enabled'}
....
The total number of soldiers in the army: 70
```

At the start of the example, an empty list holds the players that should be created. The range() function will return the set of numbers. It tells Python how many times you need the loop to repeat. Each time the loop runs its course, a new player will be created. It appended each player to the list of players. I used the slice method to print the players. All the players will have the same characteristics as you define for one. Python takes each of them as a separate object, which allows you to modify the players on an individual basis.

When the comes to change the guns, you can use the for loop and the if statement.

```
players = []

for player_num in range(0, 70):
    player_army = {'gun': 'revolver', 'color': 'white', 'GPS' : 'enabled'}
    players.append(player_army)

for player in players[0:3]:
    if player['gun'] == 'revolver':
        player['gun'] = 'machine gun'
        player['color'] = 'black'
        player['GPS'] = 'disabled'


for player in players[0:6]:
    print(player)
print("....")

{'gun': 'machine gun', 'color': 'black', 'GPS': 'disabled'}
{'gun': 'machine gun', 'color': 'black', 'GPS': 'disabled'}
{'gun': 'machine gun', 'color': 'black', 'GPS': 'disabled'}
{'gun': 'revolver', 'color': 'white', 'GPS': 'enabled'}
{'gun': 'revolver', 'color': 'white', 'GPS': 'enabled'}
{'gun': 'revolver', 'color': 'white', 'GPS': 'enabled'}
....
```

You can further add to this loop the elif block and extend its functionality. The addition of the elif block will further change the specifications of players.

```
players = []

for player_num in range(0, 70):
    player_army = {'gun': 'revolver', 'color': 'white', 'GPS' : 'enabled'}
    players.append(player_army)

for player in players[0:3]:
    if player['gun'] == 'revolver':
        player['gun'] = 'machine gun'
        player['color'] = 'black'
        player['GPS'] = 'disabled'
    elif player['gun'] == 'machine gun':
```

```
    player['gun'] = 'sten gun'
    player['color'] = 'brown'
    player['GPS'] = 'enabled'


for player in players[0:6]:
  print(player)
print("....")
```

Dictionary & Lists

Instead of filling up a dictionary into a list, it is sometimes highly useful to put the list into a dictionary. For example, we can use this feature to describe something. If we have to use a list, we can add details of an object to the list. With the help of a dictionary, a list of features of a house may just be an aspect of a house or something you have been describing.

I will store two types of information in the following dictionary. One is of the house type, and the second is of the features of that house.

```
house_sale = {
  'house' : 'mansion',
  'features' : ['swimming pool', 'tennis court', 'badminton court',
'basketball court', 'bedrooms', 'open & furnished kitchen']
  }

print("The house you have you shown your interest in is a " +
house_sale['house'] + " and it has the following features: ")

for feature in house_sale['features']:
  print("\t" + feature)
```

The house you have you shown your interest in is a mansion and it has the following features:

```
swimming pool
tennis court
badminton court
```

```
basketball court
bedrooms
open & furnished kitchen
```

I started with defining the dictionary that would hold information about the house on sale. One key inside the dictionary shows the type of house. The next key ‘features’ contains a list that has all the features of the house. When someone shows interest in purchasing the house, the final for loop will print all the elements in the list to show what the buyer will get upon the purchase. This helps us achieve multiple purposes. When I had to access the features of the house, I simply used the key features, and the entire list was brought to life on the screen. Python simply grabs the list and prints all the values.

You can use the nesting feature of Python dictionaries any time when you need to add more information to a single key.

```
player_army = {'gun': ['revolver', 'grenade'], 'gun color': ['white',
'black'], 'GPS' : ['enabled', 'military grade']}



for feature, descriptions in player_army.items():
   print("\n" + "Here is the detail of the key feature of your player: "
+ feature.title() + ".")
   for description in descriptions:
      print("\t" + description.title())


Here is the detail of the key feature of your player: Gun.
Revolver
Grenade

Here is the detail of the key feature of your player: Gun Color.
White
Black

Here is the detail of the key feature of your player: Gps.
Enabled
```

Military Grade

You can see that with each key in the dictionary, I have attached a list. Each list tells us more about the key. It elaborates upon the existing information. When I looped through the dictionary, I used the name descriptions to grab each value from the dictionary because each value would be a complete list. Inside the loop, I used a for loop that ran through the details of each feature of the player. Now each feature will have all the details. You can add as many details as you want to through these lists. I will expand upon the lists in the following example to demonstrate how you can create longer lists for each value. See the following example.

```
player_army = {'gun': ['revolver', 'grenade', 'revolver', 'shot gun',
'machine gun', 'bow', 'knife'], 'gun color': ['white', 'black', 'brown',
'yellow'], 'GPS' : ['enabled', 'military grade']}



for feature, descriptions in player_army.items():
   print("\n" + "Here is the detail of the key feature of your player: "
+ feature.title() + ".")
   for description in descriptions:
      print("\t" + description.title())
```

```
Here is the detail of the key feature of your player: Gun.
Revolver
Grenade
Revolver
Shot Gun
Machine Gun
Bow
Knife

Here is the detail of the key feature of your player: Gun Color.
White
Black
Brown
Yellow
```

```
Here is the detail of the key feature of your player: Gps.
Enabled
Military Grade
```

So, that's how you can create interactive games. Just imagine how cool your games look when the user has multiple options to display the features and descriptions of his player. It just adds more fun to your games and apps. You can do the same to the grocery app that we had been trying to build through Python in the past chapters. You can take that as a test and practice this code on the grocery app code.

You can use an if statement at the start of the for loop to see if each person has multiple guns. This is done by knowing the length of the items in the lists. If a person has multiple guns, the output can stay the game. If a person has only one, you can change the output.

A word of warning is that you don't have to nest dictionaries and lists too deeply in the code. It would make the code complex, and you will find it difficult to study and comprehend it.

You also can nest a dictionary inside a dictionary. However, this, too, makes the code complicated fast. If you have multiple users on a website and each has a unique username, you can use these usernames as keys inside the dictionary. After that, you can store the information about each user by using that dictionary as a value linked to the username. See how you can do that in the following example.

```
player_army = {'gun': {'revolver' : 'one round', 'grenade': 'four' ,
'shot gun': '50 rounds', 'bow': 'classic', 'knife': 'wakanda steel'}}



for feature, description in player_army.items():
   print("\n" + "Here is the detail of the key feature of your player: "
+ feature.title() + ".")
   desc = description['revolver'] + " " + description['grenade']
```

```
    desc1 = description['bow']

    print("\t Details of Guns: " + desc.title())
    print("\t Details of Desserts: " + desc1.title())
```

```
Here is the detail of the key feature of your player: Gun.
Details of Guns: One Round Four
Details of Desserts: Classic
```

First of all, I defined the dictionary as player_army. The value associated with the keys is generally a dictionary that would include the information about each player. Python will store each key into the variable that I have created.

Chapter Seven

User Input & While Loops

Python programs are user-oriented, which means that programmers write them and use them to facilitate non-tech people. Each program solves a particular problem by doing some complex calculation etc. You may need a program to determine if a person has reached a particular age to enter the golf club. A python program can help you sort out people based on their age.

You can create a program that prompts users to enter their age. After receiving the age number, the program will display a message to the user as to whether they are eligible to join the club or not. This function is known as the input() function of Python. You have to set up the minimum age number in the program. When a user enters a number through the input() function, Python will compare that number to the one you have already fed to the program and display the message accordingly.

In this chapter, I will explain how you can use the input() function to build a program that interacts with users.

Input() Function

The input() function is quite useful because it allows you to interact directly with users through Python programs. Think about a company that is looking to automate the buying process. The input() function will let you create a program that takes input from users and reacts to that input by prompting your customers to take a certain action. The action can be buying one or more products from your e-commerce store. Let us build a program that allows users to write some text and then see the same text on their screens.

```
msg = input("This is a parrot program. Whatever you say, I will
repeat.")
print(msg)

This is a parrot program. Whatever you say, I will repeat.
```

You have to fill in the input() function with a single argument. If we analyze this example, we will know that Python ran through the first line of code to display the first message to the user. The user is prompted to hit Enter button. When he enters the button, his response is stored inside a variable. In return, Python will print the output to the user. The following example will further explain how the input() function helps programmers collect data from a large user base. The input() function, in practical form, is used by corporate companies that mostly maneuver with user data to run and reform their business.

```
msg = input("I want to enter your name: ")
print("Hi " + msg + ", please submit your documents and move in
the hall")
=== RESTART: C:/Users/saifia computers/Desktop/python.py ==
I want to enter your name: John
Hi John, please submit your documents and move in the hall
>>>
=== RESTART: C:/Users/saifia computers/Desktop/python.py ==
I want to enter your name: Adam
Hi Adam, please submit your documents and move in the hall
>>>
```

I have added a space to the end of my prompts right after the colon. This will separate this prompt from the user's response and make it clear where the users should enter the text.

You can choose to compose longer prompts, but you may need to edit your code and write it again if you must do that. Longer prompts must be channeled into a variable for storage. Once stored, they must be passed toward the input() function. In this way, you can create longer prompts that span over multiple lines,

and you will also be able to keep the code clean even after writing complex prompts.

```
msg = input("I want to enter your name to personalize a message for you.")
msg += "\nPlease enter your name: "

user_name = input(msg)
print("Hi " + user_name + ", please submit your documents and move in the hall")
```

```
I want to enter your name to personalize a message for you.

Please enter your name: John
Hi John, please submit your documents and move in the hall
```

You can see that I have built a multiline message in this example. In the second line of code, I used += takes a string that I stored in the prompt and added the new string to the end. This prompt will now span a couple of lines with some space created after the question mark to add clarity to the code.

In this example, I will determine whether people have the required age to ride the dragon coaster.

```
msg = input("Please tell me your height so that I can estimate whether you can ride the coaster or not.")
msg = int(msg)

if msg >= 150:
   print("\nYou have the required height to ride the dragon coaster!")
else:
   print("\nI am sorry. You are not tall enough to take the ride. Come back later.")
```

```
Please tell me your height so that I can estimate whether you can ride the coaster or not.55

I am sorry. You are not tall enough to take the ride. Come back later.
```

```
>>>
Let us see if the user enters the required age.
msg = input("Please tell me your height so that I can estimate
whether you can ride the coaster or not.")
msg = int(msg)

if msg >= 150:
   print("\nYou have the required height to ride the dragon
coaster!")
else:
   print("\nI am sorry. You are not tall enough to take the ride. Come
back later.")

Please tell me your height so that I can estimate whether you can
ride the coaster or not.160

You have the required height to ride the dragon coaster!
>>>
```

There is one problem with this code. If you press enter without entering the age right away, you will return an error on the display. See what type of error you will witness.

Please tell me your height so that I can estimate whether you can ride the coaster or not.

```
Traceback (most recent call last):
  File "C:/Users/saifia computers/Desktop/python.py", line 2, in
<module>
   msg = int(msg)
ValueError: invalid literal for int() with base 10: ''
>>>
```

You can see that I didn't enter the age and got an error in return.

While Loops

The for loops take a collection of several items and execute the code once for all items. In contrast, the while loop will run as long as a specific condition stands true. I will start from the basics. The following code will calculate numbers from 1 to 8.

```
present_num = 1
while present_num <= 8:
   print(present_num)
   present_num += 1

1
2
3
4
5
6
7
8
```

I will start counting from 1 by setting up the value at 1. Then I will let the while loop run as long as the condition is true, which means the number is less than or equal to 5. The code in the loop prints the value of the present_num and then adds 1 to the value with present_num += 1. The operator += is the shorthand for the present_num = present_num +1.

The loop will be repeated as long as the condition stands true. As 1 is less than 8, Python prints 1 and then adds 1, making the presently printed number 2. Then it goes on like that and prints numbers until 8. The programs you are using on an everyday basis are likely built-in while loops. For example, a game will need the while loop to keep going on as soon as you press the quit button. Python programs would not be fun if they ceased to run before we ask them to or would keep running after we quit them. That's why while loops appear to be quite useful.

Quit Button

You can keep a program running as long as you want to by integrating it inside the while loop. I will define the quit value and keep our program running if the user does not enter the quit value. This program is very useful for games. Just imagine a game that refuses to close. What happens if a user starts it but cannot shut it down. Would the game be a success, or will it just have a

real bad end? I think the second answer is most viable. The game will not run a furlong amidst high competition because users will have to unplug their PCs to shut down the game. That's why you should not forget to add the quit feature in the while loop. It will help you control the program by letting it run as long as you want and then quitting it at will.

```
msg = "\nThis is a parrot program. Whatever you say, I will repeat it
for you."
msg += "\nPlease enter 'quit' to close the program: "

msg1 = ""
while msg1 != 'quit':
   msg1 = input(msg)
   print(msg1)

=== RESTART: C:/Users/saifia computers/Desktop/python.py ==
This is a parrot program. Whatever you say, I will repeat it for you.
Please enter 'quit' to close the program: I am a florist.
I am a florist.

This is a parrot program. Whatever you say, I will repeat it for you.
Please enter 'quit' to close the program: I love to watch UFC.
I love to watch UFC.

This is a parrot program. Whatever you say, I will repeat it for you.
Please enter 'quit' to close the program: quit
quit
>>>
```

You can see that I have used the program twice and then I entered quit to exit. You can use any keyword here instead of quit, and it should work just fine. I will change the keyword from quit to q to simplify and speed up the process of quitting. See how it is done.

```
msg = "\nThis is a parrot program. Whatever you say, I will repeat it
for you."
msg += "\nPlease enter 'q' to close the program: "

msg1 = ""
```

```
while msg1 != 'q':
   msg1 = input(msg)
   print(msg1)

This is a parrot program. Whatever you say, I will repeat it for you.
Please enter 'q' to close the program: q
q
```

When the while loop runs for the first time, the msg1 is an empty string, so Python immediately starts the loop. At msg1 = input(msg), Python will display the prompt message and then wait for the next entry by the user. Whatever is entered is immediately channeled to the msg1 and stored there. It also is printed on the screen. Then Python reevaluates the condition inside the while statement. If the user has not yet entered q, it will display the prompt message and keep the program running. When the user enters ‘q’ in the program, Python will stop executing the code and end the program.

There is another problem that you have to deal with. The program displays the quit message such as ‘quit’ or ‘q’ as a real message. If we can change the code, we can get rid of it. I will change the code and make sure that the word quit or alphabet q are not displayed. The user should be able to quit seeing any of these messages.

```
msg = "\nThis is a parrot program. Whatever you say, I will repeat it
for you."
msg += "\nPlease enter 'q' to close the program: "

msg1 = ""
while msg1 != 'q':
   msg1 = input(msg)

   if msg1 != 'q':
      print(msg1)

=== RESTART: C:/Users/saifia computers/Desktop/python.py ==
This is a parrot program. Whatever you say, I will repeat it for you.
```

```
Please enter 'quit' to close the program: Brazil houses largest
Amazon rainforests.
Brazil houses largest Amazon rainforests.
This is a parrot program. Whatever you say, I will repeat it for you.
Please enter 'q' to close the program: Glaciers are packed up with
the purest water on the earth.
Glaciers are packed up with the purest water on the earth.
This is a parrot program. Whatever you say, I will repeat it for you.
Please enter 'quit' to close the program: q
>>>
```

Flag

The program in the previous example did several tasks while also keeping the condition true. In most complex programs, there may come times where more than one event can cause the program to stop running. This can be destructive for your code. In a game, many events can lead to the end of the game. For example, a player runs of bullets and guns. He runs out of time, or he kills all enemies. The game needs to end if any of these events occur. If potentially possible events fail to end a program, testing all the conditions inside a while statement will get complicated.

A program should run as long as the conditions are true, and you may define a variable that will determine if the program is active or not. The variable, namely flag, will act as a signal to the program. You can write the programs to run as the flag is fixed at True and then stop running when different events will set its value to false.

I will use the same program to demonstrate how you can use the flag in the code and how it will affect the program's output.

```
msg = "\nThis is a parrot program. Whatever you say, I will repeat it
for you."
msg += "\nPlease enter 'q' to close the program: "

active = True
while active:
   msg1 = input(msg)
```

```
    if msg1 == 'q':
        active = False
    else:
        print(msg1)
```

```
This is a parrot program. Whatever you say, I will repeat it for you.
Please enter 'q' to close the program: Deep-sea diving is good for
mental health.
Deep-sea diving is good for mental health.

This is a parrot program. Whatever you say, I will repeat it for you.
Please enter 'q' to close the program: q
```

The if statement in the while loop will check the value of msg1. If a user enters ‘q,’ the active value is set to False, and the loop stops. If the user enters any value other than q, it keeps running. You might be wondering that the output is the same as the previous example. It was meant to be like that. It would be easy to add a couple of more tests. You can display a game over message in your game when a user ends the game.

Break Statement

If you want to exit the while loop in the middle of a code, you can do that by using the break statement. It will stop right there without executing the rest of the code. The break statement will regulate the flow of the program. You can use it to control different lines of code. By using it, you will ensure that the program executes the code as per your will. I will create a new program and add the break statement to see how it works and how you can use it in your programs.

```
msg = "\nI want you to enter the names of English literary figures."
msg += "\n(Please enter 'q' to close the program.) "

while True:
    msg1 = input(msg)
```

```
    if msg1 == 'q':
        break
    else:
        print("I love the works of " + msg1.title() + "!")
```

```
I want you to enter the names of English literary figures.
(Please enter 'q' to close the program.) John Donne
I love the works of John Donne!

I want you to enter the names of English literary figures.
(Please enter 'q' to close the program.) Christopher Marlowe
I love the works of Christopher Marlowe!

I want you to enter the names of English literary figures.
(Please enter 'q' to close the program.) Alexander Pope
I love the works of Alexander Pope!

I want you to enter the names of English literary figures.
(Please enter 'q' to close the program.) q
```

If a loop starts with while True, it is meant to run on end until it hits upon the break statement. The loop will continue until the user enters ‘q.’ When Python finds ‘q’ in the prompt, it will activate the break statement that runs its course and stops the program from any further execution.

Continue Statement

The continue statement works with the break statement. Instead of breaking the loop, you can add the continue statement to your code and return to the start of your loop based on the conditional test results. Let see how I add the continue statement to the following program.

```
my_num = 0
while my_num < 8:
    my_num += 1
    if my_num % 2 == 0:
        continue
```

```
  print(my_num)
```

```
1
3
5
7
```

While Loops, Lists & Dictionaries

I have so far worked with just a single piece of information that users entered as input. I then printed that input on the screen of the interpreter. Next time through the while loop, I will receive an input value that would respond to the same. To keep track of different information and users, I will have to use different dictionaries and lists to pair up with the while loops. The for loop does not work well with dictionaries and lists.

Moving Items

I will form two lists of graduated students and ungraduated students from a university. After the verification process of students, I will shift students from the ungraduated students to graduated students. The While loop will help pull students from the ungraduated list to the graduated list. The code for the program takes the following shape.

```
ungraduated_students = ['john', 'abe', 'jasmine', 'asim']
graduated_students = []

while ungraduated_students:
  my_students = ungraduated_students.pop()

  print("I am verifying the students' degrees: " +
my_students.title())
  graduated_students.append(my_students)

  print("\nHere is the list of the graduated students:")
  for graduated_student in graduated_students:
    print(graduated_student.title())
```

```
I am verifying the students' degrees: Asim

Here is the list of the graduated students:
Asim
I am verifying the students' degrees: Jasmine

Here is the list of the graduated students:
Asim
Jasmine
I am verifying the students' degrees: Abe

Here is the list of the graduated students:
Asim
Jasmine
Abe
I am verifying the students' degrees: John

Here is the list of the graduated students:
Asim
Jasmine
Abe
John
```

I started with a list of ungraduated students, and then using the while loop, I shifted all the students from the ungraduated list to the graduated list. I did this one by one. The while loop helped me flawlessly achieve this objective.

Removing Instances

You can use the remove() function to remove different values from the list. You can remove different instances of a single value from a list by using the while loop. The while loop will find each instance and kick it out of the loop.

```
fruits = ['apple', 'peach', 'guava', 'apple', 'melon', 'apple', 'grapes']
print(fruits)

while 'apple' in fruits:
   fruits.remove('apple')
```

```
print(fruits)

['apple', 'peach', 'guava', 'apple', 'melon', 'apple', 'grapes']
['peach', 'guava', 'melon', 'grapes']
```

I created a list that contained different names of fruits. After I had printed the list, Python entered the while loop because it found the value ‘apple’ at least once. When Python entered the loop, it removed the first instance and returned to the while loop. Then it once again entered the while line and removed another instance. It kept repeating the circle until it removed the last instance of the word ‘apple.’

Filling Up Dictionaries

You can create an empty dictionary and fill it up with user input as you proceed with the program's functioning. You create prompts for as much input as you need with the help of the while loop. Let us create a program that receives user input and stores it in a dictionary. I will store the data that I will gather inside a dictionary.

```
data = {}

survey = True

while survey:
   name = input("\nPlease enter your name. ")
   datum = input("What is your favorite tourist destination? ")
   data[name] = datum

   repeat = input("Would like to pass on the survey paper to another
person?")
   if repeat == 'no':
      survey = False
print("\n------Here are the results of the survey------")
for name, datum in data.items():
   print(name + " loves to visit " + datum + ".")
```

Please enter your name. John
What is your favorite tourist destination? Greece
Would you like to pass on the survey paper to another person?yes

Please enter your name. Adam
What is your favorite tourist destination? Italy
Would you like to pass on the survey paper to another person?yes

Please enter your name. Jasmine
What is your favorite tourist destination? Spain
Would you like to pass on the survey paper to another person?no

------Here are the results of the survey------
John loves to visit Greece.
Adam loves to visit Italy.
Jasmine loves to visit Spain.

Chapter Eight

Python Functions

This chapter will walk you through Python functions that are blocks of code specifically designed to perform a particular job. When you perform a specific task defined as a function, you will call the name of that function responsible for it. If you have to perform the same task more than one time in the same program, you need not write the same code over and over again. All you need is to call that function that handles the job, and Python will execute the entire code associated with that function. You will find that the use of functions will make the programs easier to handle.

This chapter will walk you through the information you need to write functions that display information and process data. There may be many other key jobs that your functions may perform. I will explain how you can store functions in separate files known as modules to help organize the major program files.

```
def greetings():
   print("Hello, how are you dear?")

greetings()

Hello, how are you dear?
```

This is an example of a simple function. The example explains the structure of the function. The first line uses the def keyword to inform Python that a function is being defined. This is a function definition that tells Python what the name of the function is. Here the name is greetings(). There is no extra information needed at the moment to ensure the smooth flow of the function. Its job is

merely to print greetings for the function. That's why the parentheses are empty. The definition ends on a colon.

The indented lines make up the body of the function. The line that contains the print statement is the real line of code in the entire structure. When you want Python to execute the code, you simply write a function call by writing the function's name and adding to it the parentheses.

Once you have created a function, you can pass on different snippets of information to that function. I will use the same function and pass on different pieces of information to see how the function receives them and uses them in the code.

```
def greetings(yourname):
   print("Hello, " + yourname.title() + " how are you? Have some
tea!")

greetings('Jasmine')
greetings('Johnny')
greetings('Stark')

Hello, Jasmine how are you? Have some tea!
Hello, Johnny how are you? Have some tea!
Hello, Stark how are you? Have some tea!
```

Arguments & Parameters

When I passed the names to the functions, I used the parameter feature of the function. The name is the parameter of the function. The value that I filled into the function when I made the function call is an example of the function argument. You must pass information to the function in the form of arguments and parameters. Some people interchangeably treat parameters and arguments. However, this is not appropriate. You can pass on multiple parameters to a function. Also, you can use several ways to do that. One of the top methods is to use positional arguments that need to be in a coherent order in which the parameters were written.

When you make a function call, Python needs to match arguments inside the function call with one parameter in the definition of that function. The simplest way is usually based on the order in which you have provided the arguments. The values that you have matched up in this way are labeled as positional arguments. See the following example.

```
def desc_pet(pet_kind, name):
   print("\nOur house has a " + pet_kind + ".")
   print("\nWe have named our " + pet_kind + " as " + name + ".")

desc_pet('eagle', 'Gabriel')
```

```
Our house has a eagle.

We have named our eagle as Gabriel.
```

The output of the code describes a pet with name and kind. You can make multiple function calls to describe more than one animal through the same function. You don’t have to write new code for that. All you will do is pass on new arguments to the parameters that we have defined. See the following example to understand thoroughly.

```
def desc_pet(pet_kind, name):
   print("\nOur house has a " + pet_kind + ".")
   print("\nWe have named our " + pet_kind + " as " + name + ".")

desc_pet('eagle', 'Gabriel')
desc_pet('tiger', 'John')
desc_pet('cat', 'David')
```

```
Our house has a eagle.

We have named our eagle as Gabriel.

Our house has a tiger.

We have named our tiger as John.
```

```
Our house has a cat.

We have named our cat as David.
```

Multiple function calls are an efficient way to get the work done. You can easily produce multiple results with the same code.

Well, you might have thought that functions are easy. However, this is not the case if you miss out on the order of the arguments. Python does not differentiate between the right and the wrong orders. It just takes an argument and connects it with the parameter. If you write the arguments in the wrong order, it is highly likely that you produce the wrong output and ends up getting frustrated. See how a wrong order can ruin your code.

```
def desc_pet(pet_kind, name):
   print("\nOur house has a " + pet_kind + ".")
   print("\nWe have named our " + pet_kind + " as " + name + ".")

desc_pet('Gabriel', 'eagle')
desc_pet('John', 'tiger')
desc_pet('David', 'cat')
```

```
Our house has a Gabriel.

We have named our Gabriel as eagle.

Our house has a John.

We have named our John as tiger.

Our house has a David.

We have named our David as cat.
```

However, Python also offers a solution to this problem. If you are worried about the order of the arguments, you can use keyword arguments. A keyword argument is like a name-value pair that you pass on to a function. You should link the name and the value

inside the argument so that there is little confusion left when you pass on the argument.

```
def desc_pet(pet_kind, name):
   print("\nOur house has a " + pet_kind + ".")
   print("\nWe have named our " + pet_kind + " as " + name + ".")

desc_pet(name = 'Gabriel', pet_kind = 'eagle')
desc_pet(name = 'John', pet_kind = 'tiger')
```

```
Our house has a eagle.

We have named our eagle as Gabriel.

Our house has a tiger.

We have named our tiger as John.
```

Default Values

Python also takes in some default values. You can define the value of your choice to default values for each parameter. If you fail to provide arguments to a function call, it will pick up the default values and produce the results. This helps in multiple function calls where you can miss out on one call and then let the function use the default values to finish the function call. In the following example, I will define the default values for a function and use them in the function call.

```
def desc_pet(pet_kind = 'peacock', name = 'Rosie'):
   print("\nOur house has a " + pet_kind + ".")
   print("\nWe have named our " + pet_kind + " as " + name + ".")

desc_pet(name = 'Gabriel', pet_kind = 'eagle')
desc_pet()
desc_pet(name = 'Lucy')
desc_pet(name = 'John', pet_kind = 'tiger')
```

```
Our house has a eagle.
```

```
We have named our eagle as Gabriel.

Our house has a peacock.

We have named our peacock as Rosie.

Our house has a peacock.

We have named our peacock as Lucy.

Our house has a tiger.

We have named our tiger as John.
```

If you do not fill in the default values and still you make an empty function call, you will receive an error. Here is how the error looks like.

```
def desc_pet(pet_kind , name):
   print("\nOur house has a " + pet_kind + ".")
   print("\nWe have named our " + pet_kind + " as " + name + ".")
desc_pet()

Traceback (most recent call last):
  File "C:\Users\saifia computers\Desktop\python.py", line 4, in
<module>
   desc_pet()
TypeError: desc_pet() missing 2 required positional arguments:
'pet_kind' and 'name'
>>>
```

You can use a function inside a loop.

```
def formatted_nm(ft_nm, lt_nm):
   """This code will return the complete name in a neatly formatted
manner."""
   complete_name = ft_nm + ' ' + lt_nm
   return complete_name.title()
# This is an infinite loop!
while True:
   print("\nPlease tell the name:")
   fst_name = input("My first name: ")
   lst_name = input("My last name: ")
```

```
    the_formatted_name = formatted_nm(fst_name, lst_name)
    print("\nHello, how are you, " + the_formatted_name + "!")

Please tell the name:
My first name: Eva
My last name: Jasmine

Hello, how are you, Eva Jasmine!

Please tell the name:
My first name: John
My last name: Adams

Hello, how are you, John Adams!

Please tell the name:
My first name:
```

By looking at the code, you can tell that the code is not going to end easily. That's why we need to add the break statement to kill the code whenever you need to. The break statement will help you exit the while loop whenever you need to.

```
def formatted_nm(ft_nm, lt_nm):
    """This code will return the complete name in a neatly formatted
manner."""
    complete_name = ft_nm + ' ' + lt_nm
    return complete_name.title()
# This is an infinite loop!
while True:
    print("\nPlease tell the name:")
    print("(please enter 'q' to exit the loop.)")


    fst_name = input("My first name: ")
    if fst_name == 'q':
        break


    lst_name = input("My last name: ")
```

```
    if lst_name == 'q':
        break



    the_formatted_name = formatted_nm(fst_name, lst_name)
    print("\nHello, how are you, " + the_formatted_name + "!")

============= RESTART: C:\Users\saifia
computers\Desktop\python.py =============

Please tell the name:
(please enter 'q' to exit the loop.)
My first name: John
My last name: q
>>>
============= RESTART: C:\Users\saifia
computers\Desktop\python.py =============

Please tell the name:
(please enter 'q' to exit the loop.)
My first name: John
My last name: Adams

Hello, how are you, John Adams!

Please tell the name:
(please enter 'q' to exit the loop.)
My first name: q
>>>
```

I produced two outputs to show how you can end at two different points because I have added two break statements to the code.

Chapter Nine

Classes

Python classes are all about object-oriented programming. By using the classes, you can create objects that are modeled on real-life objects. When you write a class, you define the behavior of the entire category of objects that you have to model.

Each object in a class is equipped with a behavior. You can give each object a bunch of unique traits as per your will. You may be amazed at how well you can model some real-world situations in your codes. The process of creating an object inside of a class is called instantiation.

Creating Eagle Class

In the following code, I will show you how you can create an eagle class in Python. I will add some instances to the class and make the object perform some key functions.

```
class Bald_eagle():
  """I am modeling an eagle in the following code."""
  def __init__(self, eagle_name, eagle_age, eagle_color):
    """Here I am initializing the name and age attributes of the
eagle."""
    self.eagle_name = eagle_name
    self.eagle_age = eagle_age
    self.eagle_color = eagle_color
  def sitting(self):
    """Now I am Simulating the eagle to sit on the top of the
mountain."""
    print(self.eagle_name.title() + " is sitting at the moment on the
top of the mountain.")
  def flying(self):
    """The eagle is now flying through the clouds."""
```

```
        print(self.eagle_name.title() + " is flying across the clouds!")
    def preying(self):
        """The eagle is now attacking a rabbit."""
        print(self.eagle_name.title() + " is now preying upon a wild
rabbit along the bank of a lake!")

eagle1 = Bald_eagle('Tin Tin', 5, 'brown')
print("The name of my eagle is " + eagle1.eagle_name.title() + ".")
print("The age of the eagle is " + str(eagle1.eagle_age) + ".")
print("The color of the eagle is " + eagle1.eagle_color.title() + ".")
eagle1.sitting()
eagle1.flying()
eagle1.preying()

eagle2 = Bald_eagle('Teem', 6, 'brown')
print("The name of my eagle is " + eagle2.eagle_name.title() + ".")
print("The age of the eagle is " + str(eagle2.eagle_age) + ".")
print("The color of the eagle is " + eagle2.eagle_color.title() + ".")
eagle2.sitting()
eagle2.flying()
eagle2.preying()

eagle3 = Bald_eagle('Sim', 7, 'black')
print("The name of my eagle is " + eagle3.eagle_name.title() + ".")
print("The age of the eagle is " + str(eagle3.eagle_age) + ".")
print("The color of the eagle is " + eagle3.eagle_color.title() + ".")
eagle3.sitting()
eagle3.flying()
eagle3.preying()

The name of my eagle is Tin Tin.
The age of the eagle is 5.
The color of the eagle is Brown.
Tin Tin is sitting at the moment on the top of the mountain.
Tin Tin is flying across the clouds!
Tin Tin is now preying upon a wild rabbit along the bank of a lake!
The name of my eagle is Teem.
The age of the eagle is 6.
The color of the eagle is Brown.
Teem is sitting at the moment on the top of the mountain.
Teem is flying across the clouds!
```

```
Teem is now preying upon a wild rabbit along the bank of a lake!
The name of my eagle is Sim.
The age of the eagle is 7.
The color of the eagle is Black.
Sim is sitting at the moment on the top of the mountain.
Sim is flying across the clouds!
Sim is now preying upon a wild rabbit along the bank of a lake!
```

You can see that I have created three objects from the same class. Technically, it means that I have created three different objects from the same class.

Conclusion

Now that you have made it to the end of the book, I am sure that you have been equipped with Python programming basics. Python programming can be a piece of cake if you master the basics like data types, loops, functions, conditionals, etc. Once you have aced these, you can go on to master object-oriented programming, of which I have given you a slight glimpse in this book.

The best course of action onward is to take your computer and practice the codes as much as you can so that you get familiar with different types of outputs, errors, and other complications. Unless you get familiar with the complications and errors, you are unlikely to make sufficient progress. It is the errors and frustratingly wrong outputs that compel programmers to delve deeper into the code and understand why something got wrong in the first place.

Once you start questioning yourself as to what happens to something in the first place, you can consider yourself a programmer. Don't worry if you don't get it right and if things take time to be corrected. This means you are learning new codes and new techniques. Also, the most important is not to stop experimenting. The more you try new methods and produce more examples, the better you will learn Python.

References

Decorators with parameters in Python . (2021, July 7). GeeksforGeeks. https://www.geeksforgeeks.org/decorators-with-parameters-in-python/?ref=lbp

Python data types . (n.d.). W3Schools Online Web Tutorials. https://www.w3schools.com/python/python_datatypes.asp

Python Tuples . (n.d.). W3Schools Online Web Tutorials. https://www.w3schools.com/python/python_tuples.asp

Python while loop . (2021, May 14). GeeksforGeeks. https://www.geeksforgeeks.org/python-while-loop/?ref=lbp

Taking input from console in Python . (2020, November 27). GeeksforGeeks. https://www.geeksforgeeks.org/taking-input-from-console-in-python/?ref=lbp

PYTHON
PYTHON AUTOMATION TECHNIQUES AND WEB SCRAPING

Andy Vickler

Introduction

If you are keen on learning how to program using Python, this is the book for you. Python is an easy language to learn since it is a high-level programming language. Python is an open-source language and can be downloaded from the official website. The developers have been working on pushing new versions regularly, with a few updates. The latest version is 3.9.0, but some programmers still use the older versions because they may have applications or software developed using the earlier versions.

Python is not only simple to learn but is very easy to read and write. You will not take too long to learn the code, and therefore, it becomes easier for you to build applications and software easily. If you are keen to learn the language, use the book as your guide.

This book has all the information you need about Python. You will learn how to scrape data from the web and automate some tasks using Python. Before you learn how to do this, you need to understand the basics of the programming language. This book has information about how you should install Python on your system, data types and variables, data structures, and more.

Bear in mind you need to learn the language before you use it to develop applications. It is an easy language to learn, so spend some time to understand it better. Using this language, you can solve numerous problems and change the way you write code. The book has exercises at the end of some chapters to help you implement what you have learned in the chapter. I recommend you try to solve the problem yourself before looking at the solutions at the end of the book.

It is only when you practice that you can determine how well you have understood the concepts. You can develop numerous

applications, scrape any information from the Internet and automate different tasks using Python. This can only happen if you know what function, method, or library to use and when to use them in the code.

Thank you for purchasing the book. I hope you gather the information you are looking for.

Chapter One

An Introduction to Python

Before we dive into learning more about how you can use Python for web scraping, it is important to understand the basics of the language.

Running Python

Python is a language that can be installed on your system, regardless of the operating system you use. This means you can use it on Mac OS X or OS/2, Linux, Unix, and Windows. Most Mac OS X or GNU/Linus operating systems come pre-installed with Python. It is recommended to use this type of system since it already has Python set up as an integral part. The examples in this book can be used on any operating system since the software you use will be the same on any operating system.

Installing on Windows

If you use a system with the Windows operating system, you need to configure some settings in the system when you install Python. You can only run the examples on your system once you configure these settings. If you are unsure of what to do, use the following resources:

- http://wiki.python.org/moin/BeginnersGuide/Download
- http://www.python.org/doc/faq/windows/
- http://docs.python.org/dev/3.0/using/windows.html

The first step is to download the official Python installer. An alternative version is to use an Itanium or AMD machine that is available at http://www.python.org/download/ . This file, which

has a .msi extension, must be saved at a location that you can find easily. Double-click on this file to begin the installation of Python. The installation wizard will open and take you through the necessary steps to install the language. If you are not sure which settings to choose, let the installation wizard set the default settings.

Installing on Other Systems

If you must install the language on any other operating system, it is best to download the latest version of Python, which is version 3.9.0. The following link has instructions for Unix-like systems, such as AIX and Linux:

- http://docs.python.org/dev/3.0/using/unix.html

If you use Mac OS X, use the instructions in the following links:

- http://www.python.org/download/mac/
- http://docs.python.org/dev/3.0/using/mac.html

Choosing the Right Version

Every installer has a different version of Python. If you are unsure of where to look for the version number, you need to read the number that comes after Python. Different websites will have different version numbers. The python team released version 2.6 while it released version 3.0 since some people may still want to stick to version 2 of Python. They preferred to continue to write code the old way but still want to benefit from general fixes and some of the new features introduced in version 3.0.

The Python language evolves continuously, and the current version used by most programmers is version 3.9.0. However, some may choose to run older versions. In this book, we will refer to all versions of 3.0 as version 3. This version includes several changes to the programming language incompatible with version 2.0. Do not worry about programming using different versions of

Python, as there is a very small difference between the different versions in the language or syntax.

There could be some differences running Python on different operating systems, and I will point this out in the book if it is necessary. Otherwise, the codes in the book will work in the same way across different operating systems. This is one of the many good points of Python. In this book, we will look at writing programs using Python. If you want more information about Python, read the documentation written by the Python developers. It is available at http://www.python.org/doc/ .

Learning While Having Fun

Unfortunately, people think programming is something serious and not something fun to do. They underestimate the fun people can have while programming. People only learn a subject well when they have fun with it. Developing software using Python is often an engrossing and enjoyable experience, partly because you can test out your changes as soon as you have made them without performing any intermediary steps. Unlike other programming languages, Python also takes care of the different tasks that happen in the background. This will make it easier for you to focus only on the design of the program and the code you are using without worrying about background jobs. This makes it easy for the user to stay in the creative flow and continue developing and refining the program.

The language is extremely easy to read, and it is one of the many languages that use syntax closer to English. This makes it easier for you to spend less time understanding the code and more time on the program's structure. You can also spend some time working on improving the code. This would mean you could expand the code to encompass different aspects.

Another aspect of Python is you can be used to complete tasks, regardless of the size of that task. You can develop simple text-driven or numerical-based programs as well as major graphical

applications. Python has some limitations, and it is important to understand them to find a way to work around those limitations and mistakes.

Choosing to Code

Using a Text Editor

Most people choose to write or create Python scripts using plain text editors that have basic programming features. Programs such as Kate, NEdit, gedit, BBedit, and notepad (preferably notepas2/++) are the best options to choose from. Multiple editors offer specific enhancements for programmers, such as syntax highlighting, which is useful for showing coding errors immediately as you type. If you are an old-school programmer, you may want to use Vi or Emacs. Python does not require any specific software that needs to be used to create or design the code. That choice is up to you. Avoid using OpenOffice, Word, or other word processing software to do this. Otherwise, it will mess you up.

Using an Integrated Development Environment

IDEs or integrated development environment is a graphical interface that uses a lot of features, which are designed to make it easier for you to work with Python. Bear in mind that the code needs to be tied the same way, but you can use the same methods or functions when you code on one application. There are numerous programming environments you can use, and some also provide reminders and shortcuts.

There are numerous IDEs specific to Python, and some popular applications include

1. Eric
2. IDLE
3. Geany
4. DrPython

5. SPE

The second option is an environment that comes installed with Python. You can also use some programming environments, such as Bluefish and other hosts, which allow you to work on different environments. We are not looking at the use of IDEs in this book or other distributions of Python that you cannot use.

Every approach would need to use a different environment. The book contains a lot of information about Python and its functions, and you can use them to improve your knowledge of Python. The book uses simple approaches and tools that make it easier for you to work with Python on different operating systems. You can write your scripts either on the Python Interactive Shell or any text editor.

Getting Started

When you spend time writing a new program or code, bear in mind that it all begins with a problem. Only when you know the problem statement can you develop or design the code to overcome the problem. Before you write code for anything, develop an idea of what you would like to create and the problem you are looking to solve. This will help you develop a fair idea of how you would like to solve the problem.

In this book, we will look at how you can develop software and the process to follow. This is a step that most people will need to learn separately. Most programming guides usually switch to the intricacies of the language and focus on developing code, making it difficult for a beginner to know how to understand the code and what needs to be done to fix it. It is important to understand the principles of designing any software that can dramatically speed the process, thereby improving or creating new software. This helps to ensure there are no important details that are missed out during the coding.

In the next few chapters, you will learn how to create and build ideas and designs. It is important to construct the basic units of the code using data, numbers, and words. You will also learn how to manipulate these inputs to refine the code. It is important to learn how to compare different sets of data to make informed decisions. You will also learn how to refine any program or code design by breaking the code down into smaller and easy-to-understand chunks of code. Using this information, you can improve your understanding of the language and convert your ideas into functional and easy to develop programming scripts.

Using Python to Create Files

Python is a self-documenting language, but this does not mean it will write the documentation for you. It does not know what each function block can do. So, you need to add some information or blocks of text called documentation strings or docstrings to provide information about the code. These strings will then show up in an online help utility. This information can easily be turned into web pages to provide a useful reference. We will look at this later in the book and help you understand how to work on writing code to create your own files.

Chapter Two

Data Types and Variables

It is important to understand certain aspects of Python before you dive into writing code to scrape information from the Internet. The following are some points to remember:

1. Identifier: An identifier is a part of any variable which has a unit of data
2. Variable: A variable is one that you define in your code that is used in different operations

Identifiers and variables are both maintained in the computer's memory. The values can be changed when you modify the variable and its value stored in the memory. This chapter will look at the different types of variables you can use when writing a program in Python.

Choosing the Right Identifier

Every section of your code is identified using an identifier. The compiler or editor in Python will consider any word delimited by quotation marks, has not been commented out, or has escaped in a way by which it cannot be considered or marked as an identifier. Since an identifier is only a name label, it could refer to just about anything, making sense to have names that the language can understand. You have to ensure that you do not choose a name that has already been used in the current code to identify any new variable.

If you choose a name that is the same as the older name, the original variable becomes inaccessible. This can be a bad idea if the name chosen is an essential part of your program. Luckily,

when you write a code in Python, it does not let you name a variable with a name used already. The next section of this chapter lists out the important words, also called keywords, in Python that will help you avoid the problem.

Python Keywords

The following words, also called keywords, are the base of the Python language. You cannot use these words to name an identifier or a variable in your program since these words are considered the core words of the language. These words cannot be misspelled and must be written in the same way for the interpreter to understand what you want the system to do. Some of the words listed below have a different meaning that will be covered in later chapters.

- and
- class
- del
- for
- break
- None
- assert
- True
- as
- continue
- False
- is
- from
- global
- raise
- not

- or
- nonlocal
- yield
- pass
- except
- return
- else
- def
- while
- with
- finally
- if
- import
- in
- elif
- try
- lambda

Understanding the naming convention

Let us talk about the words that you can use and those you cannot use. Every variable name must always begin with an underscore or a letter. Some variables can contain numbers, but they cannot start with one. If the interpreter comes across a set of variables that begin with a number instead of quotation marks or a letter, it will only consider that variable as a number. You should never use anything other than an underscore, number, or letter to identify a variable in your code. You must also remember that Python is a case-sensitive language, so false and False are two different entities. The same can be said for vvariable, Vvariable, and VVariable. As a beginner, you must make a note of all the

variables you use in your code. This will also help you find something easier in your code.

Creating and Assigning Values to Variables

Every variable is created in the following manner:

1. Declaring the variable
2. Initializing the variable
3. Assigning a value to the variable

You can do this using the assignment operator, which is the equal-to sign. When you must assign a value, you should write the following code:

Variable = value

Every section of the code that performs some function, like an assignment, is called a statement. The part of the code that can be evaluated to obtain a value is called an expression. Let us take a look at the following example:

```
Length = 14
Breadth = 10
Height = 10
Area_Triangle = Length * Breadth * Height
```

Any variable can be assigned a value or an expression, like the assignment made to Area_Triangle in the example above.

Statements should be written on separate lines to ensure the code is documented the right way. Follow the steps below:

1. Define the variables you want to use
2. Assign the values to the variables
3. Create functions and methods to operate on those variables

Using Quotes

In Python, characters describe a single number, punctuation mark, or a single letter. If you need to tell the interpreter that you want a block of text to be displayed as text, you must enclose those characters in quotation marks. You can use quotes in the following ways:

‘A text string enclosed in single quotation marks.’

“A text string enclosed in double quotation marks.”

“’A text string enclosed in triple quotation marks.’”

If text is enclosed in quotes, it is considered the type str (string).

Nesting Quotes

At times, you may need to include some quotation marks in the code. In Python, you can use a different set of quotation marks depending on what you want to represent. Consider the following line of code:

```
>>>text= “You are learning ‘how to’ use nested quotes in Python”
```

Here, the compiler will assume it has reached the end of the string when it reaches only the end of the text at the end of the second set of double quotes in the string above. This indicates that the ‘how to’ phrase is a part of the main string, including the quotes. In this way, you can have at least one level of nested quotes. The best way to understand the use of nested quotes is to experiment with strings.

```
>>> boilerplate = """
... #===(")===#===(*)===#===(")===#
... Egregious Response Generator
... Version '0.1'
... "FiliBuster" technologies inc.
... #===(")===#===(*)===#===(")===#
... """
>>> print(boilerplate) #=== (“) ===#=== (*) ===#=== (“) ===#
```

```
Egregious Response Generator
Version '0.1'
"FiliBuster" technologies inc.
#===(")===#===(*)===#===(")===#
```

This is a useful trick to use if you want to format a whole block of text or a whole page.

Using Whitespace Characters

If a sequence of characters begins with a backslash, it indicates the whitespace characters in the code. For example, '\n' produces a linefeed character different from the '\r' character. You will see the output has moved onto a new line in your output window, where the code begins with /n. If the line begins with /r, the output window will have the output moving to a new paragraph. Learn the difference between how different operating systems use to translate the text.

Unfortunately, the meaning and usage of most sequences are unknown to programmers. You may often want to use \n to shift to a new line. Another useful sequence is \t, which can be used to indent the text by producing a tab character. Most whitespace characters in Python are used only in specific situations.

Sequence	Meaning
\n	New line
\r	Carriage Return
\t	Tab
\v	Vertical Tab
\e	Escape Character
\f	Formfeed
\b	Backspace
\a	Bell

You can use the example below to format the output for your screen:

```
>>> print (“Characters\n\nDescription\nChoose your character\n \
…. \tDobby\n\tElf\n\tMale\nDon\’t forget to escape \’\\\’.”
)
Output:
Characters

Description

Choose your character
Hippogriff
Elf
Dragon
Don’t forget to escape ‘\’.
```

Bear in mind that strings are immutable, and we will learn more about this later in the book.

Chapter Three

Data Structures

Until now, you have learned how to work with different pieces of data, their types, and variables. Using the above information, you can work with real-world data. Bear in mind that this data will usually appear in groups or lumps, and it is best to work with these groups without using repetitive statements in the code. Fortunately, Python provides a variety of data types that can make handling groups of data much simpler.

The following data types are often used in Python:

- Tuples
- Strings
- Lists
- Dictionaries

These elements are called data structures.

1. Strings are characters that are strung together, and these are pieces of text
2. Lists and tuples are groups of data items that are ordered
3. Dictionaries are pairs of keys and values

Tuples, lists, and strings are types of sequences. You can access the elements in the sequences using specific methods, and we will look at them later in this chapter. Another way of addressing or working with these data types is to assess their mutability. Strings and tuples are immutable, meaning we cannot modify an existing string or tuple, although we can use them to create new strings and tuples, respectively. Since lists are mutable, it indicates you can remove or add items to the list.

Items in Sequences

You can fetch an individual item in the sequence using the concept of an index. The index is the position of the variable in the sequence. You can only specify an integer as the index. If you say s[i] in your code where s is the sequence and i is the position of the variable, the compiler will give you the variable at the ith index as the output. You can also create a single character in a string:

```
>>> vegetable = 'pumpkin'
>>> vegetable [0]
'p'
```

Or an item in a list:

```
>>>vegetable = ['squash', 'tomatoes', 'turnips', 'broccoli']
>>>vegetable [1]
'squash'
```

You will notice that indexing is a zero-based method, which means you begin counting at the number zero. If you write index [5], it indicates to the compiler that it should look for the sixth element in the sequence. You can use integers 0 through to the number of elements in the sequence minus one (0 to n – 1) as indices. Negative indices count backward from the end:

```
>>>vegetable [-1]
'broccoli'
```

You can grab a section of your sequence using a slice. This term is often used to obtain multiple items from a sequence. A slice is written using the same notation as an index, but this would be separated using integers. The first value is the inclusive index or the starting point, while the second number can be inclusive or exclusive depending on the type of parenthesis used. So, s [0:5] would mean the slice will begin at the zeroth index and will stop right after the fourth. The fifth value in the sequence is not considered.

The third value is optional, and this only specifies an additional value. This may be negative, so instead of picking out a sequential list, you can retrieve every other or every nth item, and you can also retrieve them backward if you need to. So, s [i: j: step] will allow you to retrieve a value that is present in the slice and starts at I and goes upward.

If you ignore the starting point, the slice will begin at the start of the original sequence. If you leave the end out, the slice will run until the end of the sequence.

Slicing and indexing do not change the sequence of variables, but these methods allow you to create a new sequence, but the actual individual data items are the same. Therefore, if you do choose to modify an individual item in any sequence, you can see the change in the item.

Tuples

Tuples are immutable, and the elements in these are little sealed packets of information. A tuple is specified as a comma-separated list of values, which may be enclosed in parentheses. In certain situations, you may need to use the parentheses. So, if you are unsure of what to do, use the parentheses. The values need not all be of the same type. You can also declare a value as a tuple.

Creating a Tuple

You can create a tuple easily with no items in it. All you need to do is use the round brackets.

```
>>>empty_tuple= ()
```

If you do not want more than one item in the tuple, you should enter the first item followed by a comma.

```
>>>one_item = (‘blue’,)
```

Changing Values in a Tuple

It is difficult to change values in a tuple since these are sealed packets of information that cannot be changed. You can define tuples in situations when you do not have to pass on values from one set to another. If you wish to change the sequence of the data, you should use a list.

List

Lists are comma-separated, ordered lists of items that are enclosed within square brackets. These items do not have to be the same type, and you can have one item in more than one list.

A list can be concatenated, indexed, and sliced the same way you work with any other sequence. It is possible to change individual items in a list, as opposed to immutable strings and tuples. Where a tuple is rather like a fixed menu, lists are more flexible. You can also assign the data to slices, making it easier for you to change the list size and clear that list completely.

Creating a List

You can create a list easily. Consider the example below:

```
>>> shopping_list = [‘detergent’, ‘deodorant’, ‘shampoo’, ‘body wash’]
```

Modifying a List

Using the assignment operator, you can add a new value to the list. Consider the example below:

```
>>> shopping_list [1] = ‘candles’
>>> shopping_list
[‘detergent’, ‘candles’, ‘deodorant’, ‘shampoo’, ‘body wash’]
```

Stacks and Queues

Since lists are ordered data types, you can retrieve and store data items in a specific order. The main models you can use to perform this function are termed stacks and queues.

A stack is a data structure that uses the last in, first out (LIFO) method. It is like how you would discard cards in a game. You put cards on the top of the pile and take them back off the top. Push items onto the stack with list.append () and remove them from the end of the stack using the pop() function. There is no additional index argument, so the pop() function automatically removes the last variable in the stack.

```
>>> shopping_list.append (‘brush’)
>>> shopping_list.pop()
‘candles’
>>> shopping_list
[‘detergent’, ‘deodorant’, ‘shampoo’, ‘body wash’]
```

Another way to do this is to use the first in, first out (FIFO) structure. This is called the queue. This works more like a pipe, where you push items in at one end, and the first thing you put in the pipe pops out of the other end. You can then push the variables or values into the pipe using the append() function. You can retrieve the values in the queue using the pop(0) function. Ensure you use the right index variable to ensure you remove the right variables from the queue.

```
>>> shopping_list.append (‘brush’)
>>> shopping_list.pop(0)
‘detergent’
>>> shopping_list
[‘deodorant’, ‘shampoo’, ‘body wash’, ‘brush]’
```

Dictionaries

A dictionary is synonymous with address books. When you pick up an address book and look for a person’s name, you will get all that person's details. A dictionary is called a key, and any corresponding detail is referred to as the value.

Any key you use in a dictionary should be immutable. This means you would have to use a number, list, tuple, or string. The value of this key can be anything you need. Dictionaries are mutable,

which means you can add, remove, and modify key-value pairs. You need to map the keys to objects, which is why dictionaries are referred to as “mappings.” These remind us their behavior is different when compared to sequences.

You can use dictionaries anywhere you want to store attributes or values. Ensure these attributes or values describe the entity or concept. For instance, use a dictionary to count instances of objects or states. Since every key should have a unique identifier, there cannot be duplicate values for the same key. For this reason, you need to use keys to store the input data items and ensure you leave the value part to store the results of the operations.

Here are a few exercises that will help you master data structures.

1. Write a program to change a tuple.
2. Write a program to delete elements in a List.
3. Write a program to slice a list in Python.
4. Write a program to access the elements in a list
5. Write a program to access elements in a tuple.

Chapter Four

Working with Strings

The commands in the latest version of Python work in the same way as they did in the other versions of Python. There are, however, some important changes between the versions released. The most fundamental change is the rationalization of the string data type.

In earlier versions of Python, a string was not coded as a sequence of single bytes, using the limited American Standard Code for Information Interchange (ASCII) character set to represent text. This 7-bit encoding allows for up to 128 characters, including uppercase and lowercase letters, numbers, punctuation, and 33 invisible control characters.

It is okay to use ASCII to represent any language taken from the Latin script, such as English and most European languages. It is, however, useless when it comes to representing any language that has numerous ideograms or alphabets, such as Mandarin. To deal with these sorts of problems, the Unicode standard was created to cover all written languages. Python 3.9.0 uses Unicode as the default method of encoding.

The STR type is what used to be the Unicode type, and a new type, byte, has been introduced to represent raw text strings and binary data. Earlier versions of Python had different methods to deal with text encoding. Fortunately, all you really need to know is that the str type in Python 3.9.0 supports international characters by default. Therefore, you do not have to do anything extra to write or declare a string in the program.

In earlier versions of Python, the 2.x versions, a print statement was used along with string types to ensure the compiler knew what to print. In the latest version of Python, the print() function is a built-in function that replaces most of the earlier syntax with keyword arguments. To balance this, the old raw_input() is replaced by input(). If you want the older functionality of the input() function, use the format eval(input()).

Splitting Strings

The string data type is immutable, and you may want to split this up often into lists, which makes it easier for you to manipulate the content. To do this, you need to use a delimiter, which is a string or character. You can use these to split the string into sections or units of data. The list will be split up to maxsplit times, so you'll end up with a list that's maxsplit + 1 item long. If you do not specify a separator, the compiler will split the string into sections based on the number of whitespace characters.

```
>>>sentence = 'This is a long sentence'
>>> sentence.rstrip('sentence').split()
['This', 'is', 'a', 'long']
```

Python also uses an alternative string splitting method called the string.partition(sep). This method will return a tuple: (head, sep, tail). Using this method, the compiler will search for the separator within the string, and it will return the part before the separator. If it cannot find the separator, the method will return the original string and two empty strings.

Concatenation and Joining Strings

Using the + operator to join strings together is very inefficient; combined with lots of calls to the print() function (or statement in Python 2), using the + operator can potentially slow your program's execution to a crawl. Python isn't that slow. Often, it works out better to manipulate a list of words and then use string.join(sequence) to return a string that is a concatenation of

the strings in the sequence. This method is the opposite of string.split(): the data you want to manipulate is in the sequence of the argument, and the string that you call the method on is just the string of characters you want to use to separate the items. This could be a space or an empty string.

```
>>> s1=”example”
>>> s2 = “text”
>>> s3 = “ “
>>> s3.join([s1,s2])
‘example text’
```

You must remember that the function string.join() always expects a sequence of strings as the argument.

```
>>> s3 = “-“
>>> s3.join (‘castle’)
‘c-a-s-t-l-e’
```

You also may need to convert different data types into strings by using a sublist.

Editing Strings

A string cannot be edited in one place. You may have understood this by now. Python has numerous methods that can be used to edit strings and obtain different results.

You may need to clean the string up in the beginning before you use it to perform any operations on that string. This is done with the string.strip([chars]) method. If the sequence is found, it returns a copy of the string, with no chars at the start or finish. If no arguments are given, string.strip() will remove whitespace characters by default.

```
>>> sentence = ‘This is a long sentence’
>>> sentence.strip(‘A’)
‘ This is long sentence’
```

How to Match Patterns

The basic string methods in Python are not enough. For instance, you may need to retrieve values that appear within a regular pattern in a piece of text. You do not know what the values will be or just have a rough idea of what they should not be. This is where regular expressions come in. A regex or regular expression is a pattern that can be used to match pieces of text. In the simplest form, a regex is a type of plain string with ordinary characters that match. Regular expression syntax uses additional special characters to recognize a range of possibilities that can be matched. You can use these expressions to perform search and replace operations allowing you to split the text up in different ways using the string.split() function.

Regular expressions are complex, powerful, and difficult to read. Most of the time, you can manage without them, but they are particularly useful when dealing with complex, structured pieces of text. It's best to take regular expressions slowly, learning a bit at a time. Trying to learn the whole regular expression syntax all in one go could be quite overwhelming.

The re module provides a regular expression matching operation. As this module is not built in the language by default, you must first import it into the program before it is used.

```
>>> import rein
```

The module supports both 8-bit and Unicode strings, so it should be possible to recognize any characters that you can type in from the keyboard or read from a file.

The next thing you need to do is construct or write a regular expression string that helps you represent the pattern you should catch. We will use the string from the earlier chapter as an example.

Creating a Regular Expression Object

Using the re.compile(pattern[, flags]) method, you can compile a regular expression pattern into an object. This will return a

pattern object on which you can call all the previous methods. You will not have to use a pattern as an argument. You may want to do this if you need to use this pattern to perform lots of matches all at once. If you compile the pattern into an object, it increases the speed of the pattern comparison process every time it is used. You may also need to use this approach where a function specifically requires a regular expression object to be passed as a part of a pattern string in the main program.

Exercises:

Write a Program to create a string.

Chapter Five

Conditional Statements

In the last few chapters, we looked at how you can use Python to manipulate strings. Python also allows you to simplify calculations. We have also looked in the previous chapter about how you can design your software. Now, it is time to learn how to refine your code. Therefore, using the information you have learned so far, pull out your old scripts and refine them.

Comparing Variables

If you want to generate accurate answers, it is important to learn how to compare these values and specify what the compiler or interpreter needs to do to obtain the desired result. Python, like other programming languages, uses conditional statements that allow you to make decisions. A conditional statement can transform the code or script from just being a list of instructions to a code that the user can use to make their own decisions. It is important to write the correct instructions in your code so the interpreter knows the different activities it needs to perform based on the decision made in the code. For example, let us look at the following pseudocode:

> if a certain condition is true:
> then the following actions must be performed;
> if another condition is true:
> then these actions must be performed.

In the above example, the two pairs of lines are conditional statements. Before we look at these statements, let us understand how you should specify these conditions. Different values can be compared using the following operators:

- !=: Not equal to
- >=: Greater than equal to
- >: Greater than
- <: Less than
- <=: Less than equal to
- >=: Greater than equal to
- ==: Equal to

Using these operators, you can change the way the data types work. The results of these statements would be Boolean operators. The data bits on either side of the operator are called operands and these are the variables that are compared. The comparative operator and the operands together form the conditional expression. It is important to check the conditional expressions and statements to remove any errors when you compare different data types. Some data types cannot be compared to others. Consider the following examples:

```
>>> 7.65 != 6.0
True
>>> 49 > 37
True
>>> -2 < 5
True
>>> -5 <= -2
True
>>> 23.5 > 37.75
False
>>> -5 >= 5
False
>>> 7 < -7
False
>>> 3.2 != 3.2
False
```

You can also use variables in conditional statements.

```
>>> variable = 3.0
>>> variable == 3
True
```

Manipulating Boolean Variables

Before you move onto using different conditional structures used in Python, learn how to use and manipulate Boolean values. Use these values to understand the characteristics of any variable. Most of these operators use the terms NOT, AND, and OR. In the statements below, you will learn more about how to use these statements:

```
>>> a = True
>>> b = False
>>> c = True
```

Let us now look at how to use the logical operators, AND, OR, and NOT.

```
>>> a or b
```

If you use the OR logical operator, the compiler will return the value as TRUE. For the OR operator, either one of the conditions needs to be true.

```
>>> c and e
```

If you use the AND logical operator, the compiler will return the value as FALSE. For the AND operator, both conditions need to be true.

```
>>> not d
```

If you use the NOT logical operator, the compiler will return the value as FALSE since the NOT operator returns the opposite value.

Combine Conditional Expressions

Conditional expressions can be combined to produce complex conditions that use the logical operators AND and OR. Let us take a look at the following conditions:

```
(a < 6) AND (b > 7)
```

This statement will only return True if the value of a is less than 6 and the value of b is greater than 7.

The Assignment Operator

Since you are familiar with the assignment operator (=) that you use to put a value into a variable, let us examine how you can use this operator to assign values to variables. This assignment operator can be used to unpack sequences.

```
>>> char1, char2, char3 = 'cat'
>>> char1
'c'
>>> char2
'a'
>>> char3
't'
```

The assignment operator can also be used to assign different variables with the same value.

```
a = b = c = 1
```

The assignment operator can also be used along with mathematical operators.

```
counter += 1
```

The statement above is interpreted as counter = counter + 1. Other operators also can be used to either increment or decrement the value of the variable.

How to Control the Process

You have the liberty to decide what happens next in the program you have written using a control flow statement. The comparison

statement results can be used to create conditional statements that allow the interpreter to provide the output based on whether the predefined conditions hold true. Conditional statements can be constructed using the keywords if, elif, and else. Unlike other languages, Python does not use the keyword then. The syntax is very specific so you must pay close attention to the layout and punctuation.

```
if condition:
# Perform some actions
print "Condition is True"
elif condition != True:
# Perform some other actions
print "Condition is not True"
else:
# Perform default or fall-through actions
print "Anomaly: Condition is neither True nor False"
```

In the syntax above, the first line begins with the word if, followed by a conditional statement that gives a True or False output followed by the colon. This colon means yes. The statements that follow must always start on a new line. The number of spaces doesn't strictly matter so long as all the instructions after the colon are indented by the same amount, though it's good practice to use the same number of spaces to indicate control flow throughout your code. The statements following after the colon are known as a suite.

You can include further conditional sections using the elif keyword (an abbreviation of else-if, which is not a Python keyword); statements following elif will be evaluated only if the previous test fails (i.e., the conditional expression is False).

You can also include a final else: statement, which will catch any value that did not satisfy any of the conditions; it doesn't take any conditional expression at all. This can be used to specify a default set of actions to perform. In our previous example, things would have to go very wrong for us ever to see the final anomaly

warning, as the preceding if and elif statements would have caught either of the two possible results. If statements can be nested to allow for more possibilities, and you can leave out the elif or else statements if you don't want anything to happen unless the condition is satisfied. In other words, sometimes you want to do something if a condition is satisfied but do nothing if it is not satisfied.

After the final statement, the indentation must go back to its original level: this will indicate to the interpreter that the conditional block has come to an end. Python marks out blocks of code using indentation alone; it doesn't use punctuation marks like the curly braces you may see in other languages. This unique feature of Python means you have to be extra careful about indentation. If you do get it wrong, you'll find out soon enough, as the interpreter will complain loudly.

```
>>> if c:
... print(c)
... c += 1
... indent = "bad"
File "<stdin>", line 4
indent = "bad"
^
```

IndentationError: unindent does not match any outer indentation level

A conditional statement always gives the user the ability to check or validate the data used as the input. Validation is often performed when the data is first fed into the computer and written on a database record or file.

How to Deal with Logical Errors

As your applications become more complex, you will need more formal methods of testing your designs. One of the ways of doing this is to construct a trace table. You must trace the values of all

the variables and the conditional expressions throughout the program's execution.

A trace should be performed with as many different sets of data as necessary to ensure that all the possible alternatives get tested. Most errors in programming don’t occur if the values lie within some expected range, but they often occur for unusual values (also called critical values). Critical values are those that lie outside the program's tolerances, such a number that the application is not equipped to deal with.

Critical values should be worked out early on in the design process so that the program can be properly tested against them. In the calculation of the area of the triangle, the value that most needs taking into account is that of the breadth, which has been set at 14 cm. Allowing 8 cm means that the maximum breadth of the triangle can only be 8 cm.

Using the Conditional Code

Now you can apply your knowledge of conditional statements to allow for different ways of measuring up the material. If the breadth of the triangle were too much, it would become a different type of triangle. Therefore, you need to identify the right code that reflects the right conditions. The first step would be to translate your trace values into pseudocode. The following example is about measuring the length of a curtain.

```
if curtain width < roll width:
total_length = curtain width
else:
total_length = curtain length
if (curtain width > roll width) and (curtain length > roll width):
if extra material < (roll width / 2):
width +=1
if extra material > (roll width / 2):
width +=2
```

Loops

While Statement

```
result = 1
while result < 1000:
result *= 2
print result
```

To control the number of times the loop is processed, it is necessary to specify a conditional expression; as long as this conditional expression is True at the beginning of an iteration, the loop continues. In the preceding example, our conditional expression is result < 1000. So, as long as the value of the result is less than 1,000, the loop will continue processing. Once the result reaches 1,024 (210), the program will stop processing the loop body.

The variables used in the conditional expression are often expendable entities, which are only required for as long as the loop is active. Rather than keep thinking up different names, this kind of integer counter is usually named i or j by convention.

Two things are important to remember in this sort of construction: Any variable used in the conditional expression must be initialized before the execution of the loop. Also, there must be some way of updating this variable within the loop; otherwise, the loop will just go around and round forever, called an infinite loop.

It is possible to use different sorts of variables in the conditional expression. Let's consider the problem of calculating the average of several numbers input by the user. The main problem here is that I don't know how many numbers will be input. The solution is to use what is called a sentinel value to control the loop. Rather than using the counter in this instance, the script checks the value of the user input number. While it is positive (i.e., >= 0), the loop processes as it should, but as soon as a negative number is entered, the loop is broken, and the script goes on to calculate the average. Let us take a look at the following example:

```
counter = 0
total = 0
number = 0
while number >= 0:
number = int (input ("Enter a positive number\nor a negative to
exit: "))
total += number
counter += 1
average = total / counter
print(average)
```

You have different ways of defining and designing clean loops. The best way to do this is to use the continue and break keywords. If you want to leave a loop without executing statements in the body of the loop, use the break statement. Alternatively, if you want to move out of a specific iteration, you can use the continue keyword to move onto the next iteration in the loop.

There will be times when you need the compiler to interpret the condition and accurately. In such situations, use a pass keyword. This will create a null statement that will tell the compiler how it would need to interpret and look at the next instruction in the code.

Nesting Loops

You can nest conditional statements and loops in Python. You can have an infinite loop in the code, and this is something you need to track. It is important to ensure you limit the number of levels. For one thing, it's very easy to get confused about which option the program is taking at any point. If you have too many indented blocks in your code, it will make it hard to read the code. You need to design the code in a way that involves two or three layers of loops. Therefore, you should think about redesigning the code.

For

Another control statement you should understand is the 'for' statement, and this is constructed like the other conditional statements. The syntax is constructed in the following manner:

```
For (condition)
Suite of instructions
```

During the first iteration of the loop, the variable element contains the first element in the sequence and is available to the indented suite. In the second iteration, the second element of the sequence is considered, and this goes on until the conditional statement is false.

If you want to learn more about how this statement works, it is important to understand sequences. The simplest sequence in Python is a string, a sequence of individual characters, including spaces and punctuation. Some other sequence forms are lists and tuples. These are the sequences of data items, and the difference between these is that lists can be edited while tuples cannot be edited. It is also possible for you to use the loop, and you can construct it using the following statements:

```
# tuple

sequence1 = (1, 2, 3)

# list

sequence2 = [1, 2, 3]
```

Understanding the Jargon

Let us now look at some terms used in this chapter:

- Assignment operator: The assignment operator is the equal-to sign, and you can use it to equate a variable to a value or combine it with different operators. This allows you to perform numerous functions.

- Built-in: A built-in element is a part of the programming language, and it can be imported into any program from the library. These elements are a part of the standard library in Python
- Comparison operators: These operators compare two values:
- Conditional statement: Conditional statements are used to evaluate operations to determine the output as True or False. Conditional statements are control flow statements that determine the flow operations based on a condition.
- Critical values: A critical value is one that exists at the edge of any permissible range of values set for any variable. These values will change and can lead to unexpected results
- Loop body: This is a set of instructions that are to be repeated until a conditional statement proves false
- Null statement: A null statement is created because of the pass keyword, and this statement will tell the compiler to ignore the null statement and move to the next
- Validation: This refers to the process of checking that your data is what you expect.

You now know how to work with conditional statements and loops, so try to work on the exercises below to see how well you have understood the concepts. The solutions to these programs are in the last chapter.

1. Write a program to print the Fibonacci series
2. Write a program to print a palindrome
3. Write a program to check if a number is even or odd

Chapter Six

How to Use Files

So far, we have used data in a program in the following manner:

- Information written into the program
- Information received via the input() function

This information is printed in the output using the function print(). When the compiler runs the program, the information is lost. For an application to have any practical value, ensure it can store the information in the memory. This is the only way you can retrieve this information when the program needs to use it at a different time. A majority of computer information is stored in files on a hard drive or any other storage medium. Alternatively, you can also transmit the information using a file-type object. File-type objects share similar properties with files and can often be treated with the same methods. A stream is an important form of a file-type object.

How to Open Files

You can create a file object in Python using the built-in function open(). This function has the following parameters:

- Filename
- Mode
- Buffering

Built-in functions and methods also return file objects. In the example below, we will look at how you can open a plain text file within the same directory where the interpreter began:

```
>>> open('python.txt')
<io.TextIOWrapper object at 0xb7ba990c>
```

In the above example, we use a Python object called an io.TextIOWrapper. In simple words, this is a file object. If this file does not exist in the system or memory, the interpreter or compiler will throw an IOError. This error stands for an input and output error, and in other words, it means there is no file for it to read or write. The file object now holds the contents of story.txt, but you need to learn some file methods to do anything with it.

I am sure you are now used to the concept that most elements in Python are a form of an object, and there are different object types. You can use different methods to access these objects, and these methods can be used to edit the function or object. Before you use different file methods, it is important to understand how Python allows you to create any file object using an existing file in the memory.

Modes and Buffers

You can open a file by using the open function and passing the name of the file as the parameter. This function will only create a read-only object in the memory. If you want to write to that file as well, set the optional mode argument. The syntax of this argument will include the following:

- A single character: r (read), w (write), or a (append)
- This should be followed either by + (read/write) or b (binary)

If you don't provide a mode argument, Python will assume your file is a text file, and mode r will be assumed.

```
>>> open('python.txt', 'rb')
<io.BufferedReader object at 0xb7ba990c>
```

The object mode used in the example above will return an object different from the default file object. The default contains the

same information in byte format. Use this if you want to handle any audio data or image. The write mode (w) lets you change the contents of the file completely. Using the append mode (a) you can add information to the end of the file. This last option is useful for creating log files.

When you use the buffering argument, you can either use the w or a object mode. The values for these should either be 0 or 1. If it is 0, your data is written straight to your hard drive. If it is 1, Python creates a temporary file to store the text in before it gets written out. The compiler will not write this information to the disk unless you call the functions file.flush() or file.close() explicitly. You may not use this option in most cases.

Reading and Writing

The most basic method to access the file's contents is file.read([size]). This reads up to size bytes from the file (less if the read hits the end of the file before it gets to size bytes). The complete file is read as one string if there is no size argument provided or it is negative. Unless the file is opened as a binary object, the bytes are returned as string objects. In such cases, you will only have raw bytes as output. If you are reading a plain text file containing ordinary characters, you might not notice a great deal of difference.

```
>>> text = open('python.txt')
>>> text.read()
'Are you keen to learn the python language
[... pages more text here ...]
Thank you for purchasing the book.\n'
```

If you are dealing with a large body of text like the following example, you may wish to deal with it in smaller pieces. file.readline([size]) reads a single line from the file (up to the next newline character, \n). An incomplete line may be returned in an iteration. The size argument is defined as the number of bytes that the interpreter must read. The bytes also include the trailing

newline. If the interpreter reaches the end of the file, an empty string is returned.

```
>>> text = open('python.txt')
>>> text.readline()
```

"Are you keen to learn more about the python language? Thank you for purchasing the book. I hope you gather all the information you were looking for."

Files are their own iterators, so you can also iterate through lines in a file using a for loop. The same result as the file.readline() method is returned at each iteration, and the loop only ends when the method returns a null or empty string.

```
Try the following code out:
>>> for line in text:
... print (line)
```

Closing Files

When you are done with working on a file, use the file.close() function. Python will notice you are not using the file anymore, which can eventually free up the memory space, but it is important to close the file. Often, the data in a file.write() operation is stored in a buffer until the file is closed. You can make sure the data gets written out to the file without closing it by using file.flush(). The compiler cannot work on a closed file, and Python does not allow you to use the close() function more than once.

Chapter Seven

Working with Functions

If you want to create a new function, you first need to determine the input for the function. You also should determine the value the function should return, too. The structure and type of the data you want to feed or use in the function is of utmost importance. The information supplied or given to the function is termed as parameters and the information that comes as a result of the operations performed within the function. Our initial specification for the function design should also include a general description of the function's specific purpose.

Function Definitions

Functions in any program are defined using the def statement. The def keyword is followed by the function name and has an optional list of parameters written within parentheses. End the statement using a colon. This indicates to the compiler that the next statements are specific to the function. These lines should be intended as a suite or block of instructions. Consider the following example where we use a function that does not take any parameters:

```
>>> def generate_rpc():
... """Role-Playing Character generator
"""
... profile = {}
... print "New Character"
... return profile
```

In the above example, you see some instructions are to be compiled by the compiler. Each block of instructions works in the same way, and hence these are called functions. If you want to

provide some notes on what the function does, you can do so in the docstring, which must be the first thing in the function. This docstring will include a set of statements that provide information on the core functionality. This function will return some data that is based on the return statement.

The last line in the function specifies the variable you will send to the main program. If there is nothing to return, do not include the return statement in your function block. The compiler will return None as the value if there is no return function.

Until now, we have not run the function block of code. This block has only been assigned to the function. To get the code to run, you need to call the function from the main body of the program. This is termed the function call, and it is easy to call functions anywhere in your code using the name you have assigned to the function.

```
>>> generate_rpc()
New Character
{}
```

We still haven't specified the parameters in the example above, meaning the function does not have any values in the parentheses.

Defining Parameters

Functions use some information that can be fed to the main program based on the operations and statements within the function. For the function to receive the necessary information from the main program, you need to set up some empty variables that hold these values. These become variables unique to the function and are known as formal parameters, and it is best if you do not use any restricted words, keywords, or the names of variables in your main program. These formal parameters should be specified in the parentheses after the function name in the first line of the function definition.

```
import random
def roll(sides, dice):
result = 0 for rolls in range(0,dice):
result += random.randint(1,sides)
return result
```

You can call this function in the main program using the following statement:

```
muscle = roll(33,3)
```

The values or numbers in the parentheses in the function above are termed arguments. These are the values that are mapped to the parameters in the function name. In the above example, there are two arguments –

- The first argument is 33, which is bound to the function parameter side
- The second argument, 3, is bound to the function parameter dice

This means there are two variables in the function that are used. The following are some aspects to consider when to comes to working with functions:

- If you send values to the function like this, you need to send the same number of values in the arguments in the function.
- The function call should have the same number of values as the arguments in the function
- The arguments should also be in the same order as the function parameters, and this is because every parameter in the function is a positional parameter.

When you send parameters to a function, you need to specify the actual values in the function. Bear in mind the function will refer to that value by the name of the parameter that receives it. The

parameter's original value will not be affected, but the new value will be passed on.

Documenting Your Function

When you fully document the function and pass the code to the compiler, you need to test it. Edit the docstring, so the meaning of the functions will fit with the documentation of the Python tool.

Every docstring needs to follow the conventions listed below:

- The first line in any docstring should include a short description of the function that makes sense by itself. Ensure you restrict the number of characters you use to 79
- The second line should be left blank
- The next sections should include the body of the docstring and should contain a description of the function's parameters. You can also include a longer description of the function
- The docstring also has notes about the algorithm used. You can include some information about the keyword arguments, optional arguments, and the return values.
- You may also want to include information about any throw-catch or error handling statements. In short, all the information that another programmer would need to be able to use the function.

Bear in mind that the docstring should be updated regularly, and comments must be updated when you change the code at any point.

Working with Scope

It is important to understand and conceptualize the scope of any function. Bear in mind that the function is a black box that can take in the source information, process it, and pass the value onto

the main program. The code in the main program that sends the source data and receives the result is known as the calling procedure. This does not mean the main program needs to know the lines of code that are in the black box. The source data and results should be defined clearly.

Understanding Scope

It is important to understand the meaning of the scope of a function. Never use the same name in the main program to name any function you write. This is the basic understanding of scope.

When any program is run, the interpreter keeps track of all the names that are created and used in the program. These names are tracked in a table referred to as a symbol table and can be viewed as a dictionary using the built-in vars() function. The variables created in the main program's symbol table are known as global variables because any part of the program can access these variables. These can be viewed with the built-in globals() function. You will notice that the result of running vars() with no arguments inside the main program is the same as running globals(). So far, the script examples in this book have only used global data.

Any variable that is created in a function is stored in a symbol table unique to that function; this data is known as local data and can be accessed within that function using the locals() function. The main body of the program (global scope) cannot access the variables set inside the function (local scope). The function, however, is still able to access names in the main body, that is, globals. Hence, you have two ways to get at a function to process some data: The correct way is to take that data as a parameter for the function; the function will process it and then return the result, which the main program will collect. The other way is to let the function directly access and process the global data, but this second way is discouraged because the function works with only particular variable names and cannot be reused anywhere else.

Manipulating Dictionaries and Lists

Parameters to a function are passed by value. However, in the case of objects (unlike simple types such as integers and strings), if you modify the parts of an object, including items in a list or a dictionary, the changes are reflected outside the function as well. This means that a function can be used to modify a mutable data type such as a dictionary or list. Technically, such a modification procedure is not a pure function, as it neither needs a return value or, indeed, any input parameters.

A good general policy is sending and returning immutable values from functions and keeping any modification procedures for your mutable types separate to avoid strange side effects. If you send a list or dictionary to a function, you are only sending a pointer to the same global instance of that object, just as if you had written local_list = global_list.

Abstraction

This round of testing may bring new issues to light. To help you keep track of what's going on with your code, do the minimum necessary to get the new code to pass its tests. If the new code fails, you should roll back the changes to the point where the code passes again. Also, if using the function doesn't work, don't sweat it: reverse the changes and move on. No rule says you have to abstract everything into a function.

The processes of placing code into functions (abstraction) and turning code that deals with specifics into general-purpose code (generalization) can also be applied to your testing procedures. This is a good thing to do. All these test procedures will eventually get moved outside the program into a separate module, so abstraction and generalization are also good preparation. The rule to remember is to write your tests first and then make sure your existing code still passes before attempting to refactor it. Although this might seem like a laborious approach it speeds up the development of new code and provides a mechanism for

flagging up which part of the code needs fixing next, rather than having to keep all the details in your head. This has to be a good thing.

Chapter Eight

Web Scraping Using Python

There is a lot of data and information available on the Internet, and most people, especially businesses, want to use this information to their benefit. If you want to use the information in the right way, you need to learn how to scrape data from the Internet. Python has many libraries and tools that can be used to perform this job. This chapter has all the information you need about Python and HTML. We will look at the following:

- Inspecting the HTML structure of the target website using the developer tools in your browser
- Break the code down in the URLs
- Use Beautiful Soup and requests in Python to scrape and parse data on the Internet
- Use the web scraping pipeline from start to finish
- Use a script to perform the job offers from the Internet and display the needed information on your system

When you go through this chapter, you will gather information about the different tools and processes you need to scrape on any static website on the Internet. So, let us begin.

Understanding Web Scraping

In simple words, web scraping is the process of looking at new information on the Internet. Even when you copy or paste information from the Internet, your favorite song or movie plot, you scrape information from the Internet. The term "web scraping" is the process that uses automation and other objects

and packages in the library. Some websites have restrictions on the data being extracted, while others do not.

If you scrape the information from a page for the purpose of education, you will not have any problems. Therefore, it is best to perform some research to ensure you do not violate any terms or conditions set by the website developer. You need to do this before you begin working on web scraping.

Reasons to Scrape

Let us assume you surf the information on a website or are often looking for a new job. However, you are not looking for information about just any job. If you have this mindset, the only thing you need to wait for is the perfect opportunity.

One website has all the job postings you need, and unfortunately, new positions are often not found on this website. The website does not provide the relevant notification services, so you check the website every day to ensure you do not miss any opportunity. It is best to avoid doing this if you want to save time.

Thankfully, you can find easier and better ways to surf the information on the Internet. You do not have to look for the information every day but can automate the task using Python. Python offers you to search the information just by running a script, and this helps to improve the pace at which information is collected. All you have to do is write the code or script and run it every time you need some information about a new job posting.

Alternatively, when you look for this information manually, you spend a lot of time searching, clicking, and scrolling the information. You will waste a lot of time extracting large volumes of information from the website. Every website is updated with new information every second, so the task of collecting information becomes tedious. It will take a long time for you to scrape the information, especially if the act is repetitive.

There is a lot of information on the Internet, and you may be interested in the latest information on the website. Most of this information is on different websites on the Internet. You can accomplish different goals using the automated web scraping method.

Web Scraping Challenges

The Internet has grown drastically over the years, and numerous sources can be used to source relevant information. The Internet uses different technologies, personalities, and styles. It continues to grow every second of every day. In simple words, the Internet is a mess, and it is for this reason you are going to have some issues with web scraping:

Variety

Since websites are very different, you may encounter repetitive structures. Every website is different and needs to be treated differently, especially if you want to obtain the relevant information.

Durability

Every website changes frequently. Let us assume you have a web scraper developed that picks the relevant information you need from the website. When you run this script the first time, you will see that it works flawlessly. If you run this script repeatedly, you will see that you can perform tracebacks easily.

If you cannot build a stable script, do not use it. Most websites are continuously developing, and when the website structure changes, you need to find a way to navigate through the sitemap the right way. This is how you obtain the required information. The advantage is that every website is built differently, and the changes are incremental and small. You only need to update the scraper with the information needed by making small adjustments.

Bear in mind the Internet is dynamic, and the website changes every day. Any scraper you build wil require maintenance. Use continuous integration, set up and run the tests to ensure the script you use does not stop working when you run the script.

Using APIs for Web Scraping

Some website developers or providers use APIs or Application Programming Interfaces to access information on the website in the right manner. Using APIs, it becomes easy to parse the data in the HTML elements. Alternatively, you can access information using the XML and JSON formats. The developers use the HTML elements to present this information the right way.

If you use an API, the process of scraping information from any website is stable. It can collect the relevant information from the Internet, and most developers use APIs that consumers can use. The website can look different frequently, and a change in the design will not affect the API design. The design of any API is often permanent, and this means the API is more relevant in terms of the information.

APIs also change when you update the website, and the challenges of using APIs are almost the same as that of web scrapers. It is very difficult to understand the design of an API, especially if the documentation does not have the relevant information you need.

Scraping a Job Website

In this chapter, we will build a web scraper using which you can obtain or extract the information of all developer job postings on the Internet. We are considering a fake website with the information we want to scrape. The web scraper you develop is now going to be used to scrape the HTML elements from the website, using which you can pick the relevant details. You can also use the scraper to filter the needed information.

You can use the web scraper to obtain information from the Internet, but the issue of using this scraper is based on the website being used. This chapter introduces web scraping and helps you understand the entire process of web scraping. You can then apply the process of using this scraper.

Step 1: Inspecting the Data Source

Before you begin working with any code in Python, understand the website you want to use and scrape information from. This is the first step for any project you want to work on. Understand the website and the structure before you extract the relevant information from the Internet. So, before you begin working on web scraping, open the website and understand it better.

Explore the Website

Go to the website you want to scrape information from and understand it. Understand the information on the website and see how to use it the way you would to look for a relevant job posting. For instance, you can scroll through the website. On this website, you will see numerous postings, and every posting has two buttons – one is to learn more about the job, and the other is to apply for it. If you choose the second option, you will move onto a new page that has more information or details about the job. Include the URLs you identify in the browser address to interact with all the information on the website. Assess how you would want to use the information on the website.

Decipher the Information in URLs

The next stage is to understand the information on the website. As a programmer, you can encode a vast amount of information in any URL. The process of web scraping will be easier if you familiarize yourself with how the URL works. You also understand the information in the URL. For instance, you may find yourself looking for information on the website. Consider the website stored at the following link: https://realpython.github.io/fake-jobs/jobs/senior-python-developer-0.html

Any URL can be broken down into the following parts:

- Base URL: This URL represents the information on the website, and it is the path that provides details of the functionality of the website. In the above example, the URL is https://realpython.github.io/fake-jobs/
- Specific Site Location: This location will have the HTML information that has the details of the unique resource of the job description

This indicates that any job posting on this website will use the same URL, but it will come with unique resources. This is based on the job posting you want to look at. A URL holds more information about the file location. Most websites use parameters in queries to encode the values which are submitted to the website when you perform any search on a website. You can also think about the queries and parameters you can use to obtain specific information. You will see the parameters at the end of every base URL. Let us assume you go to LinkedIn and look for a software developer job in New Zealand in the search bar. The URL automatically changes since it will include the parameters, as well.

https://au.linkedin.com/jobs?q=software+developer&l=Australia

In the above URL, the parameters have changed to are ? q=software+developer&l=Australia. There are three parts to the parameters:

1. Start: This is the start of the query parameters, and the start is represented using a question mark symbol
2. Information: The URL query parameters have the relevant information that constitutes the parameters in the pairs of information. These pairs are called key-value pairs. These pairs are related to the queries run

3. Separator: Most URLs come with various query parameters, and each of these parameters is separated using the & or ampersand symbol

Now that you have this information, you can break this URL into different value pairs based on the query parameters.

- l=Australia selects the location of the job.
- q=software+developer selects the type of job.

If you want to understand how this works, you can change or update the parameters. This will help you observe the change in the URL based on the queries. Enter the values in the URL and search the relevant details in the URL. You can also work on changing the values directly in the URL. Consider the URL below, and see how it would change the address.

https://au.linkedin.com/jobs?q=developer&l=perth

If you were to change these values and submit them in the search box, it would reflect directly in the parameters in the URL. The reverse is also possible. If you want to change any of these, you will notice a different result on this website. From the above URL, you can see that the information has changed based on the information or details you want to receive from the data set.

Now go back to the main website we are going to use and explore it. The website is static, and this will not operate at the top of this database. You do not have to worry about working with query parameters since the query does not change.

Using Developer Tools to Inspect the Website

You should now understand how the information is structured. Only when you know the page structure can you determine the response you may receive when you run the code. The code we write in this chapter can be used to collect the relevant information in the steps in the next section.

You can use a developer tool to understand how the website is structured. Every modern website will come with its own developer tool. In this section, we will look at different developer tools. The process used is like that used in modern browsers. If you were to use Chrome on a Mac OS, you could access the developer tool by following steps: View → Developer → Developer Tools. If you use a Linux or Windows system, it becomes easier to access the tools by clicking on the settings and go to the developer tools using the following path More Tools → Developer Tools. The developer tools can also be accessed using the inspect selection. You can do this using the following shortcuts:

- Windows/Linux: Ctrl+Shift+I
- Mac: Cmd+Alt+I

A developer tool allows you to explore the website DOM or document object model to understand the source of information. To learn about the DOM, use the elements in the developer tool. You can identify the elements using the developer tools. The website has a structure of how the elements are structured. You can collapse, edit and even expand any elements present on the page.

Step 2: Scraping Information from the Website

You now have an idea about the type of information you are working with. You also know the details of the website you are working with. The first thing to do is to understand the HTML code and move that information onto the Python script you write. It is important to understand how you can work with the HTML elements. We are going to use the requests library in Python for this task.

The first thing to do is to create or design a virtual environment. You need to do this before you download or create packages, internal or external. You also need to activate the virtual

environment and type the command below in the terminal. This will help you install the relevant library in Python.

```
$ python -m pip install requests
```

Now, open the file in an editor and retrieve the elements using the following lines:

import requests

```
URL = "https://realpython.github.io/fake-jobs/"
page = requests.get(URL)
print(page.text)
```

Using this code, you can use the HTTP GET request to obtain the relevant URL. You can obtain HTML element information using this code. You can also store the information in an object. If you use the .text attribute and print it, you will notice it looks like the relevant information is extracted from the browser. It is easier to do this using the developer tools in the browser. You can obtain relevant information from the Internet. Now, you can access the information on the website using your script.

Static Websites

In this section, we are looking at code used to scrape data from a static website. It is a static website, which would mean the HTML elements do not change. In this case, the website is on a server which sends the information from the website to the documents which already have the information you need. Once you inspect the page using these developer tools, you can discover the job posting which is based on the following lines of HTML code:

```
<div class="card">
  <div class="card-content">
    <div class="media">
      <div class="media-left">
        <figure class="image is-48x48">
          <img
            src="https://files.realpython.com/media/real-python-logo-
thumbnail.7f0db70c2ed2.jpg"
```

```
          alt="Real Python Logo"
        />
      </figure>
    </div>
    <div class="media-content">
      <h2 class="title is-5">Senior Python Developer</h2>
      <h3 class="subtitle is-6 company">Payne, Roberts and
Davis</h3>
    </div>
  </div>

  <div class="content">
    <p class="location">Stewartbury, AA</p>
    <p class="is-small has-text-grey">
      <time datetime="2021-04-08">2021-04-08</time>
    </p>
  </div>
  <footer class="card-footer">
    <a
      href="https://www.realpython.com"
      target="_blank"
      class="card-footer-item"
      >Learn</a
    >
    <a
      href="https://realpython.github.io/fake-jobs/jobs/senior-
python-developer-0.html"
      target="_blank"
      class="card-footer-item"
      >Apply</a
    >
  </footer>
 </div>
</div>
```

It is difficult to understand this code, especially if you do not understand HTML. If you want to make this easier for you to understand, use a formatting tool in HTML that allows you to clean the code. If you read the code better, it becomes easier for

you to break down and assess the block's structure. This may not improve the way the code is formatted, so it is best to try.

HTML is also confusing since the code is difficult to process. If you ever lose your understanding because of HTML, you can go back to exploring the website's structure using the developer tools.

Dynamic Websites

This chapter will teach you how to scrape information from a static website, rather than dynamic. These websites are easy to work with since the server sends the output of the HTML page that has the relevant information you need. It also becomes easy to parse the information present in a static website.

Step Three: Parsing Code Using Beautiful Soup

Now, you have successfully obtained some information from the website. When you look at this information, you will see it is a mess. There are numerous elements on the website, and these are the HTML elements. You will also have numerous attributes scattered all over the website. Some of the elements will also have JavaScript mixed within the output. You now need to find ways to parse the information you have obtained using Python, especially if you want to make this accessible. This is the only way you need to pick the relevant information.

Another Python library is called Beautiful Soup, and this library is used to parse any structured information. You can interact with the elements on the website using different developer tools. This library also has relevant, intuitive functions using which you can explore the different elements in HTML. To begin, you first need to install this library using the following line of code:

```
$ python -m pip install beautifulsoup4
```

You should then import this library into the relevant script, allowing you to create a Beautiful Soup object:

```
import requests
```

```
from bs4 import BeautifulSoup

URL = "https://realpython.github.io/fake-jobs/"
page = requests.get(URL)

soup = BeautifulSoup(page.content, "html.parser")
```

If you add the lines of code, you can create any new object in the Beautiful Soup library, where you can store the content of the page. This is the information you need to scrape from the website. Avoid using the page.text object to ensure you do not have any trouble with the output. “html.parser,” which is the second argument you should use to obtain the appropriate information.

Finding Elements Using IDs

On any HTML website, the elements will be assigned different attributes. You can assign one of these variables as the ID attribute, which allows you to uniquely identify information on the website. When you do this, you parse the information on the page basis the relevant information in the ID. Now, use the developer tools to identify the objects in the HTML elements that contain the relevant job postings. You need to explore the elements by hovering over the parts on the page using the inspect option. All you need to do is right-click on the page and inspect the elements.

It also helps if you switch back between the inspect elements and the browser to explore how the page works. This is the best way to find the elements you want to use to perform the scraping. To find the relevant information, look for the object followed by the ID name. Look for the combination with the result or assignment value as “ResultsContainer.” There are other attributes in these elements. For a better understanding, consider the following lines of code:

```
<div id="ResultsContainer">
  <!-- all the job listings -->
```

```
</div>
```

Using Beautiful Soup, you can specify every HTML element basis its ID:

```
results = soup.find(id="ResultsContainer")
```

If you want to view the elements individually, you can use any Beautiful Soup object to print this information. The easiest way to do this is to run the prettify() function. You can use the line of code call.prettify() on the variable you assigned above to see the information present in .

```
print(results.prettify())
```

If you use the IDs in the elements found on the website, choose the element you want to consider. You can now work with only the relevant part of the page. Bear in mind that the soup is thinner, but the result is still dense.

Using HTML Class Names to Find Elements

Every job posting on the website will have a div element within which it is wrapped. This element also comes with the class card-content. You can also work with new objects called results, using which you can select the relevant job postings on the website. These are the parts of the HTML statements for which you need information. It is easy to do this using the following line of code:

```
job_elements = results.find_all("div", class_="card-content")
```

When you call the .find_all() function using the objects present in the Beautiful Soup library, you can obtain the iterable objects that contain the details of all the job listings based on the keywords found. Now, consider the following code:

```
for job_element in job_elements:
  print(job_element, end="\n"*2)
```

This is a very neat line of code, especially when there is a lot of code in the HTML elements on the website. In the earlier sections,

you noted there are descriptive class names on the website you are scraping the information from. Using the with.find() function, you can find the child elements on the website. Consider the following lines of code:

```
for job_element in job_elements:
    title_element = job_element.find("h2", class_="title")
    company_element = job_element.find("h3", class_="company")
    location_element = job_element.find("p", class_="location")
    print(title_element)
    print(company_element)
    print(location_element)
    print()
```

Every job_element found on the website will be stored as an object by the Beautiful Soup library. You can use any method on the object list as you did with the parent element basis in the results library. The following lines of code help you get closer to the information you actually need. There is a lot of information on the website which you should filter, and you may only be interested in some bits of information. There is, however, a lot of information that is stored in the HTML attributes and tags.

```
<h2 class="title is-5">Senior Python Developer</h2>
<h3 class="subtitle is-6 company">Payne, Roberts and Davis</h3>
<p class="location">Stewartbury, AA</p>
```

Let us now look at how you can narrow the output so you only access the relevant information from the website.

Extracting Information from HTML Elements

From all the job postings on the Internet, you only want to see the title, location and company. You can do this easily using Beautiful Soup, and you can add the .text function or method to any object from the library. This will only return the relevant information to the object or element you contained. The following code can be used:

```
for job_element in job_elements:
    title_element = job_element.find("h2", class_="title")
```

```
    company_element = job_element.find("h3", class_="company")
    location_element = job_element.find("p", class_="location")
    print(title_element.text)
    print(company_element.text)
    print(location_element.text)
    print()
```

When you run the above code in Python, you will notice that every element displayed on the website will be saved as a text element, but, it is also important for you to remove any whitespaces from the text you use as filters. I am sure you are aware of how to work with strings in Python, use the .strip() function. It is also important to ensure you familiarize yourself with the string methods needed to clean the information in the text:

```
for job_element in job_elements:
    title_element = job_element.find("h2", class_="title")
    company_element = job_element.find("h3", class_="company")
    location_element = job_element.find("p", class_="location")
    print(title_element.text.strip())
    print(company_element.text.strip())
    print(location_element.text.strip())
    print()
```

The output will look better:

```
Senior Python Developer
Payne, Roberts and Davis
Stewartbury, AA

Energy engineer
Vasquez-Davidson
Christopherville, AA

Legal executive
Jackson, Chambers and Levy
Port Ericaburgh, AA
```

This is easier for you to read, and the information also contains the necessary information you need to determine if you want to

apply to the role. That said, you can also look for software developer roles, and the results will contain the information you need about developers in all fields.

Finding Elements Using Text and Name Content

Not every job listed on the website has the title developer in it. So, instead of looking at all jobs on the website, you can use keywords to filter this information. On most of these websites, the second header element will contain the job title information. If you want to look only for specific jobs, use the following string argument. Alternatively, you can also use functions, and we will look at this in detail in the next section:

```
python_jobs = results.find_all("h2", string="Python")
```

Using this line of code, you can save all the header two elements that contain the keyword “Python” and save the information in an accessible variable. You are calling this method directly on the first variable you extract. When you print the output of this function, you will not have anything in the output:

```
>>>
>>> print(python_jobs)
[]
```

From what we know, there is a Python job in the above search result, so why did that not show up in the output? If you use the string = as assignment operator above, you will see that the compiler looks for that exact information. If there is any difference in capitalization, whitespace, or spelling, it will make it hard for the compiler to match the information. In the next set, we will look at how you can use strings for general searches.

Passing Functions Using Beautiful Soup Methods

When you write code, you do not only have to use strings. You can also use functions as arguments when you use the Beautiful Soup methods. Additionally, you can update the previous line of code if you want to use a different function:

```
python_jobs = results.find_all(
    "h2", string=lambda text: "python" in text.lower()
)
```

In the above code, you see that an anonymous function is passed as the string argument in the code above. This lambda function now looks at the header two elements and converts all that information into lowercase. It also looks at the python substring to see if the information can be found anywhere. This would allow you to identify the websites where there are postings for Python jobs. Use the following approach:

```
>>>
>>> print(len(python_jobs))
10
```

If you run the above code, you will note that the output includes all job titles with the word “python.” To find elements on the website, you need to identify the elements based on the text content. You need to filter the response so you have the necessary information. Using Beautiful Soup, you can use exact functions or strings as arguments, and these can be used to filter the necessary information in every Beautiful Soup object. However, when you try to run the scarper to print the relevant information of the filtered jobs with the “python” keyword, you will receive the following error:

```
AttributeError: 'NoneType' object has no attribute 'text'
```

This is a common error most users obtain when they perform web scraping, especially if this is being done on the Internet. You also need to look at the HTML element in the list extracted to understand what the information looks like and where you believe this error will come from.

Identifying Error Statements and Conditions

If you look at the element python_jobs and the single element stored in this attribute, you will notice it has the information of all

the second header elements on the website. Consider the following line of code:

```
<h2 class="title is-5">Senior Python Developer</h2>
```

If you look at the above code, you will see that the items selected were exactly what you have requested as the target variable. You only looked for the second header element in the job posting website. This code contains the information “python,” and you will notice that these elements do not include any other information about the job posting. You would have received an error message:

```
AttributeError: 'NoneType' object has no attribute 'text'
```

You may have tried to look for the relevant information about the job, including:

1. Job title
2. Company name
3. Location

This information is stored in every element in the python_jobs module. This element, however, will only contain the text of the job title. You also use a diligent parsing library, and this library will return the output as None since it cannot find the other elements you are looking for. You should then use the print() function to show the error message on the screen. You can do this when you use the .text attribute. The information you are looking for on the website is a part of the header two elements, but it is a sibling of that element. You can use the Beautiful Soup library to help you choose the child, parent, and sibling elements.

Accessing Parent HTML Elements

The easiest way to access the information you need on the website is to set up the DOM hierarchy. This will begin with the

header two elements which the script identified. You can also look at the HTML of the job posting if needed, and see which header contains the information about the job. You can choose that element along with the closest parent element which contains the information you need:

```
<div class="card">
  <div class="card-content">
    <div class="media">
      <div class="media-left">
        <figure class="image is-48x48">
          <img
            src="https://files.realpython.com/media/real-python-logo-
thumbnail.7f0db70c2ed2.jpg"
            alt="Real Python Logo"
          />
        </figure>
      </div>
      <div class="media-content">
        <h2 class="title is-5">Senior Python Developer</h2>
        <h3 class="subtitle is-6 company">Payne, Roberts and
Davis</h3>
      </div>
    </div>

    <div class="content">
      <p class="location">Stewartbury, AA</p>
      <p class="is-small has-text-grey">
        <time datetime="2021-04-08">2021-04-08</time>
      </p>
    </div>
    <footer class="card-footer">
      <a
        href="https://www.realpython.com"
        target="_blank"
        class="card-footer-item"
        >Learn</a
      >
      <a
```

```
      href="https://realpython.github.io/fake-jobs/jobs/senior-
python-developer-0.html"
      target="_blank"
      class="card-footer-item"
      >Apply</a
     >
    </footer>
   </div>
  </div>
```

The element is a part of the card-content class, which contains the information you want to extract from the website. This element is the third-level parent in the second header section you found based on the filter being used in the code. Now that you have this information, you can use the python_jobs elements and fetch or obtain the information of the parent or grandparent elements to obtain the information you need.

```
python_jobs = results.find_all(
   "h2", string=lambda text: "python" in text.lower()
)

python_job_elements = [
   h2_element.parent.parent.parent for h2_element in python_jobs
]
```

This information is included in the list comprehension, which operates the second header title element in the module python_jobs. This information was obtained when you filtered the details using lambda. When you do this, you will select the parent element of every other parent element in the second header element in the HTML page. When you look for the information in the HTML elements on the job posting website, you need to identify the relevant parent element using the card-content class name. This would help you obtain all the information you need. You can use the loop statement ‘for’ in the code, using which you can perform an iteration over all the parent elements.

```
for job_element in python_job_elements:
  # -- snip –
```

If you run this script one more time, you will see the code has changed and updated. This new code can now access the information on the website. It is because you can loop the header elements using the module, and if you use the . parent attribute, you can obtain some information about the structure of the elements.

Extracting Attributes using HTML Elements

The script you have written can scrape the information from the website and filter the HTML elements to obtain the relevant details. What do you think is missing now? The link to apply for the role. While you were working on identifying the elements in the page, you identified there were two links at the bottom using which you can apply to the position. You cannot work with these elements the same way we worked with other elements in the above sections. Consider the script below:

```
for job_element in python_job_elements:
  # -- snip --
  links = job_element.find_all("a")
  for link in links:
    print(link.text.strip())
```

When you run the above code, you will receive the links Learn and Apply and not the actual URLs. This happens since the .text attribute only leaves that portion of the HTML element, which is easy for the compiler to see. The compiler does not look at the HTML tags. It also considers the HTML attributes that have the URLs, which will only leave you with the link text. If you want to obtain the URL, extract the attribute values and keep them without removing them from memory.

The href attribute in the code is associated with the URL link element in the HTML page, and the URL you want to use as the output is the second href attribute. This attribute takes the value

of the second tag in the HTML page. This would give you the link to apply to the job posting.

```
    <!-- snip -->
    <footer class="card-footer">
      <a href="https://www.realpython.com" target="_blank"
        class="card-footer-item">Learn</a>
      <a href="https://realpython.github.io/fake-jobs/jobs/senior-
python-developer-0.html"
        target="_blank"
        class="card-footer-item">Apply</a>
    </footer>
  </div>
</div>
```

Using the above code, you can fetch all the elements in the HTML element list. You can then extract the value of each of the attributes in the code. Consider the example below. You need to use the same notations below if you want to apply the concepts in other areas:

```
for job_element in python_job_elements:
    # -- snip --
    links = job_element.find_all("a")
    for link in links:
        link_url = link["href"]
        print(f"Apply here: {link_url}\n")
```

In the above code, we have first fetched the links on the HTML website using the elements by filtering on the selected job postings. We are then extracting the href attribute, which has the URL. These URLs are stored in the href attribute and then printed on your screen. Bear in mind that you use the same notations to address or access all the attributes in the HTML.

If you have been working on the code while reading the chapter, then run your script the way it is. You will notice that a pop-up appears on your screen in the selected terminal with information about the job. The next step would be to use the information in this chapter on actual websites. You can revisit the process if you

want to master the art of web scraping. You can also use the following websites to learn more about web scraping:

- Indeed
- Remote(dot)co
- PythonJobs

The websites will return the search results based on the instructions you have assigned in the code. The responses will be like the code we have written in this chapter. You can scrape the information off the website using Beautiful Soup and the requests library. If you are unsure of how to use web scraping on other websites, follow these instructions.

Understand that the structure of any website will be different, and you will need to rewrite the code differently to fetch the right information. It is important to track these changes in the right way. You can only do this when you practice the steps mentioned in this chapter. It will take you longer to write the code if you do not understand the concepts as well as you should.

When you attempt to work on the code, you should explore different features present in both libraries. Use the information in this chapter as your guide. Only when you practice as often as possible will you become proficient at scraping using Python. You can then create an application that you can run on the command-line interface. This application can be used to scrape the information on the Internet and filter for the relevant results.

In Python, the requests library gives you an easy way to obtain information from the URLs. You can use the requests library to obtain relevant information from HTML and parse the URLs using another package termed Beautiful Soup. This package will cater to the needs you have with respect to parsing information, including advanced searching and navigation. In this chapter, you looked at how you can scrape information from the Internet using

the requests library in Python and Beautiful Soup. We have also built a script that fetches the relevant job postings for any user from the Internet and performs scraping from the Internet from beginning to end.

Chapter Nine

Tasks to Automate Using Python

Everybody wants to automate processes to help to avoid performing redundant tasks. You can use Python to automate to-do lists, finances, and other aspects of your life. Having said that, a lot of people still avoid automating different processes for numerous reasons. Automating tasks with Python helps improve the functioning of a process and reduces the time spent on different tasks. In this chapter, we will look at some tasks you can automate. Regardless of whether you write software, take notes or write business logic, you need to automate.

Automating Tasks Using Python

This section only contains a handful of tasks that can be automated using Python. The section does not provide too much information for each task, but it will give you an idea of where to start. If you are new to automating tasks with Python, these are simple ideas, to begin with.

Converting Files to Audiobooks

The code under this section works for a Mac device, but you can convert it to be compatible with your operating system. This program can run in the background easily. Install the following on your system:

```
pip install mac-say
```

The next step is to create the file you want to use to perform the task.

```
import sysimport mac_saymac_say.say(["-f", sys.argv[1], "-v",
"Jamie"])
```

Now, using the next command, you can convert any text into an audiobook:

```
python audiobook.py fileofyourchoice.txt
```

Generating Weather Reports

Most people check their weather, and most devices come with the weather feature in the system. If you want to generate the weather report through a click, use the example below. Bear in mind that this script would only have one level of dependency.

```
pip install requests
```

Once you install the requests library in your system, you can create the following file to generate the weather report:

```
import sysimport requestsresp =
requests.get(f'https://wttr.in/{sys.argv[1].replace(" ", "+")}')
print(resp.text)
```

Once you do this, you can run this script every day. Alternatively, you can schedule the script to run when it needs to.

```
python weather.py "Your City"
```

Currency Conversion

This is an easy aspect to consider, and all you need to do is install the library using the following code:

```
pip install --user currencyconverter
```

Once you install this library on the system, the currency_converter to the desktop. You need to add this to the $PATH in the system files to convert currencies. To do this, write the following code:

```
currency_converter 1 USD --to EUR
```

Sorting the Downloads Folder

In the example below, we are going to look at sorting different types of files, including audio, video, images and PDFs. You can expand the files if you need to sort through different types of files.

```
import os
import time
import sysfrom watchdog.observers import Observer
from watchdog.events import
FileSystemEventHandlerfolder_to_monitor =
sys.argv[1]file_folder_mapping = {
    '.png':'images',
    '.jpg':'images',
    '.jpeg':'images',
    '.gif':'images',
    '.pdf':'pdfs',
    '.mp4':'videos',
    '.mp3':'audio',
    '.zip':'bundles'
}class DownloadedFileHandler(FileSystemEventHandler):
    def on_created(self, event):
        if any(event.src_path.endswith(x) for x in file_folder_mapping):
            parent =
os.path.join(os.path.dirname(os.path.abspath(event.src_path)),
file_folder_mapping.get(f".{event.src_path.split('.')[-1]}"))
            if not os.path.exists(parent):
                os.makedirs(parent)
            os.rename(event.src_path, os.path.join(parent,
os.path.basename(event.src_path)))event_handler =
DownloadedFileHandler()
observer = Observer()
observer.schedule(event_handler, folder_to_monitor,
recursive=True)
print("Monitoring started")
observer.start()
try:
    while True:
        time.sleep(10)except KeyboardInterrupt:
    observer.stop()
    observer.join()
```

Once you save this script on your desktop, you only need to point the compiler to your downloads folder to start sorting the files in the folder.

```
python downloads-watchdog.py "/your/downloads/folder"
```

Easy Access to URLs

Most people access certain URLs and read the news or any article that pops up on those screens. Using the script below, you can open browsers on your laptop easily without having to key in the URLs in your browser. You can save a script with the list of your favorite URLs:

```
python -m webbrowser -t "https://www.google.com"
python -m webbrowser -t "https://www.facebook.com"
python -m webbrowser -t “https://www.python.org ”
```

Python is a very powerful tool, and it is important for you to practice, so you learn exactly what you should do. Only when you practice will you become more productive and efficient at using the tool to benefit you. This chapter has some fun and silly automation tasks, and I hope you found them interesting and easy to understand.

Chapter Ten

Cleaning Data Using Python

This chapter will look at how to use the NumPy and Pandas libraries in Python to clean various data sets when you scrape information from the Internet and use the codes in this chapter to clean that data.

In this chapter, we will be using the following data sets:

- olympics.csv – This is a CSV file that provides a summary of the countries that participated in the Summer and Winter Olympics
- university_towns.txt – This is a text file that contains information about the college towns in the US
- BL-Flickr-Images-Book.csv – This is a CSV file that has information about the books in the British Library

You can download these data sets from the GitHub repository for Python. Use the following lines of code to import the libraries before you clean the data.

```
>>>
import pandas as pd
import numpy as np
```

Dropping Columns in a Data Frame

Most columns in the data set can be used to perform an analysis. For instance, if you look at the output from the web scraping of the job positions, you will see the data set has a lot of information about jobs, including the word “Python” and “software developer.” You, however, only want some information from this

data set. You can remove these categories to reduce the amount of space taken up by the data set, thereby improving the program's performance.

If you use the Pandas library, it becomes easy to remove any unwanted column or row in any DataFrame. You can do this using the drop() function. We will now look at an example where we remove columns from a DataFrame. Before this, we need to create the DataFrame, and we will create one using the BL-Flickr-Images-Book.csv file.

In the example below, we will pass the path to the compiler by pushing it via the pd.read.csv. This indicates all the data sets from the CSV file are saved in the folder named Data sets in the current working directory.

```
>>>
df = pd.read_csv('Data sets/BL-Flickr-Images-Book.csv')
df.head()
Identifier Edition Statement Publication Place \
0 206 NaN London
1 216 NaN London; Virtue & Yorston
2 218 NaN London
3 472 NaN London
4 480 A new edition, revised, etc. London
Publication Date Publisher \
0 1879 [1878] S. Tinsley & Co.
1 1868 Virtue & Co.
2 1869 Bradbury, Evans & Co.
3 1851 James Darling
4 1857 Wertheim & Macintosh
Title Author \
0 Walter Forbes. [A novel.] By A. A. A.
1 All for Greed. [A novel. The dedication signed... A., A. A.
2 Love the Avenger. By the author of “All for Gr... A., A. A.
3 Welsh Sketches, chiefly ecclesiastical, to the... A., E. S.
4 [The World in which I live, and my place in it... A., E. S.
Contributors Corporate Author \
0 FORBES, Walter. NaN
1 BLAZE DE BURY, Marie Pauline Rose - Baroness NaN
```

2 BLAZE DE BURY, Marie Pauline Rose - Baroness NaN
3 Appleyard, Ernest Silvanus. NaN
4 BROOME, John Henry. NaN
Corporate Contributors Former owner Engraver Issuance type \
0 NaN NaN NaN monographic
1 NaN NaN NaN monographic
2 NaN NaN NaN monographic
3 NaN NaN NaN monographic
4 NaN NaN NaN monographic
Flickr URL \
http://www.flickr.com/photos/britishlibrary/ta...
1 http://www.flickr.com/photos/britishlibrary/ta...
2 http://www.flickr.com/photos/britishlibrary/ta...
3 http://www.flickr.com/photos/britishlibrary/ta...
4 http://www.flickr.com/photos/britishlibrary/ta...
Shelfmarks
0 British Library HMNTS 12641.b.30.
1 British Library HMNTS 12626.cc.2.
2 British Library HMNTS 12625.dd.1.
3 British Library HMNTS 10369.bbb.15.
4 British Library HMNTS 9007.d.28.

The head() method is now being used to look at the first entries in the file. You will see that some of the columns provide some information about the various books which will help the library. This information is not descriptive, but it has the following information:

1. Edition Statement
2. Corporate Author
3. Corporate Contributors
4. Former owner
5. Engraver
6. Issuance type
7. Shelfmarks

If you want to remove these columns, use the lines of code below:

```
>>>
to_drop = ['Edition Statement',
... 'Corporate Author',
... 'Corporate Contributors',
... 'Former owner',
... 'Engraver',
... 'Contributors',
... 'Issuance type',
... 'Shelfmarks']
df.drop(to_drop, inplace=True, axis=1)
```

In the above section, the columns we will remove have been listed in the data set. For this, we will use the drop() function on the object and pass the axis parameter as one and the inplace parameter as true. This will indicate to the Pandas directory that you want to update and change the object directly. The compiler looks at the directory for the values in the column for the object to be dropped.

When you look at the DataFrame now, you will see that the columns we do not want to use have been removed.

```
>>>
>>> df.head()
Identifier Publication Place Publication Date \
0 206 London 1879 [1878]
1 216 London; Virtue & Yorston 1868
2 218 London 1869
3 472 London 1851
4 480 London 1857
Publisher Title \
S. Tinsley & Co. Walter Forbes. [A novel.] By A. A
1 Virtue & Co. All for Greed. [A novel. The dedication signed...
2 Bradbury, Evans & Co. Love the Avenger. By the author of “All for
Gr...
3 James Darling Welsh Sketches, chiefly ecclesiastical, to the...
4 Wertheim & Macintosh [The World in which I live, and my place in
it...
Author Flickr URL
```

```
0 A. A. http://www.flickr.com/photos/britishlibrary/ta...
1 A., A. A. http://www.flickr.com/photos/britishlibrary/ta...
2 A., A. A. http://www.flickr.com/photos/britishlibrary/ta...
3 A., E. S. http://www.flickr.com/photos/britishlibrary/ta...
4 A., E. S. http://www.flickr.com/photos/britishlibrary/ta...
```

The column parameter can be used to remove the columns. You do not have to specify any labels which the directory would need to look at. You should consider the axis.

```
>>>
>>> df.drop(columns=to_drop, inplace=True)
```

This is a readable and more intuitive syntax. It is apparent what we are doing here. You can pass the coluns you want to retain through the argument usecols in the pd.read.csv function if you know the columns which ones they are.

Changing the Index of a Data Frame

The Index function in Pandas will allow you to extend the functionality of arrays in the NumPy directories. The function will also allow you to slice and label the data with ease. It is always good to use a unique value to identify a field in the data set. For instance, you can expect that a librarian will always input the unique identifier for every book if they need to search for the record.

```
>>>
df['Identifier'].is_unique
True
```

You can use the set_index method to replace the existing index using a column.

```
df = df.set_index('Identifier')
df.head()
Publication Place Publication Date \
206 London 1879 [1878]
216 London; Virtue & Yorston 1868
218 London 1869
```

```
472 London 1851
480 London 1857
Publisher \
206 S. Tinsley & Co.
216 Virtue & Co.
218 Bradbury, Evans & Co.
472 James Darling
480 Wertheim & Macintosh
Title Author \
206 Walter Forbes. [A novel.] By A. A A. A.
216 All for Greed. [A novel. The dedication signed... A., A. A.
218 Love the Avenger. By the author of “All for Gr... A., A. A.
472 Welsh Sketches, chiefly ecclesiastical, to the... A., E. S.
480 [The World in which I live, and my place in it... A., E. S.
Flickr URL
http://www.flickr.com/photos/britishlibrary/ta...
http://www.flickr.com/photos/britishlibrary/ta...
http://www.flickr.com/photos/britishlibrary/ta...
http://www.flickr.com/photos/britishlibrary/ta...
http://www.flickr.com/photos/britishlibrary/ta...
```

Every record in the DataFrame can be accessed using the loc[] function. This function will allow you to give every element in the cell an index based on the label. This means that you can give the record an index regardless of its position.

```
>>>
>>> df.loc[206]
Publication Place London
Publication Date 1879 [1878]
Publisher S. Tinsley & Co.
Title Walter Forbes. [A novel.] By A. A
Author A. A.
Flickr URL http://www.flickr.com/photos/britishlibrary/ta...
Name: 206, dtype: object
```

The number 206 is the first label for all the indices. If you want to access the label using its position, you can use df.iloc[0]. This function will give the element an index based on its position.

.loc[] is a class instance, and it has a special syntax that does not follow the rules of a Python instance.

In the previous sections, the index being used was RangeIndex. This function labeled the elements with an index using integers, and this is analogous to the steps performed by the in-built function range. When you pass a column name to the set_index function, you will need to use the values in Identifier to change the index. If you looked closely, you would have noticed that we now use the object returned by the df = df.set_index(…) method as the variable. This is because this method does not make any changes directly to the object, but it does return a modified copy of that object. If you want to avoid this, you should schedule the inplace parameter.

```
df.set_index('Identifier', inplace=True)
```

Clearing Fields in the Data Set

All unnecessary columns are now removed from the DataFrame, and these have been updated to indices in the DataFrame. This is a sensible approach. This section will explain to you how you can only work on specific sections in the data sets and format them to help you understand the data better. This will also help you enforce some consistency. This section will look at code to clean the fields' Publication Place and Publication Date. When you inspect the data set further, you will notice that every data type is currently dtype. This object is like the str type in Python. This data type can encapsulate every field which cannot be labeled as categorical or numerical data. It makes sense to use this since we will be working with the data sets mentioned in this chapter that have different forms of strings and other data types.

```
>>>
>>> df.get_dtype_counts()
object 6
```

You can enforce some numeric value to the publication date field. This will allow you to perform different calculations when we

need to later.

```
>>>
df.loc[1905:, 'Publication Date'].head(10) Identifier
1905 1888
1929 1839, 38-54
2836 [1897?]
2854 1865
2956 1860-63
2957 1873
3017 1866
3131 1899
4598 1814
4884 1820
Name: Publication Date, dtype: object
```

Every book can only have a single Publication Date. Use a regular expression by synthesizing these patterns. This will allow you to extract the year of publication.

```
>>>
regex = r'^(\d{4})'
```

You can find the four numbers at the start of the string using the above regular expression. This is enough for us. The example used in the section above is a raw extract of the information from the data set. This means that the backslash used in this string cannot be used as an escape character. This is the standard practice when it comes to regular expressions. (4) in the above string indicates to the compiler that the rule must be iterated four times. The \d represents a digit. The ^ depicts the beginning of the string, and the capturing group is indicated to the compiler using parentheses. This will signal to the Pandas directory that we only want to remove only a portion of the regular expression. Let us now run this regular expression or regex across the DataFrame.

```
>>>
extr = df['Publication Date'].str.extract(r'^(\d{4})', expand=False)
extr.head()
```

```
Identifier
1879
1868
1869
1851
1857
Name: Publication Date, dtype: object
```

The dtype is present in the column, which is a string object. You can obtain the numerical version of that object using the pd.to.numeric function.

```
>>>
df['Publication Date'] = pd.to_numeric(extr)
df['Publication Date'].dtype dtype('float64')
```

This will indicate that there are missing values in the output. Do not worry about this because you can now perform functions on all the valid values in the DataFrame.

```
>>>
df['Publication Date'].isnull().sum() / len(df) 0.11717147339205986
```

This is now done.

Combining NumPy and Str Methods to Clean Column Data

In the above section, you will have noticed that the df[‘Publication Date’].str is used to access the relevant operations for strings in the Pandas directory. This attribute performs functions on some regular expressions compiled by Python or Python strings. If you want to clean the Publication Place field, you will need to combine the str method from the Pandas directories with the np.where function from the NumPy directories. This means that you will be creating a loop that maintains the form of a vector in Excel. The syntax is as follows:

```
>>>
>>> np.where(condition, then, else)
```

In the above section, the term condition can hold a Boolean mask or an array-like object. The then value is to be looked at if the condition is true, and the else section is looked at if the condition is false. The .where() method will take every cell or element and check that element against the condition. If the value of the condition is True, the context in the condition will hold true, while the else condition is run if the condition is False. You can also change the code and use if-then statements documented independently or nested, allowing you to compute the values based on many conditional statements.

```
>>>
np.where(condition1, x1, np.where(condition2, x2,
np.where(condition3, x3, ...)))
```

The column Publication Place column only has string values. We will use the following functions to clean the column. The following lines of code are the results of the contents present in the column:

```
>>>
df['Publication Place'].head(10) Identifier
206 London
216 London; Virtue & Yorston
218 London
472 London
480 London
481 London
519 London
667 pp. 40. G. Bryan & Co: Oxford, 1898 London]
1143 London
Name: Publication Place, dtype: object
```

There may be some unnecessary information in some rows in the data fields which surround the Publication Place. If you look at the values closely, you will notice this is only for some rows, especially for those rows where the Publication Place is ‘Oxford’ or ‘London.’ Let us now look at two different user inputs:

```
>>>
```

```
>>> df.loc[4157862]
Publication Place Newcastle-upon-Tyne
Publication Date 1867
Publisher T. Fordyce
Title Local Records; or, Historical Register of rema...
Author T. Fordyce
Flickr URL http://www.flickr.com/photos/britishlibrary/ta...
Name: 4157862, dtype: object
>>> df.loc[4159587]
Publication Place Newcastle upon Tyne
Publication Date 1834
Publisher Mackenzie & Dent
Title An historical, topographical and descriptive v...
Author E. (Eneas) Mackenzie
Flickr URL http://www.flickr.com/photos/britishlibrary/ta...
Name: 4159587, dtype: object
```

The books in the above list were published at the same time and place. However, one has a hyphen and the other does not include it. You can use the str.contains() function to clean the data in the column in one shot. Use the code below to clean the data in the columns:

```
>>>
pub = df['Publication Place']
london = pub.str.contains('London')
london[:5]
Identifier
True
True
True
True
True
Name: Publication Place, dtype: bool
>>> oxford = pub.str.contains('Oxford')
These can be combined using the method np.where.
>>>
df['Publication Place'] = np.where(london, 'London',
np.where(oxford, 'Oxford',
pub.str.replace('-', ' ')))
df['Publication Place'].head() Identifier
```

```
206 London
216 London
218 London
472 London
480 London
Name: Publication Place, dtype: object
```

In the example above, we use the np.where() function to create the nested structure. The condition in this structure is a series of Boolean values obtained using the function str.contains(). When you use the contains() method, the compiler will look for the entity in the substring of a string or a variable that can be iterated. This method is like the keyword. The replacement that you will need to use is a string that will represent the Publication Place. You can also replace a hyphen using a space. This can be done using the str.replace() function, and the column can be reassigned in the DataFrame. There is indeed a lot of messier data in the DataFrame, but we will look at these columns for now. Let us now look at the first five rows in the DataFrame, which looked better than they did when we started,

```
>>>
>>> df.head()
Publication Place Publication Date Publisher \
206 London 1879 S. Tinsley & Co.
216 London 1868 Virtue & Co.
218 London 1869 Bradbury, Evans & Co.
472 London 1851 James Darling
480 London 1857 Wertheim & Macintosh
Title Author \
206 Walter Forbes. [A novel.] By A. A AA
216 All for Greed. [A novel. The dedication signed... A. A A.
218 Love the Avenger. By the author of “All for Gr... A. A A.
472 Welsh Sketches, chiefly ecclesiastical, to the... E. S A.
480 [The World in which I live, and my place in it... E. S A.
Flickr URL
http://www.flickr.com/photos/britishlibrary/ta...
http://www.flickr.com/photos/britishlibrary/ta...
http://www.flickr.com/photos/britishlibrary/ta...
```

http://www.flickr.com/photos/britishlibrary/ta...
http://www.flickr.com/photos/britishlibrary/ta...

Cleaning the Entire Data Set Using the applymap()

In some instances, you will see that the "mess" is across the whole data set, not just in one column. You may need to apply a function to every cell or element in the data frame in some instances. The applymap() function can be used to apply a function to every cell or element in the DataFrame, and it works in the same way as the map() function. In the following example, we will use the university_towns.txt to create a DataFrame.

```
head Data sets/univerisity_towns.txt Alabama[edit]
Auburn (Auburn University)[1] Florence (University of North
Alabama) Jacksonville (Jacksonville State University)[2] Livingston
(University of West Alabama)[2] Montevallo (University of
Montevallo)[2] Troy (Troy University)[2]
Tuscaloosa (University of Alabama, Stillman College, Shelton State)
[3][4] Tuskegee (Tuskegee University)[5]
Alaska[edit]
```

On running the above code, you note every periodic state will have a university town name against it—for example, StateA TownA1 TownA2. If you look at how the names of the states are written, you will see that each one of them has the "[edit]" substring. You can also wrap the list around the DataFrame using a tuple.

```
>>>
university_towns = []
with open('Data sets/university_towns.txt') as file:
... for line in file:
... if '[edit]' in line:
... # Remember this `state` until the next is found
... state = line
... else:
... # Otherwise, we have a city; keep `state` as last-seen
... university_towns.append((state, line))
university_towns[:5]
```

```
[('Alabama[edit]\n', 'Auburn (Auburn University)[1]\n'),
('Alabama[edit]\n', 'Florence (University of North Alabama)\n'),
('Alabama[edit]\n', 'Jacksonville (Jacksonville State University)
[2]\n'), ('Alabama[edit]\n', 'Livingston (University of West Alabama)
[2]\n'), ('Alabama[edit]\n', 'Montevallo (University of Montevallo)
[2]\n')]
```

You can wrap the list in the DataFrame and set the columns to RegionName and State. The Pandas directory will take every element from the list above and set the values on the left to State and those on the right to RegionName. Your DataFrame will have the following manner:

```
>>>
towns_df = pd.DataFrame(university_towns,
... columns=['State', 'RegionName'])
towns_df.head()
State RegionName
0 Alabama[edit]\n Auburn (Auburn University)[1]\n
1 Alabama[edit]\n Florence (University of North Alabama)\n
2 Alabama[edit]\n Jacksonville (Jacksonville State University)[2]\n
3 Alabama[edit]\n Livingston (University of West Alabama)[2]\n
4 Alabama[edit]\n Montevallo (University of Montevallo)[2]\n
```

It is best to fix or clean the strings in the above loops. You can use the Pandas directory for this. The only aspect you would need to consider is the use of the.str() method. In this instance, you can also use the applymap() method to map every element in the DataFrame to a Python callable. Consider the example below to understand this concept better:

```
>>>
1
Mock Data set
1 Python Pandas
2 Real Python
3 NumPy Clean
```

In the example above, we are looking at each cell in the data frame as an independent element. Therefore, the applymap()

function will look at each of these cells independently and apply the necessary function. We will now be defining these functions:

```
>>>
def get_citystate(item):
... if ' (' in item:
... return item[:item.find(' (')]
... elif '[' in item:
... return item[:item.find('[')]
... else:
... return item
```

The applymap() function in the Pandas directory only takes one parameter. This parameter is the function, also called the callable, which can be applied to every element.

```
>>>
>>> towns_df = towns_df.applymap(get_citystate)
```

The first thing to do is to define the function, where you need to select the parameter. This parameter will be an element present in the dataset. The compiler then performs a check within the function to determine whether the element contains a “(“ or “[“ or not. The values are mapped to specific function elements based on the check’s value. The applymap() function will then be called on the object. Now, if you run the following lines of code, you will see the output will be very neat.

```
>>>
towns_df.head() State RegionName
0 Alabama Auburn
1 Alabama Florence
2 Alabama Jacksonville
3 Alabama Livingston
4 Alabama Montevallo
```

In the above section of the code, the applymap() method looked at every element in the DataFrame and passed that element to the function, after which the returned value was used to replace the original value. It is this simple. The applymap() method is

both a convenient and versatile method. This method can, however, take some time to run for large data sets. This is because it maps every individual element to a Python callable. Therefore, it is always good to use vectorized operations that use the NumPy or Cython directories.

Renaming Columns and Skipping Rows

There will be some situations where you need to remove any unimportant and unnecessary information in the columns. These also may be difficult for you to understand. For example, some definitions or some footnotes in the data set are not necessary for you to use. In this case, you should skip those rows or rename the columns to remove unnecessary information or work with sensible labels. The following lines of code will look at the rows in the Olympics.csv.

```
head -n 5 Data sets/olympics.csv
0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15
,? Summer,01 !,02 !,03 !,Total,? Winter,01 !,02 !,03 !,Total,?
Games,01 !,02 !,03 !,Combined total Afghanistan
(AFG),13,0,0,2,2,0,0,0,0,0,13,0,0,2,2
Algeria (ALG),12,5,2,8,15,3,0,0,0,0,15,5,2,8,15
Argentina (ARG),23,18,24,28,70,18,0,0,0,0,41,18,24,28,70 Now, we'll
read it into a Pandas DataFrame:
>>>
>>> olympics_df = pd.read_csv('Data sets/olympics.csv')
>>> olympics_df.head()
0 1 2 3 4 5 6 7 8 \
0 NaN ? Summer 01 ! 02 ! 03 ! Total ? Winter 01 ! 02 ! 1 Afghanistan
(AFG) 13 0 0 2 2 0 0 0
2 Algeria (ALG) 12 5 2 8 15 3 0 0
3 Argentina (ARG) 23 18 24 28 70 18 0 0 4 Armenia (ARM) 5 1 2 9 12 6
0 0
9 10 11 12 13 14 15
0 03 ! Total ? Games 01 ! 02 ! 03 ! Combined total
1 0 0 13 0 0 2 2
2 0 0 15 5 2 8 15
3 0 0 41 18 24 28 70
4 0 0 11 1 2 9 12
```

This is definitely a messy line of code and output. The columns are all set to the zeroth index, and the string is an integer. The row that you are using to set the names of the other columns is at the Olympics_df.iloc[0]. This happened since the file being used has an index starting at zero and ending at 15.

If you want to go to the data set source, you will see that Nan above a column will represent Country, “? Summer” will represent the Summer Games, “01!” represents Gold, etc. Therefore, we will need to do the following:

Using the read.csv() function, you can pass the parameters you want to use to skip any rows in the data set. Ensure you use the index for the headers. The function read.csv uses many parameters, but in this instance, we only need to remove the first row (which has the index zero).

```
>>> olympics_df = pd.read_csv('Data sets/olympics.csv', header=1)
olympics_df.head()
Unnamed: 0 ? Summer 01 ! 02 ! 03 ! Total ? Winter \ 0 Afghanistan
(AFG) 13 0 0 2 2 0 1 Algeria (ALG) 12 5 2 8 15 3
2 Argentina (ARG) 23 18 24 28 70 18
3 Armenia (ARM) 5 1 2 9 12 6
4 Australasia (ANZ) [ANZ] 2 3 4 5 12 0
01 !.1 02 !.1 03 !.1 Total.1 ? Games 01 !.2 02 !.2 03 !.2 \
0 0 0 0 0 13 0 0 2
1 0 0 0 0 15 5 2 8
2 0 0 0 0 41 18 24 28
3 0 0 0 0 11 1 2 9
4 0 0 0 0 2 3 4 5
Combined total
2
1 15
2 70
3 12
4 12
```

The data set no longer has any unnecessary rows, and the correct names have been used for all fields. You should see how the

Pandas library has changed the name of the Countries column from NaN to Unnamed: 0.

If you want to rename any columns, use the rename() method in the DataFrame. This will allow you to re-label any axis in the data set using a mapping. In this instance, we will need to use dict. We must first define the dictionary, which will map the existing names of the columns to the usable names present in the dictionary.

```
>>>
new_names = {'Unnamed: 0': 'Country',
... '? Summer': 'Summer Olympics',
... '01 !': 'Gold',
... '02 !': 'Silver',
... '03 !': 'Bronze',
... '? Winter': 'Winter Olympics',
... '01 !.1': 'Gold.1',
... '02 !.1': 'Silver.1',
... '03 !.1': 'Bronze.1',
... '? Games': '# Games',
... '01 !.2': 'Gold.2',
... '02 !.2': 'Silver.2',
... '03 !.2': 'Bronze.2'}
```

We will now call upon the rename() function on the object.

```
>>>
>>> olympics_df.rename(columns=new_names, inplace=True)
```

If you want to make changes directly to the object in the code, set the inplace value to true. See how this would work for us:

```
>>>
>>> olympics_df.head()
Country Summer Olympics Gold Silver Bronze Total \
0 Afghanistan (AFG) 13 0 0 2 2
1 Algeria (ALG) 12 5 2 8 15
2 Argentina (ARG) 23 18 24 28 70
3 Armenia (ARM) 5 1 2 9 12
4 Australasia (ANZ) [ANZ] 2 3 4 5 12
Winter Olympics Gold.1 Silver.1 Bronze.1 Total.1 # Games Gold.2 \
0 0 0 0 0 0 13 0
1 3 0 0 0 0 15 5
```

2 18 0 0 0 0 41 18
3 6 0 0 0 0 11 1
4 0 0 0 0 0 2 3
Silver.2 Bronze.2 Combined total
0 0 2 2
1 2 8 15
2 24 28 70
3 2 9 12
4 4 5 12

Python Data Cleaning: Recap and Resources

In this chapter, you gathered information on how you can remove any unnecessary fields in a data set and set an index for every field in the data set to make it easier for you to access it. You also learned how to use the applymap() function to clean the complete data set and use the .str() accessor to clean specific object fields. It is important to know how to clean data since it is a big part of data analytics. You now know how you can use NumPy and Pandas to clean different data sets.

Manipulating Data Using Python

Now that you understand the libraries, we will see how you can use the libraries to manipulate and work with data.

NumPy

#load the library and check its version, just to make sure we aren't using an older version

```
import numpy as np
np.__version__
'1.12.1'
#create a list comprising numbers from 0 to 9
L = list(range(10))
#converting integers to string - this style of handling lists is known as list comprehension.
#List comprehension offers a versatile way to handle list manipulations tasks easily. We'll learn about them in future tutorials. Here's an example.
```

```
[str(c) for c in L]
['0', '1', '2', '3', '4', '5', '6', '7', '8', '9']
[type(item) for item in L]
[int, int, int, int, int, int, int, int, int, int]
```

Creating Arrays

An array is a homogeneous data type in the sense that it can only hold variables of the same data type. This holds true for arrays in NumPy as well.

```
#creating arrays
np.zeros(10, dtype='int')
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
#creating a 3 row x 5 column matrix
np.ones((3,5), dtype=float)
array([[ 1., 1., 1., 1., 1.],
1., 1., 1., 1., 1.],
1., 1., 1., 1., 1.]])
#creating a matrix with a predefined value np.full((3,5),1.23)
array([[ 1.23, 1.23, 1.23, 1.23, 1.23],
1.23, 1.23, 1.23, 1.23, 1.23],
1.23, 1.23, 1.23, 1.23, 1.23]]) #create an array with a set sequence
np.arange(0, 20, 2)
array([0, 2, 4, 6, 8,10,12,14,16,18])
#create an array of even space between the given range of values
np.linspace(0, 1, 5)
array([ 0., 0.25, 0.5 , 0.75, 1.])
#create a 3x3 array with mean 0 and standard deviation 1 in a given
dimension np.random.normal(0, 1, (3,3))
array([[ 0.72432142, -0.90024075, 0.27363808],
0.88426129, 1.45096856, -1.03547109], [-0.42930994, -1.02284441,
-1.59753603]]) #create an identity matrix
np.eye(3) array([[ 1., 0., 0.],
0., 1., 0.],
0., 0., 1.]])
#set a random seed np.random.seed(0)
x1 = np.random.randint(10, size=6) #one dimension
x2 = np.random.randint(10, size=(3,4)) #two dimension
x3 = np.random.randint(10, size=(3,4,5)) #three dimension
print("x3 ndim:", x3.ndim)
```

```
print("x3 shape:", x3.shape)
print("x3 size: ", x3.size)
('x3 ndim:', 3)
('x3 shape:', (3, 4, 5))
('x3 size: ', 60)
```

Array Indexing

If you are familiar with programming languages, you will be aware that the indexing in an array always begins at zero.

```
x1 = np.array([4, 3, 4, 4, 8, 4])
x1
array([4, 3, 4, 4, 8, 4])
#assess value to index zero
x1[0]
4
#assess fifth value
x1[4]
8
#get the last value
x1[-1]
4
#get the second last value
x1[-2]
8
#in a multidimensional array, we need to specify row and column
index x2
array([[3, 7, 5, 5],
[0, 1, 5, 9],
[3, 0, 5, 0]])
#1st row and 2nd column value
x2[2,3]
0
#3rd row and last value from the 3rd column
x2[2,-1]
0
#replace value at 0,0 index
x2[0,0] = 12
x2
array([[12, 7, 5, 5],
```

```
0, 1, 5, 9],
3, 0, 5, 0]])
```

Array Slicing

You can slice an array to access a specific element or a range of elements within an array.

```
x = np.arange(10)
x
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
#from start to 4th position
x[:5]
array([0, 1, 2, 3, 4])
#from 4th position to end
x[4:]
array([4, 5, 6, 7, 8, 9])
#from 4th to 6th position
x[4:7]
array([4, 5, 6])
#return elements at even place
x[ : : 2]
array([0, 2, 4, 6, 8])
#return elements from first position step by two
x[1::2]
array([1, 3, 5, 7, 9])
#reverse the array
x[::-1]
array([9, 8, 7, 6, 5, 4, 3, 2, 1, 0])
```

Array Concatenation

It is useful to combine multiple arrays to perform complex operations. You will not have to type the elements in the different arrays, but can instead concatenate those arrays to handle these complex operations with ease.

```
#You can concatenate two or more arrays at once.
x = np.array([1, 2, 3])
y = np.array([3, 2, 1])
z = [21,21,21]
np.concatenate([x, y,z])
```

```
array([ 1, 2, 3, 3, 2, 1, 21, 21, 21])
#You can also use this function to create 2-dimensional arrays.
grid = np.array([[1,2,3],[4,5,6]])
np.concatenate([grid,grid])
array([[1, 2, 3],
[4, 5, 6],
[1, 2, 3],
[4, 5, 6]])
#Using its axis parameter, you can define row-wise or column-wise
matrix np.concatenate([grid,grid],axis=1)
array([[1, 2, 3, 1, 2, 3],
[4, 5, 6, 4, 5, 6]])
```

In the above code, we have used the concatenation function on those arrays that have variables of the same data type and the same dimensions. So, what if you need to combine a one-dimensional array with a two-dimensional array? You can use the np.vstack or the np.hstack functions in such instances. The next section of code will help you understand how this can be done.

```
x = np.array([3,4,5])
grid = np.array([[1,2,3],[17,18,19]])
np.vstack([x,grid])
array([[ 3, 4, 5],
1, 2, 3], [17, 18, 19]])
#Similarly, you can add an array using np.hstack z = np.array([[9],
[9]])
np.hstack([grid,z]) array([[ 1, 2, 3, 9], [17, 18, 19, 9]])
It is always a good idea to use a predefined position or condition to
split an array.
x = np.arange(10)
x
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9]) x1,x2,x3 = np.split(x,[3,6]) print
x1,x2,x3
[0 1 2] [3 4 5] [6 7 8 9]
grid = np.arange(16).reshape((4,4)) grid
upper,lower = np.vsplit(grid,[2]) print (upper, lower)
(array([[0, 1, 2, 3],
[4, 5, 6, 7]]), array([[ 8, 9, 10, 11], [12, 13, 14, 15]]))
```

The NumPy directory gives you access to numerous mathematical functions apart from the functions that have been used in the examples above. Some of these functions include sum, divide, abs, multiple, mod, power, log, sin, tan, cos, mean, min, max, var, etc. You can use these functions to perform numerous arithmetic calculations. To learn more about these functions, you should refer to the NumPy documentation to learn more about the functions. Let us now take a look at how you can manipulate data using the Pandas library. Ensure that you look at every line in the code carefully before you manipulate the data.

Using Pandas

```
#load library - pd is just an alias. I used pd because it's short and
literally abbreviates pandas.
#You can use any name as an alias.
import pandas as pd
#create a data frame - dictionary is used here where keys get
converted to column names and values to row values.
data = pd.DataFrame({'Country':
['Russia','Colombia','Chile','Ecuador','Nigeria'],
Rank':[121,40,100,130,11]})
data
CountryRank
0Russia121
1Colombia40
2Chile100
3Ecuador130
4Nigeria11
#We can do a quick analysis of any data set using:
data.describe()
Rank
count5.000000
mean80.400000
std52.300096
min11.000000
25%40.000000
50%100.000000
75%121.000000
max130.000000
```

You can obtain the summary of the statistics of only the integer and double variables using the describe() method. If you want to obtain all the information available about the data set, you should use the function named info().

```
#Among other things, it shows the data set has 5 rows and 2
columns with their respective names. data.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5 entries, 0 to 4
Data columns (total 2 columns):
Country 5 non-null object
Rank 5 non-null int64
dtypes: int64(1), object(1)
memory usage: 152.0+ bytes
#Let's create another data frame.
data = pd.DataFrame({'group':['a', 'a', 'a', 'b','b', 'b', 'c',
'c','c'],'ounces':[4, 3, 12, 6, 7.5, 8, 3, 5, 6]})
data
groupounces
0a4.0
1a3.0
2a12.0
3b6.0
4b7.5
5b8.0
6c3.0
7c5.0
8c6.0
#Let's sort the data frame by ounces - inplace = True will make
changes to the data data.sort_values(by=
['ounces'],ascending=True,inplace=False) groupounces
1a3.0
6c3.0
0a4.0
7c5.0
3b6.0
8c6.0
4b7.5
5b8.0
2a12.0
```

The data can now be sorted using numerous columns.

```
data.sort_values(by=['group','ounces'],ascending=
[True,False],inplace=False)
groupounces
2a12.0
0a4.0
1a3.0
5b8.0
4b7.5
3b6.0
8c6.0
7c5.0
6c3.0
```

Most data sets have duplicate rows and columns, and these duplicate rows are called noise. For this reason, you should always remove all the inconsistencies in the data set before you feed the model with the data set. Let us look at an alternative way to remove the duplicates in the data set.

```
#create another data with duplicate rows
data = pd.DataFrame({'k1':['one']*3 + ['two']*4, 'k2':[3,2,1,3,3,4,4]})
data
k1k2
0one3
1one2
2one1
3two3
4two3
5two4
6two4
#sort values
data.sort_values(by='k2')
k1k2
2one1
1one2
0one3
3two3
4two3
5two4
```

```
6two4
#remove duplicates - ta da!
data.drop_duplicates()
k1k2
0one3
1one2
2one1
3two3
5two4
```

Here, the values in the rows and columns have been matched to remove the duplicates in the data set. You can remove the duplicate values if needed using parameters. These parameters can be specific columns. Let us look at how the duplicates in Column K can be removed.

```
data.drop_duplicates(subset='k1')
k1k2
0one3
3two3
```

Let us now understand how the data can be categorized based on some predefined rules or criteria. This will often happen when you work on processing data, especially that data which needs to be categorized. For example, if you have a column with the names of countries, you want to split the countries into separate columns based on their continents. To do this, you will need to create a new variable. The code below will help you achieve this with ease.

```
data = pd.DataFrame({'food': ['bacon', 'pulled pork', 'bacon',
'Pastrami','corned beef', 'Bacon', 'pastrami', 'honey ham','nova
lox'],
'ounces': [4, 3, 12, 6, 7.5, 8, 3, 5, 6]})
data
foodounces
0bacon4.0
1pulled pork3.0
2bacon12.0
3Pastrami6.0
```

```
4corned beef7.5
5Bacon8.0
6pastrami3.0
7honey ham5.0
8nova lox6.0
```

Let us now look at how we can create a new variable that will help the model predict which animal will be the source of food for another animal. For the model to do this, we should first map the food to the animals by using the dictionary. The map function can then be used to pull the values in from the dictionary. The code below will help us achieve this.

```
meat_to_animal = {
'bacon': 'pig',
'pulled pork': 'pig',
'pastrami': 'cow',
'corned beef': 'cow',
'honey ham': 'pig',
'nova lox': 'salmon'
}
def meat_2_animal(series):
if series['food'] == 'bacon':
return 'pig'
elif series['food'] == 'pulled pork':
return 'pig'
elif series['food'] == 'pastrami':
return 'cow'
elif series['food'] == 'corned beef':
return 'cow'
elif series['food'] == 'honey ham':
return 'pig'
else:
return 'salmon'
#create a new variable
data['animal'] = data['food'].map(str.lower).map(meat_to_animal)
data
foodouncesanimal
0bacon4.0pig
1pulled pork3.0pig
```

```
2bacon12.0pig
3Pastrami6.0cow
4corned beef7.5cow
5Bacon8.0pig
6pastrami3.0cow
7honey ham5.0pig
8nova lox6.0salmon
#another way of doing it is: convert the food values to the lower
case and apply the function
lower = lambda x: x.lower()
data['food'] = data['food'].apply(lower)
data['animal2'] = data.apply(meat_2_animal, axis='columns')
data
foodouncesanimalanimal2
0bacon4.0pigpig
1pulled pork3.0pigpig
2bacon12.0pigpig
3pastrami6.0cowcow
4corned beef7.5cowcow
5bacon8.0pigpig
6pastrami3.0cowcow
7honey ham5.0pigpig
8nova lox6.0salmonsalmon
```

Alternatively, you can use the assign function if you wish to create another variable. You will come across numerous functions in this chapter that you should keep in mind. This section will help you understand how you can use the Pandas library to solve different data analytics problems. This library has numerous packages you can use to handle large volumes of data.

```
data.assign(new_variable = data['ounces']*10)
foodouncesanimalanimal2new_variable
0bacon4.0pigpig40.0
1pulled pork3.0pigpig30.0
2bacon12.0pigpig120.0
3pastrami6.0cowcow60.0
4corned beef7.5cowcow75.0
5bacon8.0pigpig80.0
6pastrami3.0cowcow30.0
```

```
7honey ham5.0pigpig50.0
8nova lox6.0salmonsalmon60.0
Let us now remove the animal2 column from the data frame.
data.drop('animal2',axis='columns',inplace=True)
data
foodouncesanimal
0bacon4.0pig
1pulled pork3.0pig
2bacon12.0pig
3Pastrami6.0cow
4corned beef7.5cow
5Bacon8.0pig
6pastrami3.0cow
7honey ham5.0pig
8nova lox6.0salmon
```

If you scrape and load information from the Internet, you will have some missing information in the data set. You can choose to substitute the missing variable with a dummy variable. Alternatively, you can use a default value depending on the type of problem you are addressing. It can also be that there are many outliers in the data set that you want to get rid of. You can do this using the Pandas directory.

```
#Series function from pandas are used to create arrays
data = pd.Series([1., -999., 2., -999., -1000., 3.])
data
0 1.0
-999.0
2 2.0
3 -999.0
4 -1000.0
5 3.0
dtype: float64
#replace -999 with NaN values data.replace(-999,
np.nan,inplace=True) data
0 1.0
NaN
2.0
3 NaN
```

4 -1000.0
5 3.0
dtype: float64
#We can also replace multiple values at once. data = pd.Series([1., -999., 2., -999., -1000., 3.])
data.replace([-999,-1000],np.nan,inplace=True) data
0 1.0
1 NaN
2.0
NaN
4 NaN
5 3.0
dtype: float64

Let us now look at how we can rename the rows and columns.

data = pd.DataFrame(np.arange(12).reshape((3, 4)),index=['Ohio', 'Colorado', 'New York'],columns=['one', 'two', 'three', 'four'])
data
onetwothreefour
Ohio0123
Colorado4567
New York891011
Using rename function
data.rename(index = {'Ohio':'SanF'}, columns= {'one':'one_p','two':'two_p'},inplace=True) data
one_ptwo_pthreefour
SanF0123
Colorado4567
New York891011
#You can also use string functions
data.rename(index = str.upper, columns=str.title,inplace=True)
data
One_pTwo_pThreeFour
SANF0123
COLORADO4567
NEW YORK891011
We will need to split or categorize the continuous variables.
ages = [20, 22, 25, 27, 21, 23, 37, 31, 61, 45, 41, 32]
Let us now divide the ages into smaller segments or bins like 18-25, 26-35,36-60 and 60 and above.

#Understand the output - '(' means the value is included in the bin, '[' means the value is excluded
bins = [18, 25, 35, 60, 100]
cats = pd.cut(ages, bins)
cats
[(18, 25], (18, 25], (18, 25], (25, 35], (18, 25], ..., (25, 35], (60, 100], (35, 60], (35, 60], (25, 35]] Length: 12
Categories (4, object): [(18, 25] < (25, 35] < (35, 60] < (60, 100]] #To include the right bin value, we can do:
pd.cut(ages,bins,right=False)
[[18, 25), [18, 25), [25, 35), [25, 35), [18, 25), ..., [25, 35), [60, 100), [35, 60), [35, 60), [25, 35)] Length: 12
Categories (4, object): [[18, 25) < [25, 35) < [35, 60) < [60, 100)]
#pandas library intrinsically assigns an encoding to categorical variables. cats.labels
array([0, 0, 0, 1, 0, 0, 2, 1, 3, 2, 2, 1], dtype=int8)
#Let's check how many observations fall under each bin
pd.value_counts(cats)
(18, 25] 5
(35, 60] 3
(25, 35] 3
(60, 100] 1
dtype: int64
A label name can also be passed in the code.
bin_names = ['Youth', 'YoungAdult', 'MiddleAge', 'Senior'] new_cats = pd.cut(ages, bins,labels=bin_names)
pd.value_counts(new_cats)
Youth5
MiddleAge3
YoungAdult3
Senior1
dtype: int64
#we can also calculate their cumulative sum
pd.value_counts(new_cats).cumsum()
Youth5
MiddleAge3
YoungAdult3
Senior1
dtype: int64

Alternatively, you can create and use pivots to look at different variables. You also must look at the data sets in the Pandas directories. A pivot table is one of the easiest ways to perform an analysis of the data set, and it is for this reason that it is essential that you understand how this can be done.

```
df = pd.DataFrame({'key1' : ['a', 'a', 'b', 'b', 'a'],
'key2' : ['one', 'two', 'one', 'two', 'one'],
'data1' : np.random.randn(5),
'data2' : np.random.randn(5)})
df
data1data2key1key2
00.9735990.001761a
10.207283-0.990160a
21.0996421.872394b
30.939897-0.241074b
40.6063890.053345a
#calculate the mean of data1 column by key1
grouped = df['data1'].groupby(df['key1'])
grouped.mean()
key1
0.595757 b 1.019769
Name: data1, dtype: float64
#We will now slice the data frame
dates = pd.date_range('20130101',periods=6)
df =
pd.DataFrame(np.random.randn(6,4),index=dates,columns=list('AB
CD')) df
ABCD
2013-01-011.030816-1.2769890.837720-1.490111 2013-01-02-
1.070215-0.2091290.604572-1.743058
2013-01-031.5242271.8635751.2913781.300696
2013-01-040.918203-0.158800-0.964063-1.990779
2013-01-050.0897310.114854-0.5858150.298772
2013-01-060.2222600.435183-0.0457480.049898 #get first n rows
from the data frame df[:3]
ABCD
2013-01-011.030816-1.2769890.837720-1.490111
2013-01-02-1.070215-0.2091290.604572-1.743058
2013-01-031.5242271.8635751.2913781.300696
```

```
#slice based on date range
df['20130101':'20130104']
ABCD
2013-01-011.030816-1.2769890.837720-1.490111
2013-01-02-1.070215-0.2091290.604572-1.743058
2013-01-031.5242271.8635751.2913781.300696
2013-01-040.918203-0.158800-0.964063-1.990779 #slicing based on
column names df.loc[:,['A','B']]
AB
2013-01-011.030816-1.276989
2013-01-02-1.070215-0.209129
2013-01-031.5242271.863575
2013-01-040.918203-0.158800
2013-01-050.0897310.114854
2013-01-060.2222600.435183
#slicing based on both row index labels and column names
df.loc['20130102':'20130103',['A','B']] AB
2013-01-02-1.070215-0.209129
2013-01-031.5242271.863575
#slicing based on index of columns
df.iloc[3] #returns 4th row (index is 3rd)
0.918203 B -0.158800 C -0.964063 D -1.990779
Name: 2013-01-04 00:00:00, dtype: float64 #returns a specific range
of rows df.iloc[2:4, 0:2]
AB
2013-01-031.5242271.863575
2013-01-040.918203-0.158800
#returns specific rows and columns using lists containing columns
or row indexes
df.iloc[[1,5],[0,2]]
AC
2013-01-02-1.0702150.604572
2013-01-060.222260-0.045748
```

Using the Boolean indexing method, you can check the values in the columns. This will also make it easier for you to filter the required information based on the conditions you have set. Bear in mind these conditions must be defined in the beginning of the code.

```
df[df.A > 1]
```

ABCD
2013-01-011.030816-1.2769890.837720-1.490111
2013-01-031.5242271.8635751.2913781.300696
#we can copy the data set
df2 = df.copy()
df2['E']=['one', 'one','two','three','four','three']
df2
ABCDE
2013-01-011.030816-1.2769890.837720-1.490111one
2013-01-02-1.070215-0.2091290.604572-1.743058one
2013-01-031.5242271.8635751.2913781.300696two
2013-01-040.918203-0.158800-0.964063-1.990779three
2013-01-050.0897310.114854-0.5858150.298772four
2013-01-060.2222600.435183-0.0457480.049898three #select rows based on column values df2[df2['E'].isin(['two','four'])] ABCDE
2013-01-031.5242271.8635751.2913781.300696two
2013-01-050.0897310.114854-0.5858150.298772four #select all rows except those with two and four
df2[~df2['E'].isin(['two','four'])] ABCDE
2013-01-011.030816-1.2769890.837720-1.490111one
2013-01-02-1.070215-0.2091290.604572-1.743058one
2013-01-040.918203-0.158800-0.964063-1.990779three
2013-01-060.2222600.435183-0.0457480.049898three
Using the query method, you can also select the relevant columns needed. You also need to determine the criteria you want to consider.
#list all columns where A is greater than C
df.query('A > C')
ABCD
2013-01-011.030816-1.2769890.837720-1.490111
2013-01-031.5242271.8635751.2913781.300696
2013-01-040.918203-0.158800-0.964063-1.990779
2013-01-050.0897310.114854-0.5858150.298772
2013-01-060.2222600.435183-0.0457480.049898
#using OR condition
df.query('A < B | C > A')
ABCD
2013-01-02-1.070215-0.2091290.604572-1.743058
2013-01-031.5242271.8635751.2913781.300696
2013-01-050.0897310.114854-0.5858150.298772
2013-01-060.2222600.435183-0.0457480.049898

Using a pivot table, you can customize the information in the DataFrame to ensure you understand the information in the data set better. Most engineers use Excel to build these pivot tables, and this is an easy way to understand the data and information extracted.

```
#create a data frame
data = pd.DataFrame({'group': ['a', 'a', 'a', 'b','b', 'b', 'c', 'c','c'],
'ounces': [4, 3, 12, 6, 7.5, 8, 3, 5, 6]})
data
groupounces
0a4.0
1a3.0
2a12.0
3b6.0
4b7.5
5b8.0
6c3.0
7c5.0
8c6.0
#calculate means of each group
data.pivot_table(values='ounces',index='group',aggfunc=np.mean)
group
6.333333 b 7.166667 c 4.666667
Name: ounces, dtype: float64 #calculate count by each group
data.pivot_table(values='ounces',index='group',aggfunc='count')
group
3
3 c 3
Name: ounces, dtype: int64
```

We have now understood the basics of both the NumPy and Pandas libraries in Python. Now, let us see how we can use the functions and libraries we have discussed thus far in the book.

Exploring the Data Set

We are going to work with a data set with information about adults in this section. You can download this data set from the following location: https://s3-ap-southeast-

1.amazonaws.com/he-public-data/datafiles19cdaf8.zip. The data set poses a binary classification problem, and the objective is to calculate the salary of an individual using some variables.

```
#load the data
train = pd.read_csv("~/Adult/train.csv")
test = pd.read_csv("~/Adult/test.csv")
#check data set
train.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 32561 entries, 0 to 32560
Data columns (total 15 columns):
age 32561 non-null int64
workclass 30725 non-null object
fnlwgt 32561 non-null int64
education 32561 non-null object
education.num 32561 non-null int64
marital.status 32561 non-null object
occupation 30718 non-null object
relationship 32561 non-null object
race 32561 non-null object
sex 32561 non-null object
capital.gain 32561 non-null int64
capital.loss 32561 non-null int64
hours.per.week 32561 non-null int64
native.country 31978 non-null object
target 32561 non-null object
dtypes: int64(6), object(9)
memory usage: 3.7+ MB
```

The training data set that we are using for this example has 32561 rows and 15 columns. Out of these fifteen columns only six columns have integer data, while the other columns have object or character data. You can also check the test data set in a similar manner. The number of rows and columns in the data set can also be identified using the following code:

```
print ("The train data has",train.shape)
print ("The test data has",test.shape)
('The train data has', (32561, 15))
('The test data has', (16281, 15))
#Let have a glimpse of the data set
train.head()
```

ageworkclassfnlwgteducationeducation.nummarital.statusoccupat
ionrelationshipracesexcapital.gaincapit
al.losshours.per.weeknative.countrytarget
039State-gov77516Bachelors13Never-marriedAdm-clericalNot-in-
familyWhiteMale2174040United-States<=50K
150Self-emp-not-inc83311Bachelors13Married-civ-spouseExec-
managerialHusbandWhiteMale0013United-States<=50K
238Private215646HS-grad9DivorcedHandlers-cleanersNot-in-
familyWhiteMale0040United-States<=50K
353Private23472111th7Married-civ-spouseHandlers-
cleanersHusbandBlackMale0040United-States<=50K
428Private338409Bachelors13Married-civ-spouseProf-
specialtyWifeBlackFemale0040Cuba<=50K Let us verify if the data
set has any missing values. nans = train.shape[0] -
train.dropna().shape[0]
print ("%d rows have missing values in the train data" %nans)
nand = test.shape[0] - test.dropna().shape[0]
print ("%d rows have missing values in the test data" %nand) 2399
rows have missing values in the train data 1221 rows have missing
values in the test data

We should now look for those columns that have any missing values.

#only 3 columns have missing values
train.isnull().sum()
age0
workclass1836
fnlwgt0
education0
education.num0
marital.status0
occupation1843
relationship0
race0
sex0
capital.gain0
capital.loss0
hours.per.week0
native.country583
target0

```
dtype: int64
```

Using the character variables, let us now count the number of values in the data set that are unique.

```
cat = train.select_dtypes(include=['O'])
cat.apply(pd.Series.nunique)
workclass8
education16
marital.status7
occupation14
relationship6
race5
sex2
native.country41
target2
dtype: int64
```

From the earlier section, we learned how to work with missing data in the data set. Let us now substitute the missing values with other values.

```
#Education
train.workclass.value_counts(sort=True)
train.workclass.fillna('Private',inplace=True)
#Occupation
train.occupation.value_counts(sort=True)
train.occupation.fillna('Prof-specialty',inplace=True)
#Native Country
train['native.country'].value_counts(sort=True)
train['native.country'].fillna('United-States',inplace=True)
```

We will now need to check if there are any values missing in the data set.

```
train.isnull().sum()
age0
workclass0
fnlwgt0
education0
education.num0
marital.status0
```

```
occupation0
relationship0
race0
sex0
capital.gain0
capital.loss0
hours.per.week0
native.country0
target0
dtype: int64
```

We will now look at the target variable to check if there is any issue within the data set.

```
#check proportion of target variable
train.target.value_counts()/train.shape[0]
<=50K 0.75919
>50K 0.24081
Name: target, dtype: float64
```

We can see that seventy-five percent of the variables in this data set belong to the <=50k class. This means that the model will give you a result with seventy-five percent accuracy, even if it is providing you with a rough estimate. This is amazing, isn't it? Let us now create a crosstab where we will check education using the target variable. This will help us identify if the target variable is affected by education.

```
pd.crosstab(train.education,
train.target,margins=True)/train.shape[0]
target<=50K>50KAll
education
10th0.0267500.0019040.028654
11th0.0342430.0018430.036086
12th0.0122850.0010130.013298
1st-4th0.0049750.0001840.005160
5th-6th0.0097360.0004910.010227
7th-8th0.0186110.0012280.019840
9th0.0149570.0008290.015786
Assoc-acdm0.0246310.0081390.032769
Assoc-voc0.0313570.0110870.042443
```

```
Bachelors0.0962500.0682100.164461
Doctorate0.0032860.0093980.012684
HS-grad0.2710600.0514420.322502
Masters0.0234640.0294520.052916
Preschool0.0015660.0000000.001566
Prof-school0.0046990.0129910.017690
Some-college0.1813210.0425970.223918
All0.7591900.2408101.000000
```

From the above data, we can see that out of the seventy-five percent of the people with a salary above 50k, at least twenty-seven percent of them were high school graduates. This is accurate since people with a lower level of education are not expected to earn more. On the other hand, out of the twenty-five percent of the people with a salary of 50k, five percent of them are high-school graduates while six percent are bachelors. We need to be concerned with this pattern, and we should, therefore, look at all the variables before we make a conclusion.

```
#load sklearn and encode all object type variables
from sklearn import preprocessing
for x in train.columns:
if train[x].dtype == 'object':
lbl = preprocessing.LabelEncoder()
lbl.fit(list(train[x].values))
train[x] = lbl.transform(list(train[x].values))
```

We will now look at the different changes that have been applied to the data set.

```
train.head()
ageworkclassfnlwgteducationeducation.nummarital.statusoccupat
ionrelationshipracesexcapital.gaincapit
al.losshours.per.weeknative.countrytarget
039677516913401412174040380
150583311913230410013380
238321564611905141004038 0
353323472117250210040380
428333840991329520004040
```

If you pay close attention to the output, you will notice that every variable has been converted into the numeric data type.

```
#<50K = 0 and >50K = 1
train.target.value_counts()
24720
1 7841
```

Name: target, dtype: int64 Building a Random Forest Model.

Chapter Eleven

Mistakes Made by Programmers

Now that you know how to use Python and the different structures and data types to work with, it is important to understand different mistakes programmers make using Python. Learn from this chapter, and try to avoid making those mistakes.

Using Defaults as Function Arguments

When you write a function in any program, you can specify the parameters, but this is optional. Alternatively, you can provide default values to the function as parameters. This is an interesting feature, but it confuses some programmers, especially if the parameter can be changed, i.e., it is mutable. Consider the function defined in the example below:

```
>>> def foo(bar=[]):        # bar is optional and defaults to [] if not
specified
...    bar.append("baz")    # but this line could be problematic, as
we'll see...
...    return bar
```

One mistake most programmers make is they create an optional argument in the program function. However, they forget to set the default value or expression for the optional argument, which can lead to errors in the program. For instance, you may want to call the foo() function in the code above, but you will only receive the ‘baz’ output since there is no argument specified in the function. According to the compiler, the foo() function does not have an argument set to it. So, what do you think happens when you type the following code:

```
>>> foo()
["baz"]
```

```
>>> foo()
["baz", "baz"]
>>> foo()
["baz", "baz", "baz"]
```

Why do you think the compiler added the same word to the list every time the foo() function was called? Why was a new list not created every time? Experts will tell you the compiler only reviews the default value in any function only once. It reviews this when you define the function. Therefore, in the above example, the function argument ‘bar’ is given the default value, which is the empty list. This happens when the function is defined. When the compiler runs the function repeatedly, it will use the empty list to which the parameter is assigned. The best way to work around this issue is:

```
>>> def foo(bar=None):
...    if bar is None: # or if not bar:
...        bar = []
...    bar.append("baz")
...    return bar
...
>>> foo()
["baz"]
>>> foo()
["baz"]
>>> foo()
["baz"]
```

Incorrect Use of Class Variables

Look at the example below:

```
>>> class A(object):
...    x = 1
...
>>> class B(A):
...    pass
...
>>> class C(A):
...    pass
```

```
...
>>> print A.x, B.x, C.x
1 1 1
```

The above example makes sense, does it not? Now consider the example below:

```
>>> B.x = 2
>>> print A.x, B.x, C.x
1 2 1
```

Yes, this will also work, as expected. How do you think the following lines of code will work?

The output is not what you expected, is it? The only thing we did was change the A.x variable, so why did the C.x also change?

Bear in mind that the dictionaries handle class variables in Python, and these classes follow a method called MRO or Method Resolution Order. In the code above, since the compiler did not find the attribute x in the class, it will look only at the base classes. In the above examples, the base class is A, and though you can use multiple inheritances, the compiler does not function the way you would expect it to. In simple words, class c does not have its own property, and it is different from the base class. Therefore, any reference made in the code to C.x would be independent of the A.x class. This would lead to an exception in your code, and you need to add an error-handling code to the program to ensure the issues are covered.

Specifying Incorrect Parameters

Consider the following lines of code:

```
>>> try:
...     l = ["a", "b"]
...     int(l[2])
... except ValueError, IndexError:  # To catch both exceptions, right?
...     pass
```

```
...
Traceback (most recent call last):
  File "<stdin>", line 3, in <module>
IndexError: list index out of range
```

The trouble with the above code is that the except statement will not look at the exception list you have specified. In the 2.x versions of Python, *except Exception, e* is a statement you can use to bind any exception to the next parameter which you have specified in the code. This is the only way you can make the exception available for the compiler to inspect. It is for this reason the compiler does not catch the IndexError exception. The exception ends since the parameter is bound to IndexError.

The best way to catch these exceptions using an except statement is to list the parameters in a tuple containing all the exceptions to be called. It is best to use the 'as' keyword, and this is supported in all versions of Python after version 2. Consider the example below:

```
>>> try:
...     l = ["a", "b"]
...     int(l[2])
... except (ValueError, IndexError) as e:
...     pass
...
>>>
```

Misusing Scope of Pythons

The scope of any Python resolution is based on the LEGB or Local, Enclosing, Global, Built-in rule. This does seem easy to understand, does it not? There are some subtleties in understanding how Python works, and this is where the example comes below:

```
>>> x = 10
>>> def foo():
...     x += 1
...     print x
```

```
...
>>> foo()
```

Traceback (most recent call last):

```
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 2, in foo
```

UnboundLocalError: local variable 'x' referenced before assignment

So, what do you think the problem is here?

You will receive an error in the above example since the assignment variable is created within the scope. The compiler automatically considers this variable to be local only to that scope. This indicates that any variable with a similar name will not be considered as part of this function. Most programmers are surprised when they receive the UnboundLocalError, especially when they use code that worked previously. You can overcome this error only when there is an assignment statement used in the function body. It is for this reason most developers use lists to overcome the error. Consider the example below:

```
>>> lst = [1, 2, 3]
>>> def foo1():
...     lst.append(5)   # This works ok...
...
>>> foo1()
>>> lst
[1, 2, 3, 5]

>>> lst = [1, 2, 3]
>>> def foo2():
...     lst += [5]      # ... but this bombs!
...
>>> foo2()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 2, in foo
```

```
UnboundLocalError: local variable 'lst' referenced before
assignment
```

Why did the first function run perfectly but not the second function? The issue is the same that we considered above, but it is not easy to spot. The first function does not assign the variable to 1st, while the foo2 function does this. Bear in mind that 1st += [5] is a simpler version of the function 1st = 1st + [5].

In the above example, we are assigning a value to the variable 1st. The value you are assigning to the variable is based on the variable's initial value, but this value has not been defined.

Modifying and Iterating Lists

Consider the following lines of code:

```
>>> odd = lambda x : bool(x % 2)
>>> numbers = [n for n in range(10)]
>>> for i in range(len(numbers)):
...     if odd(numbers[i]):
...         del numbers[i]  # BAD: Deleting item from a list while
iterating over it
...
Traceback (most recent call last):
   File "<stdin>", line 2, in <module>
IndexError: list index out of range
```

When you delete an item from an array or list while you iterate it during the program is an aspect most programmers consider. The above example is obvious, and some advanced programmers also use these tricks. This code is often more complex than others. Python allows the use of numerous programming paradigms, and when these are used properly, you can result in streamlined or simplified code. Another benefit of using simpler code is that it is less likely to cause deletion of the list item. A list comprehension is useful to avoid this issue. Consider the following line of code:

```
>>> odd = lambda x : bool(x % 2)
>>> numbers = [n for n in range(10)]
```

```
>>> numbers[:] = [n for n in numbers if not odd(n)]  # ahh, the
beauty of it all
>>> numbers
[0, 2, 4, 6, 8]
```

Unsure of Binding Variables

Consider the example below:

```
>>> def create_multipliers():
...     return [lambda x : i * x for i in range(5)]
>>> for multiplier in create_multipliers():
...     print multiplier(2)
...
```

When you run this code, you may expect to receive the following output:

```
0
2
4
6
8
```

You will get the following output:

```
8
8
8
8
8
```

Why do you think this happens? Python has a behavior termed late binding which states the value of variables used in the program are obtained from the functions in the code. In the above code, when the functions you return are called, the value of the variable ‘i’ is taken from the other sections in the code. The solution to this issue is quite the same:

```
>>> def create_multipliers():
...     return [lambda x, i=i : i * x for i in range(5)]
...
```

```
>>> for multiplier in create_multipliers():
...     print multiplier(2)
...
0
2
4
6
8
```

There you go. The only thing you are doing is taking advantage of the arguments in the function to generate any other function you can use to obtain the required information. This is an elegant way to work with code. If you are a developer, you need to understand how to do this in any case.

Creating Circular Modules

Let us assume you have two files in your code – a.py and b.py. Both are Python-based programs, and one can be used to import the other. You can do this in the following way.

```
Consider the function a.py:
import b
def f():
   return b.x
print f()
Now, consider the second program:
import a
x = 1
def g():
   print a.f()
import a
x = 1
def g():
   print a.f()
Now, try to import the function a.py:
>>> import a
1
```

This works the way you want it, too. This probably comes as a surprise to you. You have circular imports here, and this

technically should not be a problem. The solution is the presence of a circular import that is also a problem in this programming language. If you have imported any function or module in Python, bear in mind that Python cannot re-import it. Depending on when these imports are made in the module that accesses variables and functions, it can lead to other problems.

Let us now return to the example above. The first thing we did was import the a.py program, which did not have any problem with importing the b.py program. This only happens because the b.py code does not need anything from the a.py code. This means nothing would need to be defined when the functions are being imported. The reference in the b.py program to the a program is to call the a.f() function. Note that neither a.py nor b.py refers to the g() function. What do you think happens if you import the b.py program without importing the a.py program?

```
>>> import b
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "b.py", line 1, in <module>
  import a
  File "a.py", line 6, in <module>
print f()
  File "a.py", line 4, in f
return b.x
AttributeError: 'module' object has no attribute 'x'
```

This is where it becomes difficult. The issue here is that when you import the b.py program, the compiler tries to import the a.py program, and this calls the function that tries to use the b.x variable. Unfortunately, we have not defined the b.x variable. Therefore, you receive the exception AttributeError. One solution to this problem is quite simple. All you need to do is simplify the b.py program, so it uses the g() function to import the a.py file.

```
x = 1

def g():
```

```
    import a # This will be evaluated only when g() is called
    print a.f()
```

When you import the files, everything works the way it should:

```
>>> import b
>>> b.g()
1 # Printed a first time since module 'a' calls 'print f()' at the end
1 # Printed a second time, this one is our call to 'g'
```

Forgetting to Understand the Changes Between Versions

Consider the example below:

```
import sys
def bar(i):
    if i == 1:
        raise KeyError(1)
    if i == 2:
        raise ValueError(2)
def bad():
    e = None
    try:
        bar(int(sys.argv[1]))
    except KeyError as e:
        print('key error')
    except ValueError as e:
        print('value error')
    print(e)
bad()
```

The above lines of code will run perfectly on version 2 of Python. The output will be as follows:

```
$ python foo.py 1
key error
1
$ python foo.py 2
value error
2
```

What do you think it will happen when the program runs on Version 3.x?

```
$ python3 foo.py 1
key error
Traceback (most recent call last):
  File "foo.py", line 19, in <module>
    bad()
  File "foo.py", line 17, in bad
    print(e)
```

UnboundLocalError: local variable 'e' referenced before assignment

What do you think has happened in the above example? The issue is that in the later versions of Python, the error exception cannot be accessed beyond this scope. The best way to avoid this issue is to use the right references when you create an exception object outside the scope of the exception block. The next example can be used on all versions of Python after version 2.

```
import sys
def bar(i):
    if i == 1:
        raise KeyError(1)
    if i == 2:
        raise ValueError(2)
def good():
    exception = None
    try:
        bar(int(sys.argv[1]))
    except KeyError as e:
        exception = e
        print('key error')
    except ValueError as e:
        exception = e
        print('value error')
    print(exception)
good()
```

When you run the above code, you receive the following output:

```
$ python3 foo.py 1
key error
1
$ python3 foo.py 2
```

```
value error
2
```

Incorrect Use of Del Method

Let us assume you have a file named mod.py. This file contains the following information:

```
import foo

class Bar(object):
    ...
  def __del__(self):
    foo.cleanup(self.myhandle)
```

Now, using another file called another_mod.py, you can call the earlier program:

```
import mod
mybar = mod.Bar()
```

If you do this, you are going to receive an exception. Why do you think this happens? The compiler or interpreter will automatically shut down. The module will then set the value of all global variables to 'None.' It is for this reason that we invoke the del method in the example above. The interpreter has set the function value of foo() to None.

An easy way to overcome this is to use the atexit.register() method instead of the del method. When you do this, the registered handlers will be removed when the compiler finishes executing the program. With this understanding, you can fix the script using the lines of code below:

```
import foo
import atexit
def cleanup(handle):
  foo.cleanup(handle)
class Bar(object):
  def __init__(self):
    ...
```

```
atexit.register(cleanup, self.myhandle)
```

Using the above lines of code, you can call the function easily without any errors. It is up to you to determine if you want to bind the code or module with objects.

Chapter Twelve

Solutions

Concatenate two strings

```
# Python Program - Concatenate String

print("Enter 'x' for exit.");
string1 = input("Please enter the first string: ");
if string1 == 'My name is Dobby':
   exit();
else:
   string2 = input("Please enter the second string: ");
   string3 = string1 + string2;
   print("\nString after concatenation =",string3);
   print("String 1 =",string1);
   print("String 2 =",string2);
   print("String 3 =",string3);
```

Sum of Two Numbers

```
# Program to add two numbers


number1 = input(" Please Enter the First Number: ")
number2 = input(" Please Enter the second number: ")


# Using the arithmetic operator to add two numbers
sum = float(number1) + float(number2)
print('The sum of {0} and {1} is {2}'.format(number1, number2,
sum))
```

Even and Odd Numbers

```
# Program to check whether a number is even or odd
# When a number is divided by 2 and the remainder is 0, the
number is even.
```

```
# If remainder is 1, the number is odd.

num = int(input("Enter a number: "))
if (num % 2) == 0:
  print("{0} is Even".format(num))
else:
  print("{0} is Odd".format(num))
```

Fibonacci Series

```
# Program to display the Fibonacci sequence up to n-th term where
n is provided by the user

# change this value for a different result
nterms = 10

# uncomment to take input from the user
#nterms = int(input("How many terms? "))

# first two terms
n1 = 0
n2 = 1
count = 0

# check if the number of terms is valid
if nterms <= 0:
  print("Please enter a positive integer")
elif nterms == 1:
  print("Fibonacci sequence upto",nterms,":")
  print(n1)
else:
  print("Fibonacci sequence upto",nterms,":")
  while count < nterms:
     print(n1,end=' , ')
     nth = n1 + n2
     # update values
     n1 = n2
     n2 = nth
     count += 1
```

Palindrome

```
# Program to check if a string is a palindrome or not
# change this value for a different output

my_str = 'I need to save Harry Potter'

# make it suitable for caseless comparison
my_str = my_str.casefold()

# reverse the string
rev_str = reversed(my_str)

# check if the string is equal to its reverse
if list(my_str) == list(rev_str):
  print("It is palindrome")
else:
  print("It is not palindrome")
```

Access Elements in a List

```
my_list = ['p','r','o','b','e']
# Output: p
print(my_list[0])

# Output: o
print(my_list[2])

# Output: e
print(my_list[4])

# Error! Only integer can be used for indexing
# my_list[4.0]

# Nested List
n_list = ["Happy", [2,0,1,5]]

# Nested indexing

# Output: a
print(n_list[0][1])

# Output: 5
```

```
print(n_list[1][3])
```

Slice a List

```
my_list = ['p','r','o','g','r','a','m','i','z']
# elements 3rd to 5th
print(my_list[2:5])

# elements beginning to 4th
print(my_list[:-5])

# elements 6th to end
print(my_list[5:])

# elements beginning to end
print(my_list[:])
```

Delete Elements in a List

```
>>> my_list = ['p','r','o','b','l','e','m']
>>> my_list[2:3] = []
>>> my_list
['p', 'r', 'b', 'l', 'e', 'm']
>>> my_list[2:5] = []
>>> my_list
['p', 'r', 'm']
```

Access Elements in a Tuple

```
my_tuple = ('p','e','r','m','i','t')

# Output: 'p'
print(my_tuple[0])

# Output: 't'
print(my_tuple[5])

# index must be in range
# If you uncomment line 14,
# you will get an error.
# IndexError: list index out of range
```

```
#print(my_tuple[6])

# index must be an integer
# If you uncomment line 21,
# you will get an error.
# TypeError: list indices must be integers, not float

#my_tuple[2.0]

# nested tuple
n_tuple = ("mouse", [8, 4, 6], (1, 2, 3))

# nested index
# Output: 's'
print(n_tuple[0][3])

# nested index
# Output: 4
print(n_tuple[1][1])
```

Change a Tuple

```
my_tuple = (4, 2, 3, [6, 5])

# we cannot change an element
# If you uncomment line 8
# you will get an error:
# TypeError: 'tuple' object does not support item assignment

#my_tuple[1] = 9

# but item of mutable element can be changed
# Output: (4, 2, 3, [9, 5])
my_tuple[3][0] = 9
print(my_tuple)

# tuples can be reassigned
# Output: ('p', 'r', 'o', 'g', 'r', 'a', 'm', 'i', 'z')
my_tuple = ('p','r','o','g','r','a','m','i','z')
print(my_tuple)
```

Create a String

```
# all of the following are equivalent
my_string = 'Hello'
print(my_string)

my_string = "Hello"
print(my_string)

my_string = '''Hello'''
print(my_string)

# triple quotes string can extend multiple lines
my_string = """Hello, welcome to
        the world of Python"""
print(my_string)
```

Conclusion

Thank you for purchasing the book. Python is a flexible and powerful language that comes with its own paradigms and mechanisms. Using this language, you can improve productivity and automate tasks, as we have discussed in the book. As with any new language or software tool, you cannot automate or improve processes if you have a limited understanding of the language. Therefore, you need to understand the basics of Python, and this has been covered in the book.

This book has all the basic information you need about Python and how you can develop different programs using Python. The book also has numerous examples and exercises you can use to improve your understanding of the language. Use the tips and information mentioned in the book to improve your understanding.

Do not worry about making mistakes when you work with Python, but find a way to overcome the mistake. The book also comes with different mistakes people make when using Python, so use this information to avoid making mistakes in your code. Python can be used to automate numerous tasks, so understand the basics well before you dive into developing large programs.

You can only master the use of the language if you practice. Use the exercises and examples in the book to master the concepts before you use the functions and methods to develop new programs or applications. I hope the information in the book has helped you master the concepts.

References

5. Data Structures — Python 3.8.3 documentation. (n.d.). Docs.python.org. https://docs.python.org/3/tutorial/datastructures.html

5 Tasks To Automate With Python. (n.d.). KDnuggets. https://www.kdnuggets.com/2021/06/5-tasks-automate-python.html

About — Python Notes (0.14.0). (n.d.). Thomas-Cokelaer.info. http://thomas-cokelaer.info/tutorials/python/

Barnett, N. (n.d.). Web Scraping Using Python Selenium. Toptal Engineering Blog. https://www.toptal.com/python/web-scraping-with-python

Buturla, E. (2020, June 5). How To Become A Task Automation Hero Using Python [With Examples]. Monterail. https://www.monterail.com/blog/python-task-automation-examples#:~:text=Reading%20and%20writing%20files%20is,an%20approach%20I%20highly%20recommend.

Chikilian, M. (n.d.). Buggy Python Code: The 10 Most Common Mistakes That Python Developers Make. Toptal Engineering Blog. https://www.toptal.com/python/top-10-mistakes-that-python-programmers-make

Dunn, N. (n.d.). Welcome to Python.org. Python.org. https://www.python.org/success-stories/using-python-to-automate-tedious-tasks/

Hiremath, O. S. (2018, November 14). A Beginner's Guide to learn web scraping with python! Edureka; Edureka. https://www.edureka.co/blog/web-scraping-with-python/

Implementing Web Scraping in Python with BeautifulSoup. (2016, November 18). GeeksforGeeks.

https://www.geeksforgeeks.org/implementing-web-scraping-python-beautiful-soup/

Jaiswal, S. (2017, December 8). Python Data Structures Tutorial. Datacamp. https://www.datacamp.com/community/tutorials/data-structures-python

Learn Python (Programming Tutorial for Beginners). (2010). Programiz.com. https://www.programiz.com/python-programming

Loops - Learn Python - Free Interactive Python Tutorial. (n.d.). Www.learnpython.org. https://www.learnpython.org/en/Loops

Masango, S. (2018, July 26). Web Scraping using Python. Datacamp. https://www.datacamp.com/community/tutorials/web-scraping-using-python

Miller, B., & Ranum, D. (n.d.). Problem Solving with Algorithms and Data Structures using Python — Problem Solving with Algorithms and Data Structures. Runestone.academy. https://runestone.academy/runestone/books/published/pythonds/index.html

Power, V. (2020, June 25). Top 10 Tasks to Automate with Python. ActiveState. https://www.activestate.com/blog/top-10-tasks-to-automate-with-python/

Python For Loops. (2019). W3schools.com. https://www.w3schools.com/python/python_for_loops.asp

PYTHON
PYTHON FOR DATA SCIENCE AND MACHINE LEARNING

Andy Vickler

Introduction

Thank you for purchasing this guide about data science and machine learning with Python. One of the first questions people ask is, what is data science? This is not a particularly easy question to answer because the term is widespread these days, used just about everywhere. Some say it is nothing more than an unnecessary label, given that science is all about data anyway, while others say it is just a buzzword.

However, many fail to see or choose to ignore because data science is quite possibly the ONLY label that could be given to the cross-discipline skill set that is fast becoming one of the most important in applications. It isn't possible to place data science firmly into one discipline; that much is definitely true. It comprises three areas, all distinct and all overlapping:

- **Statistician** – these people are skilled in modeling datasets and summarizing them. This is an even more important skill set these days, given the ever-increasing size of today's datasets.
- **Computer Scientist –** these people design algorithms and use them to store data, and process it, and visualize it efficiently.
- **Domain Expertise -** is akin to being classically trained in any subject, i.e. having expert knowledge. This is important to ensure the right questions are asked and the answers placed in the right context.

So, going by this, it wouldn't be remiss of me to encourage you to see data science as a set of skills you can learn and apply in your own expertise area, rather than being something new you have to learn from scratch. It doesn't matter if you are examining microspore images looking for microorganisms, forecasting

returns on stocks, optimizing online ads to get more clicks or any other field where you work with data. This book will give you the fundamentals you need to learn to ask the right questions in your chosen expertise area.

Who This Book Is For

This book is aimed at those with experience of programming in Python, designed to help them further their skills and learn how to use their chosen programming language to go much further than before. It is not aimed at those with no experience of Python or programming, and I will not be giving you a primer on the language.

The book assumes that you already have experience defining and using functions, calling object methods, assigning variables, controlling program flow, and all the other basic Python tasks. It is designed to introduce you to the data science libraries in Python, such as NumPy, Pandas, Seaborn, Scikit-Learn, and so on, showing you how to use them to store and manipulate data and gain valuable insights you can use in your work.

Why Use Python?

Over the last few decades, Python has emerged as the best and most popular tool for analyzing and visualizing data, among other scientific tasks. When Python was first designed, it certainly wasn't with data science in mind. Instead, its use in these areas has come from the addition of third-party packages, including:

- **NumPy** – for manipulating array-based data of the same kind
- **Pandas** – for manipulating array-based data of different kinds
- **SciPy** – for common tasks in scientific computing
- **Matplotlib** – for high quality, accurate visualizations
- **Scikit-Learn** – for machine learning

And so many more.

Python 2 vs. 3

The code used in this book follows the Python 3 syntax. Many enhancements are not backward compatible with Python 2, which is why you are always urged to install the latest Python version.

Despite Python 3 being released around 13 years ago, people have been slow to adopt it, especially in web development and scientific communities. This is mostly down to the third-party packages not being made compatible right from the start. However, from 2014, those packages started to become available in stable releases, and more people have begun to adopt Python 3.

What's in This Book?

This book has been separated into five sections, each focusing on a specific area:

- **Part One** – introduces you to data science and machine learning
- **Part Two** – focuses on NumPy and how to use it for storing and manipulating data arrays
- **Part Three** – focuses on Pandas and how to use it for storing and manipulating columnar or labeled data arrays
- **Part Four** – focuses on using Matplotlib and Seaborn for data visualization
- **Part Five** – takes up approximately half of the book and focuses on machine learning, including a discussion on algorithms.

There is more to data science than all this, but these are the main areas you should focus on as you start your data science and machine learning journey.

Installation

Given that you should have experience in Python, I am assuming you already have Python on your machine. However, if you don't,

there are several ways to install it, but Anaconda is the most common way. There are two versions of this:

- **Miniconda** – provides you with the Python interpreter and conda, a command-line tool
- **Anaconda** – provides conda and Python, along with many other packages aimed at data science. This is a much larger installation, so make sure you have plenty of free space.

It's also worth noting that the packages discussed in this book can be installed on top of Miniconda, so it's down to you which version you choose to install. If you choose Miniconda, you can use the following command to install all the packages you need:

```
[~]$ conda install numpy pandas scikit-learn matplotlib seaborn jupyter
```

All packages and tools can easily be installed with the following:

```
conda install packagename
```

ensuring you type the correct name in place of packagename.

Let's not waste any more time and dive into our introduction to data science and machine learning.

Part One

An Introduction to Data Science and Machine Learning

The definition of data science tells us that it is a research area used to derive valuable insight from data, and it does this using a theoretical method. In short, data science is a combination of simulation, technology, and market management.

Machine learning is a bit different. This is all about learning the data science techniques the computers use to learn from the data we give them. We use these technologies to gain outcomes without needing to program any specific laws.

Machine learning and data science are trending high these days and, although many people use the terms interchangeably, they are different. So, let's break this down and learn what we need to know to continue with the rest of the book.

What Is Data Science?

The real meaning of data science is found in the deep analysis of huge amounts of data contained in a company's archive. This involves working out where that data originated from, whether the data is of sufficient quality, and whether it can be used to move the business on in the future.

Often, an organization stores its data in one of two formats – unstructured or organized. Examining the data gives us useful insights into the consumer and industry dynamics, which can be used to provide an advantage to the company over their rivals—this is done by looking for patterns in the data.

Data scientists know how to turn raw data into something a business can use. They know how to spot algorithmic coding and process the data, with knowledge of statistics and artificial learning. Some of the top companies in the world use data analytics, including Netflix, Amazon, airlines, fraud prevention departments, healthcare, etc.

Data Science Careers

Most businesses use data analysis effectively to expand, and data scientists are in the highest demand across many industries. If you are considering getting involved in data science, these are some of the best careers you can train for:

- **Data Scientist –** investigates trends in the data to see what effect they will have on a business. Their primary job is to determine the significance of the data and clarify it so that everyone can understand it.
- **Data Analyst –** analyzes data to see what industry trends exist. They help develop simplistic views of where a business stands in its industry.
- **Data Engineer –** often called the backbone of a business, data engineers create databases to store data, designing them for the best use, and manage them. Their job entails data pipeline design, ensuring the data flows adequately, and getting to where it needs to be.
- **Business Intelligence Analyst -** sometimes called a market intelligence consultant-study data to improve income and productivity for a business. The job is less theoretical and more scientific and requires a deep understanding of common machines.

How Important Is Data Science?

In today's digital world, a world full of data, it is one of the most important jobs. Here are some of the reasons why.

- First, it gives businesses a better way of identifying who their customers are. Given that customers keep a business's wheels turning, they determine whether that business fails or succeeds. With data science, businesses can now communicate with customers in the right way, in different ways for different customers.
- Second, data science makes it possible for a product to tell its story in an entertaining and compelling way, one of the main reasons why data science is so popular. Businesses and brands can use the information gained from data science to communicate what they want their customers to know, ensuring much stronger relationships between them.
- Next, the results gained from data science can be used in just about every field, from schooling to banking, healthcare to tourism, etc. Data science allows a business to see problems and respond to them accordingly.
- A business can use data analytics to better communicate with its customers/consumers and provide the business with a better view of how its products are used.
- Data science is quickly gaining traction in just about every market and is now a critical part of how products are developed. That has led to a rise in the need for data scientists to help manage data and find the answers to some of the more complex problems a business faces.

Data Science Limitations

Data science may be one of the most lucrative careers in the world right now, but it isn't perfect and has its fair share of drawbacks. It wouldn't be right to brag about what data science can do without pointing out its limitations:

- Data science is incredibly difficult to master. That's because it isn't just one discipline involving a combination of information science, statistics, and mathematics, to name a few. It isn't easy to master every discipline involved to a competent level.
- Data science also requires a high level of domain awareness. In fact, it's fair to say that it relies on it. If you have no knowledge of computer science and statistics, you will find it hard to solve any data science problem.
- Data privacy is a big issue. Data makes the world go around, and data scientists create data-driven solutions for businesses. However, there is always a chance that the data they use could infringe privacy. A business will always have access to sensitive data and, if data protection is compromised in any way, it can lead to data breaches.

What Is Machine Learning?

Machine learning is a subset of data science, enabling machines to learn things without the need for a human to program them. Machine learning analyzes data using algorithms and prepares possible predictions without human intervention. It involves the computer learning a series of inputs in the form of details, observations, or commands, and it is used extensively by companies such as Google and Facebook.

Machine Learning Careers

There are several directions you can take once you have mastered the relevant skills. These are three of those directions:

- **Machine Learning Engineer** – this is one of the most prized careers in the data science world. Engineers typically use machine learning algorithms to ensure machine learning applications and systems are as effective as they can be. A machine learning engineer is responsible for molding self-learning applications, improving their

effectiveness and efficiency by using the test results to fine-tune them and run statistical analyses. Python is one of the most used programming languages for performing machine learning experiments.

- **NLP Scientist** – these are concerned with Natural Language Processing, and their job is to ensure machines can interpret natural languages. They design and engineer software and computers to learn human speech habits and convert the spoken word into different languages. Their aim is to get computers to understand human languages the same way we do, and two of the best examples are apps called DuoLingo and Grammarly.
- **Developer/Engineer of Software** – these develop intelligent computer programs, and they specialize in machine learning and artificial intelligence. Their main aim is to create algorithms that work effectively and implement them in the right way. They design complex functions, plan flowcharts, graphs, layouts, product documentation, tables, and other visual aids. They are also responsible for composing code and evaluating it, creating tech specs, updating programs and managing them, and more.

How Important Is Machine Learning?

Machine learning is an ever-changing world and, the faster it evolves, the higher its significance is, and the more demand grows. One of the main explanations as to why data scientists can't live without machine learning is this – "high-value forecasts that direct smart decisions and behavior in real-time, without interference from humans."

It is fast becoming popular to interpret huge amounts of data and helping to automate the tasks that data scientists do daily. It's fair to say that machine learning has changed the way we extract data and visualize it. Given how much businesses rely on data these

days, data-driven decisions help determine if a company is likely to succeed or fall behind its competitors.

Machine Learning Limitations

Like data science, machine learning also has its limitations, and these are some of the biggest ones:

- Algorithms need to store huge amounts of training data. Rather than being programmed, an AI program is educated. This means they require vast amounts of data to learn how to do something and execute it at a human level. In many cases, these huge data sets are not easy to generate for specific uses, despite the rapid level at which data is being produced.

  Neural networks are taught to identify things, for example, images, and they do this by being trained on massive amounts of labeled data in a process known as supervised learning. No matter how large or small, any change to the assignment requires a new collection of data, which also needs to be prepared. It's fair to say that a neural network cannot truly operate at a human intelligence level because of the brute force needed to get it there – that will change in the future.

- Machine learning needs too much time to mark the training data. AI uses supervised learning on a series of deep neural nets. It is critical to label the data in the processing stage, and this is done using predefined goal attributes taken from historical data. Marking the data is where the data is cleaned and sorted to allow the neural nets to work on it.

  There's no doubt that vast amounts of labeled information are needed for deep learning and, while this isn't particularly hard to grasp, it isn't the easiest job to do. If

unlabeled results are used, the program cannot learn to get smarter.

- AI algorithms don't always work well together. Although there have been some breakthroughs in recent times, an AI model may still only generalize a situation if it is asked to do something it didn't do in training. AI models cannot always move data between groups of conditions, implying that whatever is accomplished by a paradigm with a specific use case can only be relevant to that case. Consequently, businesses must use extra resources to keep models trained and train new ones, despite the use cases being the same in many cases.

Data Science vs. Machine Learning

Data scientists need to understand data analysis in-depth, besides having excellent programming abilities. Based on the business hiring the data scientist, there is a range of expertise, and the skills can be split down into two different categories:

Technical Skills

You need to specialize in:

- Algebra
- Computer Engineering
- Statistics

You must also be competent in:

- Computer programming
- Analytical tools, such as R, Spark, Hadoop, and SAS
- Working with unstructured data obtained from different networks

Non-Technical Skills

Most of a data scientist's abilities are non-technical and include:

- A good sense of business
- The ability to interact
- Data intuition

Machine learning experts also need command over certain skills, including:

- **Probability and statistics –** theory experience is closely linked to how you comprehend algorithms, such as Naïve Bayes. Hidden Markov, Gaussian Mixture, and many more. Being experienced in numbers and chance makes these easier to grasp.
- **Evaluating and modeling data –** frequently evaluating model efficacy is a critical part of maintaining measurement reliability in machine learning. Classification, regression, and other methods are used to evaluate consistency and error rates in models, along with assessment plans, and knowledge of these is vital.
- **Machine learning algorithms –** there are tons of machine learning algorithms, and you need to have knowledge of how they operate to know which one to use for each scenario. You will need knowledge of quadratic programming, gradient descent, convex optimization, partial differential equations, and other similar topics.
- **Programming languages –** you will need experience in programming in one or more languages – Python, R, Java, C++, etc.
- **Signal processing techniques –** feature extraction is one of the most critical factors in machine learning. You will need to know some specialized processing algorithms, such as shearlets, bandlets, curvelets, and contourlets.

Data science is a cross-discipline sector that uses huge amounts of data to gain insights. At the same time, machine learning is an exciting subset of data science, used to encourage machines to learn by themselves from the data provided.

Both have a huge amount of use cases, but they do have limits. Data science may be strong, but, like anything, it is only effective if the proper training and best-quality data are used.

Let's move on and start looking into using Python for data science. Next, we introduce you to NumPy.

Part Two

Introducing NumPy

NumPy is one of the most basic Python libraries, and, over time, you will come to rely on it in all kinds of data science tasks, be they simple math to classifying images. NumPy can handle data sets of all sizes efficiently, even the largest ones, and having a solid grasp on how it works is essential to your success in data science.

First, we will look at what NumPy is and why you should use it over other methods, such as Python lists. Then we will look at some of its operations to understand exactly how to use it – this will include plenty of code examples for you to study.

What Is NumPy Library?

NumPy is a shortened version of **Num** erical **Py** thon, and it is, by far, the most scientific library Python has. It supports multidimensional array objects, along with all the tools needed to work with those arrays. Many other popular libraries, like Pandas, Scikit-Learn, and Matplotlib, are all built on NumPy.

So, what is an array? If you know your basic Python, you know that an array is a collection of values or elements with one or more dimensions. A one-dimensional array is a Vector, while a two-dimensional array is a Matrix.

NumPy arrays are known as N-dimensional arrays, otherwise called ndarrays, and the elements they store are the same type and same size. They are high-performance, efficient at data operations, and an effective form of storage as the arrays grow larger.

When you install Anaconda, you get NumPy with it, but you can install it on your machine separately by typing the following command into the terminal:

```
pip install numpy
```

Once you have done that, the library must be imported, using this command:

```
import numpy as np
```

NOTE

np is the common abbreviation for NumPy

Python Lists vs. NumPy Arrays

Those used to using Python will probably be wondering why we would use the NumPy arrays when we have the perfectly good Python Lists at our disposal. These lists already act as arrays to store various types of elements, after all, so why change things?

Well, the answer to this lies in how objects are stored in memory by Python.

Python objects are pointers to memory locations where object details are stored. These details include the value and the number of bytes. This additional information is part of the reason why Python is classed as a dynamically typed language, but it also comes with an additional cost. That cost becomes clear when large object collections, such as arrays are stored.

A Python list is an array of pointers. Each pointer points to a specific location containing information that relates to the element. As far as computation and memory go, this adds a significant overhead cost. To make matters worse, when all the stored objects are the same type, most of the additional information becomes redundant, meaning the extra cost for no reason.

NumPy arrays overcome this issue. These arrays only store objects of the same type, otherwise known as homogenous objects. Doing it this way makes storing the array and manipulating it far more efficient, and we can see this difference clearly when there are vast amounts of elements in the array, say millions of them. We can also carry out element-wise operations on NumPy arrays, something we cannot do with a Python list.

This is why we prefer to use these arrays over lists when we want mathematical operations performed on large quantities of data.

How to Create a NumPy Array

Basic ndarray

NumPy arrays are simple to create, regardless of which one it is and the complexity of the problem it is solving. We'll start with the most basic NumPy array, the ndarray.

All you need is the following np.array() method, ensuring you pass the array values as a list:

```
np.array([1,2,3,4])
```

The output is:

```
array([1, 2, 3, 4])
```

Here, we have included integer values and the dtype argument is used to specify the data type:

```
np.array([1,2,3,4],dtype=np.float32)
```

The output is:

```
array([1., 2., 3., 4.], dtype=float32)
```

Because you can only include homogenous data types in a NumPy array if the data types don't match, the values are upcast:

```
np.array([1,2.0,3,4])
```

The output is:

```
array([1., 2., 3., 4.])
```

In this example, the integer values are upcast to float values.

You can also create multidimensional arrays:

```
np.array([[1,2,3,4],[5,6,7,8]])
array([[1, 2, 3, 4],
    [5, 6, 7, 8]])
```

This creates a two-dimensional array containing values.

NOTE

The above example is a 2 x 4 matrix. A matrix is a rectangular array containing numbers. Its shape is N x M – N indicates how many rows there are, and M indicates how many columns there are.

An Array of Zeroes

You can also create arrays containing nothing but zeros. This is done with the np.zeros() method, ensuring you pass the shape of the array you want:

```
np.zeros(5)
array([0., 0., 0., 0., 0.])
```

Above, we have created a one-dimensional array, while the one below is a two-dimensional array:

```
np.zeros((2,3))
array([[0., 0., 0.],
    [0., 0., 0.]])
```

An Array of Ones

The np.ones() method is used to create an array containing 1s:

```
np.ones(5,dtype=np.int32)
array([1, 1, 1, 1, 1])
```

Using Random Numbers in a ndarray

The np.random.rand() method is commonly used to create ndarray's with a given shape containing random values from [0,1):

```
# random
np.random.rand(2,3)
array([[0.95580785, 0.98378873, 0.65133872],
    [0.38330437, 0.16033608, 0.13826526]])
```

Your Choice of Array

You can even create arrays with any value you want with the np.full() method, ensuring the shape of the array you want is passed in along with the desired value:

```
np.full((2,2),7)
array([[7, 7],
    [7, 7]])
```

NumPy IMatrix

The np.eye() method is another good one that returns 1s on the diagonal and 0s everywhere else. More formally known as an Identity Matrix, it is a square matric with an N x N shape, meaning there is the same number of rows as columns. Below is a 3 x 3 IMatrix:

```
# identity matrix
np.eye(3)
array([[1., 0., 0.],
    [0., 1., 0.],
    [0., 0., 1.]])
```

However, you can also change the diagonal where the values are 1s. It could be above the primary diagonal:

```
# not an identity matrix
np.eye(3,k=1)
array([[0., 1., 0.],
    [0., 0., 1.],
    [0., 0., 0.]])
```

Or below it:

```
np.eye(3,k=-2)
```

```
array([[0., 0., 0.],
       [0., 0., 0.],
       [1., 0., 0.]])
```

NOTE

A matrix can only be an IMatrix when the 1s appear on the main diagonal, no other.

Evenly-Spaced ndarrays

The np.arange() method provides evenly-spaced number arrays:

```
np.arange(5)
array([0, 1, 2, 3, 4])
```

We can explicitly define the start and end and the interval step size by passing in an argument for each value.

NOTE

We define the interval as [start, end) and the final number is not added into the array:

```
np.arange(2,10,2)
array([2, 4, 6, 8])
```

Because we defined the step size as 2, we got the alternate elements as a result. The final element was 10, and this was not included.

A similar function is called np.linspace(), which takes an argument in the form of the number of samples we want from the interval.

NOTE

This time, the last number will be included in the result.

```
np.linspace(0,1,5)
array([0.  , 0.25, 0.5 , 0.75, 1.  ])
```

That is how to create a variety of arrays with NumPy, but there is something else just as important – the array's shape.

Shaping and Reshaping a NumPy array

When your ndarray has been created, you need to check three things:

- How many axes the ndarray has
- The shape of the array
- The size of the array

NumPy Array Dimensions

Determining how many axes or dimensions the ndarray has is easily done using an attribute called ndims:

```
# number of axis
a = np.array([[5,10,15],[20,25,20]])
print('Array :','\n',a)
print('Dimensions :','\n',a.ndim)
Array :
[[ 5 10 15]
[20 25 20]]
Dimensions :
2
```

Here, we have a two-dimensional array with two rows and three columns.

NumPy Array Shape

The array shape is an array attribute showing the number of rows with elements along each of the dimensions. The shape can be further indexed so the result shows the value on each dimension:

```
a = np.array([[1,2,3],[4,5,6]])
print('Array :','\n',a)
print('Shape :','\n',a.shape)
print('Rows = ',a.shape[0])
print('Columns = ',a.shape[1])
Array :
[[1 2 3]
[4 5 6]]
Shape :
(2, 3)
```

```
Rows =  2
Columns =  3
```

Array Size

The size attribute tells you the number of values in your array by multiplying the number of rows by columns:

```
# size of array
a = np.array([[5,10,15],[20,25,20]])
print('Size of array :',a.size)
print('Manual determination of size of array
:',a.shape[0]*a.shape[1])
Size of array : 6
```

Manual determination of size of array : 6

Reshaping an Array

You can use the np.reshape() method to reshape your ndarray without changing any of the data in it:

```
# reshape
a = np.array([3,6,9,12])
np.reshape(a,(2,2))
array([[ 3,  6],
     [ 9, 12]])
```

We reshaped this from a one-dimensional to a two-dimensional array.

When you reshape your array and are not sure of any of the axes shapes, input -1. When NumPy sees -1, it will calculate the shape automatically:

```
a = np.array([3,6,9,12,18,24])
print('Three rows :','\n',np.reshape(a,(3,-1)))
print('Three columns :','\n',np.reshape(a,(-1,3)))
Three rows :
[[ 3  6]
[ 9 12]
[18 24]]
Three columns :
```

```
[[ 3  6  9]
[12 18 24]]
```

Flattening Arrays

There may be times when you want to collapse a multidimensional array into a single-dimensional array. There are two methods you can use for this – flatten() or ravel():

```
a = np.ones((2,2))
b = a.flatten()
c = a.ravel()
print('Original shape :', a.shape)
print('Array :','\n', a)
print('Shape after flatten :',b.shape)
print('Array :','\n', b)
print('Shape after ravel :',c.shape)
print('Array :','\n', c)
Original shape : (2, 2)
Array :
[[1. 1.]
[1. 1.]]
Shape after flatten : (4,)
Array :
[1. 1. 1. 1.]
Shape after ravel : (4,)
Array :
[1. 1. 1. 1.]
```

However, one critical difference exists between the two methods – flatten() will return a copy of the original while ravel() only returns a reference. If there are any changes to the array revel() returns, the original array will reflect them, but this doesn't happen with the flatten() method.

```
b[0] = 0
print(a)
[[1. 1.]
[1. 1.]]
```

The original array did not show the changes.

```
c[0] = 0
print(a)
[[0. 1.]
[1. 1.]]
```

But this one did show the changes.

A deep copy of the ndarray is created by flatten(), while a shallow copy is created by ravel().

A new ndarray is created in a deep copy, stored in memory, with the object flatten() returns pointing to that location in memory. That is why changes will not reflect in the original.

A reference to the original location is returned by a shallow copy which means the object ravel() returns points to the memory location where the original object is stored. That means changes will be reflected in the original.

Transposing a ndarray

The transpose() method is a reshaping method that takes the input array, swapping the row values with the columns and vice versa:

```
a = np.array([[1,2,3],
[4,5,6]])
b = np.transpose(a)
print('Original','\n','Shape',a.shape,'\n',a)
print('Expand along columns:','\n','Shape',b.shape,'\n',b)
Original
Shape (2, 3)
[[1 2 3]
[4 5 6]]
```

Expand along columns:

```
Shape (3, 2)
[[1 4]
[2 5]
[3 6]]
```

When you transpose a 2 x 3 array, the result is a 3 x 2 array.

Expand and Squeeze NumPy Arrays

Expanding a ndarray involves adding new axes. This is done with the method called expand_dims(), passing the array and the axis you want to expand along:

```
# expand dimensions
a = np.array([1,2,3])
b = np.expand_dims(a,axis=0)
c = np.expand_dims(a,axis=1)
print('Original:','\n','Shape',a.shape,'\n',a)
print('Expand along columns:','\n','Shape',b.shape,'\n',b)
print('Expand along rows:','\n','Shape',c.shape,'\n',c)
Original:
Shape (3,)
[1 2 3]
```

Expand along columns:

```
Shape (1, 3)
[[1 2 3]]
```

Expand along rows:

```
Shape (3, 1)
[[1]
[2]
[3]]
```

However, if you want the axis reduced, you would use the squeeze() method. This will remove an axis with one entry so, where you have a matrix of 2 x 2 x 1, the third dimension is removed.

```
# squeeze
a = np.array([[[1,2,3],
[4,5,6]]])
b = np.squeeze(a, axis=0)
print('Original','\n','Shape',a.shape,'\n',a)
print('Squeeze array:','\n','Shape',b.shape,'\n',b)
Original
```

```
Shape (1, 2, 3)
[[[1 2 3]
  [4 5 6]]]
```

Squeeze array:

```
Shape (2, 3)
[[1 2 3]
[4 5 6]]
```

Where you have a 2 x 2 matrix and you try to use squeeze() you would get an error:

```
# squeeze
a = np.array([[1,2,3],
[4,5,6]])
b = np.squeeze(a, axis=0)
print('Original','\n','Shape',a.shape,'\n',a)
print('Squeeze array:','\n','Shape',b.shape,'\n',b)cv
```

The error message will point to line 3 being the issue, providing a message saying:

ValueError: cannot select an axis to squeeze out which has size not equal to one

Index and Slice a NumPy Array

We know how to create NumPy arrays and how to change their shape and size. Now, we will look at using indexing and slicing to extract values.

Slicing a One-Dimensional Array

When you slice an array, you retrieve elements from a specified "slice" of the array. You must provide the start and end points like this [start: end], but you can also take things a little further and provide a step size. Let's say you alternative element in the array printed. In that case, your step size would be defined as 2, which means you want the element 2 steps in from the current index. This would look something like [start: end:step-size]:

```
a = np.array([1,2,3,4,5,6])
print(a[1:5:2])
[2 4]
```

Note the final element wasn't considered – when you slice an array, only the start index is included, not the end index.

You can get around this by writing the next higher value to your final value:

```
a = np.array([1,2,3,4,5,6])
print(a[1:6:2])
[2 4 6]
```

If the start or end index isn't specified, it will default to 0 for the start and the array size for the end index, with a default step size of 1:

```
a = np.array([1,2,3,4,5,6])
print(a[:6:2])
print(a[1::2])
print(a[1:6:])
[1 3 5]
[2 4 6]
[2 3 4 5 6]
```

Slicing a Two-Dimensional Array

Two-dimensional arrays have both columns and rows so slicing them is not particularly easy. However, once you understand how it's done, you will be able to slice any array you want.

Before we slice a two-dimensional array, we need to understand how to retrieve elements from it:

```
a = np.array([[1,2,3],
[4,5,6]])
print(a[0,0])
print(a[1,2])
print(a[1,0])
```

```
1
6
4
```

Here, we identified the relevant element by supplying the row and column values. We only need to provide the column value in a one-dimensional array because the array only has a single row.

Slicing a two-dimensional array requires that a slice for both the column and the row are mentioned:

```
a = np.array([[1,2,3],[4,5,6]])
# print first row values
print('First row values :','\n',a[0:1,:])
# with step-size for columns
print('Alternate values from first row:','\n',a[0:1,::2])
#
print('Second column values :','\n',a[:,1::2])
print('Arbitrary values :','\n',a[0:1,1:3])
First row values :
[[1 2 3]]
Alternate values from first row:
[[1 3]]
Second column values :
[[2]
[5]]
Arbitrary values :
[[2 3]]
```

Slicing a Three-Dimensional Array

We haven't looked at three-dimensional arrays yet, so let's see what one looks like:

```
a = np.array([[[1,2],[3,4],[5,6]],# first axis array
[[7,8],[9,10],[11,12]],# second axis array
[[13,14],[15,16],[17,18]]])# third axis array
# 3-D array
print(a)
[[[ 1  2]
  [ 3  4]
  [ 5  6]]
```

```
[[ 7  8]
 [ 9 10]
 [11 12]]

[[13 14]
 [15 16]
 [17 18]]]
```

Three-dimensional arrays don't just have rows and columns. They also have depth axes, which is where a two-dimensional array is stacked behind another one. When a three-dimensional array is sliced, you must specify the two-dimensional array you want to be sliced. This would typically be listed first in the index:

```
# value
print('First array, first row, first column value :','\n',a[0,0,0])
print('First array last column :','\n',a[0,:,1])
print('First two rows for second and third arrays :','\n',a[1:,0:2,0:2])
First array, first row, first column value :
1
First array last column :
[2 4 6]
First two rows for second and third arrays :
[[[ 7  8]
  [ 9 10]]

 [[13 14]
  [15 16]]]
```

If you wanted the values listed as a one-dimensional array, the flatten() method can be used:

```
print('Printing as a single array :','\n',a[1:,0:2,0:2].flatten())
```

Printing as a single array :

```
[ 7  8  9 10 13 14 15 16]
```

Negative Slicing

You can also use negative slicing on your array. This prints the elements from the end, instead of starting at the beginning:

```
a = np.array([[1,2,3,4,5],
[6,7,8,9,10]])
print(a[:,-1])
[ 5 10]
```

Each row's final values were printed, but if we wanted the values extracted from the end, a negative step-by-step would need to be provided. If not, you get an empty list.

```
print(a[:,-1:-3:-1])
[[ 5  4]
[10  9]]
```

That said, slicing logic is the same – you won't get the end index in the output.

You can also use negative slicing if you want the original array reversed:

```
a = np.array([[1,2,3,4,5],
[6,7,8,9,10]])
print('Original array :','\n',a)
print('Reversed array :','\n',a[::-1,::-1])
Original array :
[[ 1  2  3  4  5]
[ 6  7  8  9 10]]
Reversed array :
[[10  9  8  7  6]
[ 5  4  3  2  1]]
```

And a ndarray can be reversed using the flip() method:

```
a = np.array([[1,2,3,4,5],
[6,7,8,9,10]])
print('Original array :','\n',a)
print('Reversed array vertically :','\n',np.flip(a,axis=1))
print('Reversed array horizontally :','\n',np.flip(a,axis=0))
Original array :
[[ 1  2  3  4  5]
```

```
[ 6  7  8  9 10]]
Reversed array vertically :
[[ 5  4  3  2  1]
[10  9  8  7  6]]
Reversed array horizontally :
[[ 6  7  8  9 10]
[ 1  2  3  4  5]]
```

Stack and Concatenate NumPy Arrays

Existing arrays can be combined to create new arrays, and this can be done in two ways:

- Vertically combine arrays along the rows. This is done with the vstack() method and increases the number of rows in the array that gets returned.
- Horizontally combine arrays along the columns. This is done using the hstack() method and increases the number of columns in the array that gets returned.

```
a = np.arange(0,5)
b = np.arange(5,10)
print('Array 1 :','\n',a)
print('Array 2 :','\n',b)
print('Vertical stacking :','\n',np.vstack((a,b)))
print('Horizontal stacking :','\n',np.hstack((a,b)))
Array 1 :
[0 1 2 3 4]
Array 2 :
[5 6 7 8 9]
Vertical stacking :
[[0 1 2 3 4]
[5 6 7 8 9]]
Horizontal stacking :
[0 1 2 3 4 5 6 7 8 9]
```

Worth noting is that the axis along which the array is combined must have the same size. If it doesn't, an error is thrown:

```
a = np.arange(0,5)
```

```
b = np.arange(5,9)
print('Array 1 :','\n',a)
print('Array 2 :','\n',b)
print('Vertical stacking :','\n',np.vstack((a,b)))
print('Horizontal stacking :','\n',np.hstack((a,b)))
```

The error message points to line 5 as where the error is and reads:

ValueError: all the input array dimensions for the concatenation axis must match exactly, but along dimension 1, the array at index 0 has size 5 and the array at index 1 has size 4.

You can also use the dstack() method to combine arrays by combing the elements one index at a time and stacking them on the depth axis.

```
a = [[1,2],[3,4]]
b = [[5,6],[7,8]]
c = np.dstack((a,b))
print('Array 1 :','\n',a)
print('Array 2 :','\n',b)
print('Dstack :','\n',c)
print(c.shape)
Array 1 :
[[1, 2], [3, 4]]
Array 2 :
[[5, 6], [7, 8]]
Dstack :
[[[1 5]
  [2 6]]

 [[3 7]
  [4 8]]]
(2, 2, 2)
```

You can stack arrays to combine old ones into a new one, but you can also join passed arrays on an existing axis using the concatenate() method:

```
a = np.arange(0,5).reshape(1,5)
b = np.arange(5,10).reshape(1,5)
```

```
print('Array 1 :','\n',a)
print('Array 2 :','\n',b)
print('Concatenate along rows :','\n',np.concatenate((a,b),axis=0))
print('Concatenate along columns
:','\n',np.concatenate((a,b),axis=1))
Array 1 :
[[0 1 2 3 4]]
Array 2 :
[[5 6 7 8 9]]
Concatenate along rows :
[[0 1 2 3 4]
[5 6 7 8 9]]
Concatenate along columns :
[[0 1 2 3 4 5 6 7 8 9]]
```

However, this method has one primary drawback – the original array must contain the axis along which you are combining, or else you will get an error.

The append() method can be used to add new elements at the end of a ndarray. This is quite useful if you want new values added to an existing ndarray:

```
# append values to ndarray
a = np.array([[1,2],
          [3,4]])
np.append(a,[[5,6]], axis=0)
array([[1, 2],
     [3, 4],
     [5, 6]])
```

Broadcasting in NumPy Arrays

While ndarrays have many good features, one of the best is undoubtedly broadcasting. It allows mathematical operations between different sized ndarrays or a simple number and ndarray.

In essence, broadcasting stretches smaller ndarrays to match the larger one’s shape.

```
a = np.arange(10,20,2)
b = np.array([[2],[2]])
print('Adding two different size arrays :','\n',a+b)
print('Multiplying a ndarray and a number :',a*2)
Adding two different size arrays :
[[12 14 16 18 20]
[12 14 16 18 20]]
Multiplying a ndarray and a number : [20 24 28 32 36]
```

Think of it as being similar to making a copy of or stretching the number or scalar [2, 2, 2] to match the ndarray shape and then do an elementwise operation on it. However, no actual copies are made – it is just the best way to think of how broadcasting works.

Why is this so useful?

Because multiplying an array containing a scalar value is more efficient than multiplying another array.

NOTE

A pair of ndarrays must be compatible to broadcast together. This compatibility happens when:

- Both ndarrays have identical dimensions
- One ndarray has a dimension of 1, which is then broadcast to meet the larger one's size requirements

Should they not be compatible, an error is thrown:

```
a = np.ones((3,3))
b = np.array([2])
a+b
array([[3., 3., 3.],
       [3., 3., 3.],
       [3., 3., 3.]])
```

In this case, we hypothetically stretched the second array to a shape of 3 x 3 before calculating the result.

NumPy Ufuncs

If you know anything about Python, you know it is dynamically typed. That means that Python doesn't need to know its data type at the time a variable is assigned because it will determine it automatically at runtime. However, while this ensures a cleaner code, it does slow things down a bit.

This tends to worsen when Python needs to do loads of operations over and again, such as adding two arrays. Python must check each element's data type whenever an operation has to be performed, slowing things down, but we can get over this by using ufuncs.

NumPy speeds things up with something called vectorization. On a ndarray in compiled code, vectorization will perform one operation on each element in turn. In this way, there is no need to determine the elements' data types each time, thus making things faster.

A ufunc is a universal function, nothing more than a mathematical function that performs high-speed element-by-element operations. When you do simple math operations on an array, the ufunc is automatically called because the arrays are ufuncs wrappers.

For example, when you use the + operator to add NumPy arrays, a ufunc called add() is called automatically under the hood and does what it has to do quietly. This is how a Python list would look:

```
a = [1,2,3,4,5]
b = [6,7,8,9,10]
%timeit a+b
```

While this is the same operation as a NumPy array:

```
a = np.arange(1,6)
b = np.arange(6,11)
%timeit a+b
```

As you can see, using ufuncs, the array addition takes much less time.

Doing Math with NumPy Arrays

Math operations are common in Python, never more so than in NumPy arrays, and these are some of the more useful ones you will perform:

Basic Arithmetic

Those with Python experience will know all about arithmetic operations and how easy they are to perform. And they are just as easy to do on NumPy arrays. All you really need to remember is that the operation symbols play the role of ufuncs wrappers:

```
print('Subtract :',a-5)
print('Multiply :',a*5)
print('Divide :',a/5)
print('Power :',a**2)
print('Remainder :',a%5)
Subtract : [-4 -3 -2 -1 0]
Multiply : [ 5 10 15 20 25]
Divide : [0.2 0.4 0.6 0.8 1. ]
Power : [ 1 4 9 16 25]
Remainder : [1 2 3 4 0]
```

Mean, Median and Standard Deviation

Finding the mean, median and standard deviation on NumPy arrays requires the use of three methods – mean(), median(), and std():

```
a = np.arange(5,15,2)
print('Mean :',np.mean(a))
print('Standard deviation :',np.std(a))
print('Median :',np.median(a))
Mean : 9.0
Standard deviation : 2.8284271247461903
Median : 9.0
```

Min-Max Values and Their Indexes

In a ndarray, we can use the min() and max() methods to find the min and max values:

```
a = np.array([[1,6],
[4,3]])
# minimum along a column
print('Min :',np.min(a,axis=0))
# maximum along a row
print('Max :',np.max(a,axis=1))
Min : [1 3]
Max : [6 4]
```

You can also use the argmin() and argmax() methods to get the minimum or maximum value indexes along a specified axis:

```
a = np.array([[1,6,5],
[4,3,7]])
# minimum along a column
print('Min :',np.argmin(a,axis=0))
# maximum along a row
print('Max :',np.argmax(a,axis=1))
Min : [0 1 0]
Max : [1 2]
```

Breaking down the output, the first column's minimum value is the column's first element. The second column's minimum is the second element, while it's the first element for the third column.

Similarly, the outputs can be determined for the maximum values.

The Sorting Operation

Any good programmer will tell you that the most important thing to consider is an algorithm's time complexity. One basic yet important operation you are likely to use daily as a data scientist is sorting. For that reason, a good sorting algorithm must be used, and it must have the minimum time complexity.

The NumPy library is top of the pile when it comes to sorting an array's elements, with a decent range of sorting functions on

offer. And when you use the sort() method, you can also use mergesort, heapsort, and timesort, all available in NumPy, under the hood.

```
a = np.array([1,4,2,5,3,6,8,7,9])
np.sort(a, kind='quicksort')
array([1, 2, 3, 4, 5, 6, 7, 8, 9])
It is also possible to sort arrays on any axis you want:
a = np.array([[5,6,7,4],
          [9,2,3,7]])# sort along the column
print('Sort along column :','\n',np.sort(a,
kind='mergresort',axis=1))
# sort along the row
print('Sort along row :','\n',np.sort(a, kind='mergresort',axis=0))
Sort along column :
[[4 5 6 7]
[2 3 7 9]]
Sort along row :
[[5 2 3 4]
[9 6 7 7]]
```

NumPy Arrays and Images

NumPy arrays are widely used for storing image data and manipulating it, but what is image data?

Every image consists of pixels, and these are stored in arrays. Every pixel has a value of somewhere between 0 and 255, with 0 being a black pixel and 255 being a white pixel. Colored images comprise three two-dimensional arrays for the three color channels (RGB – Red, Green, Blue. These are back-to-back and make up a three-dimensional array. Each of the array's values stands for a pixel value so, the array size is dependent on how many pixels are on each dimension.

Python uses the scipy.misc.imread() method from the SciPy library to read images as arrays. The output is a three-dimensional array comprised of the pixel values:

```
import numpy as np
```

```
import matplotlib.pyplot as plt
from scipy import misc

# read image
im = misc.imread('./original.jpg')
# image
im
array([[[115, 106,  67],
        [113, 104,  65],
        [112, 103,  64],
        ...,
        [160, 138,  37],
        [160, 138,  37],
        [160, 138,  37]],

       [[117, 108,  69],
        [115, 106,  67],
        [114, 105,  66],
        ...,
        [157, 135,  36],
        [157, 135,  34],
        [158, 136,  37]],

       [[120, 110,  74],
        [118, 108,  72],
        [117, 107,  71],
        ...,
```

Checking the type and shape of the array is done like this:

```
print(im.shape)
print(type(type))
(561, 997, 3)
numpy.ndarray
```

Because images are nothing more than arrays, we can use array functions to manipulate them. For example, the image could be horizontally flipped using a method called np.flip():

```
# flip
plt.imshow(np.flip(im, axis=1))
```

Or the pixels value range could be changed or normalized, especially if you need faster computation:

```
im/255
array([[[0.45098039, 0.41568627, 0.2627451 ],
        [0.44313725, 0.40784314, 0.25490196],
        [0.43921569, 0.40392157, 0.25098039],
        ...,
        [0.62745098, 0.54117647, 0.14509804],
        [0.62745098, 0.54117647, 0.14509804],
        [0.62745098, 0.54117647, 0.14509804]],

       [[0.45882353, 0.42352941, 0.27058824],
        [0.45098039, 0.41568627, 0.2627451 ],
        [0.44705882, 0.41176471, 0.25882353],
        ...,
        [0.61568627, 0.52941176, 0.14117647],
        [0.61568627, 0.52941176, 0.13333333],
        [0.61960784, 0.53333333, 0.14509804]],

       [[0.47058824, 0.43137255, 0.29019608],
        [0.4627451 , 0.42352941, 0.28235294],
        [0.45882353, 0.41960784, 0.27843137],
        ...,
        [0.6       , 0.52156863, 0.14117647],
        [0.6       , 0.52156863, 0.13333333],
        [0.6       , 0.52156863, 0.14117647]],

       ...,
```

That is as much of an introduction as I can give you to NumPy, showing you the basic methods and operations you are likely to use as a data scientist. Next, we discover how to manipulate data using Pandas.

Part Three

Data Manipulation with Pandas

Pandas is another incredibly popular Python data science package, and there is a good reason for that. It offers users flexible, expressive, and powerful data structures that make data analysis and manipulation dead simple. One of those structures is the DataFrame.

In this part, we will look at Pandas DataFrames, including fundamental manipulation, up to the more advanced operations. To do that, we will answer the top 11 questions asked to provide you with the answers you need.

What Is a DataFrame?

First, let's look at what a DataFrame is.

If you have any experience using the R language, you will already know that a data frame provides rectangular grids to store data so it can be looked at easily. The rows in the grids all correspond to a measurement or an instance value, and the columns are vectors with data relating to a specific variable. So, there is no need for the rows to contain identical value types, although they can if needed. They can store any type of value, such as logical, character, numeric, etc.

Python DataFrames are much the same. They are included in the Pandas library, defined as two-dimensional data structures, labeled, and with columns containing possible different types.

It would be fair to say that there are three main components in a Pandas DataFrames – data, index, and columns.

First, we can store the following type of data in a DataFrame:

- A Pandas DataFrame
- A Pandas Series, a labeled one-dimensional array that can hold any data type with an index or axis label. A simple example would a DataFrame column
- A NumPy ndarray, which could be a structure or record
- A two-dimensional ndarray
- A dictionary containing lists, Series, dictionaries or one-dimensional ndarrays

NOTE

Do not confuse np.ndarray with np.array(). The first is a data type, and the second is a function that uses other data structures to make arrays.

With a structured array, a user can manipulate data via fields, each named differently. The example below shows a structured array being created containing three tuples. Each tuple's first element is called foo and is the int type. The second element is a float called bar.

The same example shows a record array. These are used to expand the properties of a structured array, and users can access the fields by attribute and not index. As you can see in the example, we access the foo values in the record array called r2:

```
# A structured array
my_array = np.ones(3, dtype=([('foo', int), ('bar', float
)]))
# Print the structured array
_____(my_array['foo'])
# A record array
my_array2 = my_array.view(np.recarray)
# Print the record array
_____(my_array2.foo)
```

Second, the index and column names can also be specified for the DataFrame. The index provides the row differences, while the column name indicates the column differences. Later, you will see that these are handy DataFrame components for data manipulation.

If you haven't already got Pandas or NumPy on your system, you can import them with the following import command:

```
import numpy as np
import pandas as pd
```

Now you know what DataFrames are and what you can use them for, let's learn all about them by answering those top 11 questions.

Question One: How Do I Create a Pandas DataFrame?

The first step when it comes to data munging or manipulation is to create your DataFrame, and you can do this in two ways – convert an existing one or create a brand new one. We'll only be looking at converting existing structures in this part, but we will discuss creating new ones later on.

NumPy ndarrays are just one thing that you can use as a DataFrame input. Creating a DataFrame from a NumPy array is pretty simple. All you need to do is pass the array to the DataFrame() function inside the data argument:

```
data = np.array([['','Col1','Col2'],
                 ['Row1',1,2],
                 ['Row2',3,4]])


print(pd.DataFrame(data=data[1:,1:],
                   index=data[1:,0],
                   columns=data[0,1:]))
```

One thing you need to pay attention to is how the DataFrame is constructed from NumPy array elements. First, the values from

lists starting with Row1 and Row2 are selected. Then the row numbers or index, Row1 or Row2, are selected, followed by the column names of Col1 and Col2.

Next, we print a small data selection. As you can see, this is much the same as when you subset and two-dimensional NumPy array – the row is indicated first, where you want your data looked for, followed by the column. Don't forget that, in Python, indices begin at 0. In the above example, we look in the rows in index 1 to the end and choose all elements after index 1. The result is 1, 2, 3, and 4 being selected.

This is the same approach to creating a DataFrame as for any structure taken as an input by DataFrame().

Try it for yourself on the next example:

```
# Take a 2D array as input to your DataFrame

    my_2darray = np.array([[1, 2, 3], [4, 5, 6]])
    print(_____________________)

# Take a dictionary as input to your DataFrame

    my_dict = {1: ['1', '3'], 2: ['1', '2'], 3: ['2', '4']}
    print(_____________________)

# Take a DataFrame as input to your DataFrame

    my_df = pd.DataFrame(data=[4,5,6,7], index=range(0,4),
    columns=['A'])
    print(_____________________)

# Take a Series as input to your DataFrame

    my_series = pd.Series({"Belgium":"Brussels", "India":"New
    Delhi", "United Kingdom":"London", "United States"
    :"Washington"})
    print(_____________________)
```

Your Series and DataFrame index has the original dictionary keys, but in a sorted manner – index 0 is Belgium, while index 3 is the

United States. Once your DataFrame is created, you can find out some things about it. For example, the len() function or shape property can be used with the .index property:

```
df = pd.DataFrame(np.array([[1, 2, 3], [4, 5, 6]]))

# Use the `shape` property

print(_________)

# Or use the `len()` function with the `index` property

print(_____________)
```

These options provide differing DataFrame information. We get the DataFrame dimensions using the shape property, which means the height and width of the DataFrame. And, if you use the index property and len() function together, you will only get the DataFrame height.

The df[0].count() function could also be used to find out more information about the DataFrame height but, should there be any NaN values, this won't provide them, which is why .count() is not always the best option.

Okay, now we know how to create a DataFrame from an existing structure, it's time to get down to the real work and start looking at some of the basic operations.

Question Two – How Do I Select a Column or Index from a DataFrame?

Before we can begin with basic operations, such as deleting, adding, or renaming, we need to learn how to select the elements. It isn't difficult to select a value, column, or index from a DataFrame. In fact, it's much the same as it is in other data analysis languages. Here's an example of the values in the DataFrame:

```
  A B C
0 1 2 3
```

```
1 4 5 6
2 7 8 9
```

We want to get access to the value in column A at index 0 and we have several options to do that:

```
# Using `iloc[]`
print(df.iloc[0][0])
# Using `loc[]`
print(df.loc[0]['A'])
# Using `at[]`
print(df.at[0,'A'])
# Using `iat[]`
print(df.iat[0,0])
```

There are two of these that you must remember - .iloc[] and .loc[]. There are some differences between these, but we'll discuss those shortly.

Now let's look at how to select some values. If you wanted rows and columns, you would do this:

```
# Use `iloc[]` to select row `0`
print(df.iloc[_])
# Use `loc[]` to select column ``'A''`
print(df.loc[:,'_'])
```

Right now, it's enough for you to know that values can be accessed by using their label name or their index or column position.

Question Three: How Do I Add a Row, Column, or Index to a DataFrame?

Now you know how to select values, it's time to understand how a row, column, or index is added.

Adding an Index

When a DataFrame is created, you want to ensure that you get the right index so you can add some input to the index argument. If this is not specified, by default, the DataFrame's index will be of

numerical value and will start with 0, running until the last row in the DataFrame.

However, even if your index is automatically specified, you can still re-use a column, turning it into your index. We call set_index() on the DataFrame to do this:

```
# Print out your DataFrame `df` to check it out
print(__)
# Set 'C' as the index of your DataFrame
df._____('C')
```

Adding a Row

Before reaching the solution, we should first look at the concept of loc and how different it is from .iloc, .ix, and other indexing attributes:

- **.loc[]** – works on your index labels. For example, if you have loc[2], you are looking for DataFrame values with an index labeled 2.
- **.iloc[]** – works on your index positions. For example, if you have iloc[2], you are looking for DataFrame values with an index labeled 2.
- **.ix[]** – this is a little more complex. If you have an integer-based index, a label is passed to .ix[]. In that case, .ix[2] means you are looking for DataFrame values with an index labeled 2. This is similar to .loc[] but, where you don't have an index that is purely integer-bases, .ix also works with positions.

Let's make this a little simpler with the help of an example:

```
df = pd.DataFrame(data=np.array([[1, 2, 3], [4, 5, 6], [7
, 8, 9]]), index= [2, 'A', 4], columns=[48, 49, 50])
# Pass `2` to `loc`
print(df.loc[_])
# Pass `2` to `iloc`
print(df.iloc[_])
```

```
# Pass `2` to `ix`
print(df.ix[_])
```

Note that we used a DataFrame not based solely on integers, just to show you the differences between them. As you can clearly see, you don't get the same result when you pass 2 to .loc[] or .iloc[]/.ix[].

.loc[] will examine those values labeled 2, and you might get something like the following returned:

```
48  1
49  2
 3
```

.iloc[] looks at the index position, and passing 2 will give you something like this:

```
48  7
49  8
50  9
```

Because we don't just have integers in the index, .ix[] behaves the same way as .iloc[] and examines the index positions. In that case, you get the same results that .iloc[] returned.

Understanding the difference between those three is a critical part of adding rows to a DataFrame. You should also have realized that the recommendation is to use .loc[] when you insert rows. If you were to use df.ix[], you might find you are referencing numerically indexed values, and you could overwrite an existing DataFrame row, all be it accidentally.

Once again, have a look at the example below"

```
df = pd._________(data=np.array([[1, 2, 3], [4, 5, 6], [7
, 8, 9]]), index= [2.5, 12.6, 4.8], columns=[48, 49, 50])
```

There's no index labeled `2`, so you will change the

```
index at position `2`
df.ix[2] = [60, 50, 40]
```

```
print(df)
```

This will make an index labeled `2` and add the new

```
values
df.loc[2] = [11, 12, 13]
print(df)
```

It’s easy to see how this is all confusing.

Adding a Column

Sometimes, you may want your index to be a part of a DataFrame, and this can be done by assigning a DataFrame column or referencing and assigning one that you haven't created yet. This is done like this:

```
df = pd._____(data=np.array([[1, 2, 3], [4, 5, 6], [7,
8, 9]]), columns=['A', 'B', 'C'])
# Use `.index`
df['D'] = df.index
# Print `df`
print(__)
```

What you are doing here is telling the DataFrame that the index should be column A.

However, appending columns could be done in the same way as adding an index - .loc[] or .iloc[] is used but, this time, a Series is added to the DataFrame using .loc[]:

Study the DataFrame `df`

```
print(__)
# Append a column to `df`
df.loc[:, 4] = pd.Series(['5', '6'], index=df.index)
# Print out `df` again to see the changes
_____(__)
```

Don't forget that a Series object is similar to a DataFrame's column, which explains why it is easy to add them to existing

DataFrames. Also, note that the previous observation we made about .loc[] remains valid, even when adding columns.

Resetting the Index

If your index doesn't look quite how you want it, you can reset it using reset_index(). However, be aware when you are doing this because it is possible to pass arguments that can break your reset.

```
# Check out the strange index of your DataFrame
print(df)
# Use `reset_index()` to reset the values.
df_reset = df._____(level=0, drop=True)
# Print `df_reset`
print(df_reset)
```

Use the code above but replace drop with inplace and see what would happen.

The drop argument is useful for indicating that you want the existing index eliminated, but if you use inplace, the original index will be added, with floats, as an additional column.

Question Four: How Do I Delete Indices, Rows, or Columns From a Data Frame?

Now you know how to add rows, columns, and indices, let's look at how to remove them from the data structure.

Deleting an Index

Removing an index from a DataFrame isn't always a wise idea because Series and DataFrames should always have one. However, you can try these instead:

- Reset your DataFrame index, as we did earlier
- If your index has a name, use del df.index.name to remove it

- Remove any duplicate values – to do this, reset the index, drop the index column duplicates, and reinstate the column with no duplicates as the index:

```
df = pd.DataFrame(data=np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9], [40, 50, 60], [23, 35, 37]]),
                  index= [2.5, 12.6, 4.8, 4.8, 2.5],
                  columns=[48, 49, 50])


df._____________.drop_duplicates(subset='index', keep='last').set_index('index')
```

- Finally, you can remove an index with a row – more about this later.

Deleting a Column

To remove one or more columns, the drop() method is used:

```
# Check out the DataFrame `df`
print(__)
# Drop the column with label 'A'
df._____('A', axis=1, inplace=True)
# Drop the column at position 1
df._____(df.columns[[1]], axis=1)
```

This might look too straightforward, and that's because the drop() method has got some extra arguments:

- When the axis argument indicates rows, it is 0, and when it drops columns, it is 1
- Inplace can be set to true, thus deleting the column without needing the DataFrame to be reassigned.

Removing a Row

You can use df.drop_duplicates() to remove duplicate rows but you can also do it by considering only the column's values:

```
# Check out your DataFrame `df`
print(__)
```

```
# Drop the duplicates in `df`
df._________([48], keep='last')
```

If you don't need to add any unique criteria to your deletion, the drop() method is ideal, using the index property for specifying the index for the row you want removed:

```
# Check out the DataFrame `df`
print(__)
# Drop the index at position 1
df._____(df.index[_])
```

After this, you can reset your index.

Question Five: How Do I Rename the Columns or Index of a DataFrame?

If you want to change the index or column values in your DataFrame different values, you should use a method called .rename():

```
# Check out your DataFrame `df`
print(__)
# Define the new names of your columns
newcols = {
   'A': 'new_column_1',
   'B': 'new_column_2',
   'C': 'new_column_3'
}
# Use `rename()` to rename your columns
df._______(columns=newcols, inplace=True)
# Use `rename()` to your index
df._______(index={1: 'a'})
```

If you were to change the inplace argument value to False, you would see that the DataFrame didn't get reassigned when the columns were renamed. This would result in the second part of the example taking an input of the original argument rather than the one returned from the rename() operation in the first part.

Question Six: How Do I Format the DataFrame Data?

On occasion, you may also want to do operations on the DataFrame values. We're going to look at a few of the most important methods to formatting DataFrame values.

Replacing All String Occurrences

The replace() method can easily be used when you want to replace specific strings in the DataFrame. All you have to do is pass in the values you want to be changed and the new, replacement values, like this:

```
# Study the DataFrame `df` first
_____(df)
# Replace the strings by numerical values (0-4)
df._______(['Awful', 'Poor', 'OK', 'Acceptable',
'Perfect'], [0, 1, 2, 3, 4])
```

You can also use the regex argument to help you out with odd combinations of strings:

```
# Check out your DataFrame `df`
print(df)
# Replace strings by others with `regex`
df.replace({'\n': '<br>'}, regex=True)
```

Put simply, replace() is the best method to deal with replacing strings or values in the DataFrame.

Removing Bits of Strings from DataFrame Cells

It isn't easy to remove bits of strings and it can be a somewhat cumbersome task. However, there is a relatively easy solution:

```
# Check out your DataFrame
_____(df)
# Delete unwanted parts from the strings in the `result`
column
df['result'] = df['result'].map(lambda x: x.lstrip('+-'
).rstrip('aAbBcC'))
# Check out the result again
df
```

To apply the lambda function element-wise or over individual elements in the column, we use map() on the column's result. The map() function will take the string value and removes the + or – on the left and one or more of the aABbcC you see on the right.

Splitting Column Text into Several Rows

This is a bit more difficult but the example below walks you through what needs to be done:

```
# Inspect your DataFrame `df`
print(__)
# Split out the two values in the third row
# Make it a Series
# Stack the values
ticket_series = df['Ticket'].str.split(' ').apply(pd
.Series, 1).stack()
# Get rid of the stack:
# Drop the level to line up with the DataFrame
ticket_series.index = ticket_series.index.droplevel(-1)
# Make your `ticket_series` a dataframe
ticketdf = pd._________(ticket_series)
# Delete the `Ticket` column from your DataFrame
del df['Ticket']
# Join the `ticketdf` DataFrame to `df`
df.____(ticketdef)
# Check out the new `df`
df
```

What you are doing is:

- Inspecting the DataFrame. The values in the last column and the last row are long. It looks like we have two tickets because one guest took a partner with them to the concert.
- The Ticket column is removed from the DataFrame df and the strings on a space. This ensures that both tickets are in separate rows. These four values, which are the ticket numbers, are placed into a Series object:

```
0      1
```

```
0  23:44:55      NaN
1  66:77:88      NaN
2  43:68:05  56:34:12
```

- Something still isn't right because we can see NaN values. The Series must be stacked to avoid NaN values in the result.
- Next, we see the stacked Series:

```
0  0    23:44:55
1  0    66:77:88
2  0    43:68:05
   1    56:34:12
```

- That doesn't really work, either, and this why the level is dropped, so it lines up with the DataFrame:

```
0    23:44:55
1    66:77:88
2    43:68:05
2    56:34:12
dtype: object
```

- Now that is what you really want to see.
- Next, your Series is transformed into a DataFrame so it can be rejoined to the initial DataFrame. However, we don't want duplicates, so the original Ticket column must be deleted.

Applying Functions to Rows or Columns

You can apply functions to data in the DataFrame to change it. First, we make a lambda function:

```
doubler = lambda x: x*2
Next, we apply the function:
# Study the `df` DataFrame
_____(__)
# Apply the `doubler` function to the `A` DataFrame column
df['A'].apply(doubler)
```

Note the DataFrame row can also be selected and the doubler lambda function applied to it. You know how to select rows – by using .loc[] or .iloc[].

Next, something like the following would be executed, depending on whether your index is selected based on position or label:

```
df.loc[0].apply(doubler)
```

Here, the apply() function is only relevant to the doubler function on the DataFrame axis. This means you target the columns or the index – a row or a column in simple terms.

However, if you wanted it applied element-wise, the map() function can be used. Simply replace apply() with map(), and don't forget that the doubler still has to be passed to map() to ensure the values are multiplied by 2.

Let's assume this doubling function is to be applied to the entire DataFrame, not just the A column. In that case, the doubler function is applied used applymap() to every element in your DataFrame:

```
doubled_df = df.applymap(doubler)
print(doubled_df)
```

Here, we've been working with anonymous functions create at runtime or lambda functions but you can create your own:

```
def doubler(x):
   if x % 2 == 0:
      return x
   else:
      return x * 2
# Use `applymap()` to apply `doubler()` to your DataFrame
doubled_df = df.applymap(doubler)
# Check the DataFrame
print(doubled_df)
```

Question Seven: How Do I Create an Empty DataFrame?

We will use the DataFrame() function for this, passing in the data we want to be included, along with the columns and indices. Remember, you don't have to use homogenous data for this – it can be of any type.

There are a few ways to use the DataFrame() function to create empty DataFrames. First, numpy.nan can be used to initialize it with NaNs. The numpy.nan function has a float data type:

```
df = pd.DataFrame(np.nan, index=[0,1,2,3], columns=['A'])
print(df)
```

At the moment, this DataFrame's data type is inferred. This is the default and, because numpy.nan has the float data type, that is the type of values the DataFrame will contain. However, the DataFrame can also be forced to be a specific type by using the dtype attribute and supplying the type you want, like this:

```
df = pd.DataFrame(index=range(0,4),columns=['A'], dtype
='float')
print(df)
```

If the index or axis labels are not specified, the input data is used to construct them, using common sense rules.

Question Eight: When I Import Data, Will Pandas Recognize Dates?

Yes, Pandas can recognize dates but only with a bit of help. When you read the data in, for example, from a CSV file, the parse_dates argument should be added:

```
import pandas as pd
pd.read_csv('yourFile', parse_dates=True)

# or this option:
pd.read_csv('yourFile', parse_dates=['columnName'])
```

However, there is always the chance that you will come across odd date-time formats. There is nothing to worry about as it's

simple enough to construct a parser to handle this. For example, you could create a lambda function to take the DateTime and use a format string to control it:

```
import pandas as pd
dateparser = lambda x: pd.datetime.strptime(x, '%Y-%m-%d
%H:%M:%S')


# Which makes your read command:

    pd.read_csv(infile, parse_dates=['columnName'],
    date_parser=dateparse)

    # Or combine two columns into a single DateTime column
    pd.read_csv(infile, parse_dates={'datetime': ['date', 'time']},
    date_parser=dateparse)
```

Question Nine: When Should a DataFrame Be Reshaped? Why and How?

When you reshape a DataFrame, you transform it to ensure that the structure you get better fits your data analysis. We can infer from this that reshaping is less concerned with formatting the DataFrame values and more concerned with transforming the DataFrame shape.

So that tells you when and why you should reshape, but how do you do it?

You have a choice of three ways, and we'll look at each of them in turn:

Pivoting

The pivot() function is used for creating a derived table from the original. Three arguments can be passed with the function:

- **values –** lets you specify the values from the original DataFrame to bring into your pivot table.

- **columns –** whatever this argument is passed to becomes a column in the table.
- **index –** whatever this argument is passed to becomes an index.

```
# Import pandas
import _______ as pd
products = pd.DataFrame({'category': ['Cleaning',
'Cleaning', 'Entertainment', 'Entertainment', 'Tech',
'Tech'],
                'store': ['Walmart', 'Dia',
'Walmart', 'Fnac', 'Dia','Walmart'],
                'price':[11.42, 23.50, 19.99, 15
.95, 55.75, 111.55],
                'testscore': [4, 3, 5, 7, 5, 8]})
# Use `pivot()` to pivot the DataFrame
pivot_products = products._____(index='category', columns
='store', values='price')
# Check out the result
print(pivot_products)
```

If you don't specify the values you want on the table, you will end up pivoting by several columns:

```
# Import the Pandas library
import _______ as pd
# Construct the DataFrame
products = pd.DataFrame({'category': ['Cleaning',
'Cleaning', 'Entertainment', 'Entertainment', 'Tech',
'Tech'],
                'store': ['Walmart', 'Dia',
'Walmart', 'Fnac', 'Dia','Walmart'],
                'price':[11.42, 23.50, 19.99, 15
.95, 55.75, 111.55],
                'testscore': [4, 3, 5, 7, 5, 8]})
# Use `pivot()` to pivot your DataFrame
pivot_products = products._____(index='category', columns
='store')
# Check out the results
print(pivot_products)
```

The data may not have any rows containing duplicate values for the specified columns. If they do, an error message is thrown. If your data cannot be unique, you must use a different method, pivot_table():

```
# Import the Pandas library
import _______ as pd
# Your DataFrame
products = pd.DataFrame({'category': ['Cleaning',
'Cleaning', 'Entertainment', 'Entertainment', 'Tech',
'Tech'],
                'store': ['Walmart', 'Dia',
'Walmart', 'Fnac', 'Dia','Walmart'],
                'price':[11.42, 23.50, 19.99, 15
.95, 19.99, 111.55],
                'testscore': [4, 3, 5, 7, 5, 8]})
# Pivot your `products` DataFrame with `pivot_table()`
pivot_products = products._____________(index='category',
columns='store', values='price', aggfunc='mean')
# Check out the results
print(pivot_products)
```

We passed an extra argument to the pivot_table method, called aggfunc. This indicates that multiple values are combined with an aggregation function. In our example, we used the mean function.

Reshaping a DataFrame Using Stack() and Unstack()

If you refer back to question five, you will see we already used a stacking example. When a DataFrame is stacked, it is made taller. The innermost column index is moved to become the innermost row index, thus returning a DataFrame with an index containing a new innermost row labels level.

Conversely, we can use unstack() to make the innermost row index into the innermost column index.

Using melt() to Reshape a DataFrame

Melting is useful when your data has at least one column that is an identifier variable, and all others are measured variables.

Measured variables are considered as unpivoted to the row axis. This means that, where the measured variables were scattered over the DataFrame's width, melt() will ensure they are placed to its height. So, instead of becoming wider, the DataFrame becomes longer. This results in a pair of non-identifier columns called value and variable. Here's an example to show you what this means:

```
# The `people` DataFrame
people = pd.DataFrame({'FirstName' : ['John', 'Jane'],
                'LastName' : ['Doe', 'Austen'],
                'BloodType' : ['A-', 'B+'],
                'Weight' : [90, 64]})
# Use `melt()` on the `people` DataFrame
print(pd._____(people, id_vars=['FirstName', 'LastName'],
var_name='measurements'))
```

Question Ten: How Do I Iterate Over a DataFrame?

Iterating over the DataFrame rows is done with a for loop and calling iterrows() on the DataFrame:

```
df = pd.DataFrame(data=np.array([[1, 2, 3], [4, 5, 6], [7,
8, 9]]), columns=['A', 'B', 'C'])
for index, row in df.iterrows() :
   print(row['A'], row['B'])
```

Using iterrows() lets you loop efficiently over the rows as pairs – index, Series. In simple terms, the result will be (index, row) tuples.

Question Eleven: How Do I Write a DataFrame to a File?

When you have finished doing all you want to do, you will probably want to export your DataFrame into another format. We'll look at two ways to do this – Excel or CSV file.

Outputting to a CSV File

Writing to a CSV file requires the use of to_csv():

```
import pandas as pd
```

```
df.to_csv('myDataFrame.csv')
```

That looks like a simple code but, for many people, it's where things get difficult. This is because everyone will have their own requirements for the data output. You may not want to use commas as the delimiter or want a specific type of encoding.

We can get around this with specific arguments passed to to_csv() to ensure you get the output in the format you want:

- Delimiting a tab requires the sep argument:

```
import pandas as pd
df.to_csv('myDataFrame.csv', sep='\t')
```

- The encoding argument allows you to choose the character encoding you want:

```
import pandas as pd
df.to_csv('myDataFrame.csv', sep='\t', encoding='utf-8')
```

- You can even specify how to represent missing or NaN values – you may or may not want the header outputted, you may or may not want the row names written, you may want compression, and so on. You can find out how to do all that by reading this page .

Outputting to an Excel File

For this one, you use to_excel() but it isn't quite as straightforward as the CSV file:

```
import pandas as pd
writer = pd.ExcelWriter('myDataFrame.xlsx')
df.to_excel(writer, 'DataFrame')
writer.save()
```

However, it's worth noting that you require many additional arguments, just like you do with to_csv, to ensure your data is output as you want. These include startrow, startcol, and more, and you can find out about them on this page .

More than Just DataFrames

While that was a pretty basic Pandas tutorial, it gives you a good idea of using it. Those 11 questions we answered were the most commonly asked ones, and they represent the fundamental skills you need to import your data, clean it and manipulate it.

In the next chapter, we'll take a brief look at data visualization with Matplotlib and Seaborn.

Part Four

Data Visualization with Matplotlib and Seaborn

Data visualization is one of the most important parts of communicating your results to others. It doesn't matter whether it is in the form of pie charts, scatter plots, bar charts, histograms, or one of the many other forms of visualization; getting it right is key to unlocking useful insights from the data. Thankfully, data visualization is easy with Python.

It's fair to say that data visualization is a key part of analytical tasks, like exploratory data analysis, data summarization, model output analysis, and more. Python offers plenty of libraries and tools to help us gain good insights from data, and the most commonly used is Matplotlib. This library enables us to generate all different visualizations, all from the same code if you want.

Another useful library is Seaborn, built on Matplotlib, providing highly aesthetic data visualizations that are sophisticated, statistically speaking. Understanding these libraries is critical for data scientists to make the most out of their data analysis.

Using Matplotlib to Generate Histograms

Matplotlib is packed with tools that help provide quick data visualization. For example, researchers who want to analyze new data sets often want to look at how values are distributed over a set of columns, and this is best done using a histogram.

A histogram generates approximations to distributions by using a set range to select results, placing each value set in a bucket or

bin. Using Matplotlib makes this kind of visualization straightforward.

We'll be using the FIFA19 dataset, which you can download from here . Run these codes to see the outputs for yourself.

To start, if you haven't already done so, import Pandas:

```
import pandas as pd
```

Next, the pyplot module from Matplotlib is required, and the customary way of importing it is as plt:

```
import matplotlib.pyplot as plt
```

Next, the data needs to be read into a DataFrame. We'll use the set_option() method from Pandas to relax the display limit of rows and columns:

```
df = pd.read_csv("fifa19.csv")
pd.set_option('display.max_columns', None)
pd.set_option('display.max_rows', None)
```

Now we can print the top five rows of the data, and this is done with the head() method:

```
print(df.head())
```

A histogram can be generated for any numerical column. The hist() method is called on the plt object, and the selected DataFrame column is passed in. We'll try this on the Overall column because it corresponds to the overall player rating:

```
plt.hist(df['Overall'])
```

The a and y axes can also be labeled, along with the plot title, using three methods:

- xlabel()
- ylabel()
- title()

```
plt.xlabel('Overall')

plt.ylabel('Frequency')

plt.title('Histogram of Overall Rating')

plt.show()
```

This histogram is one of the best ways to see the distribution of values and see which ones occur the most and the least.

Using Matplotlib to Generate Scatter Plots

Scatterplots are another useful visualization that shows variable dependence. For example, if we wanted to see if a positive relationship existed between wage and overall player rating (if the wage increases, does the rating go up?), we can use the scatterplot to find it.

Before generating the scatterplot, we need the wage column converted from string to floating-point, which is a numerical column. To do this, a new column is created, named wage_euro:

```
df['wage_euro'] = df['Wage'].str.strip('€')

df['wage_euro'] = df['wage_euro'].str.strip('K')

df['wage_euro'] = df['wage_euro'].astype(float)*1000.0
```

Now, let's display our new column wage_euro and the overall column:

```
print(df[['Overall', 'wage_euro']].head())
```

The result would look like this:

```
  Overall wage_euro
0 94   565000.0
1 94   405000.0
2 92   290000.0
3 91   260000.0
4 91   355000.0
```

Generating a Matplotlib scatter plot is as simple as using scatter() on the plt object. We can also label each axis and provide the plot with a title:

```
plt.scatter(df['Overall'], df['wage_euro'])
plt.title('Overall vs. Wage')
plt.ylabel('Wage')
plt.xlabel('Overall')
plt.show()
```

Using Matplotlib to Generate Bar Charts

Another good visualization tool is the bar chart, useful for analyzing data categories. For example, using our FIFA19 data set, we might want to look at the most common nationalities, and bar charts can help us with this. Visualizing categorical columns requires the values to be counted first. We can generate a dictionary of count values for every one of the categorical column categories using the count method. We'll do that for the nationality column:

```
from collections import Counter

print(Counter(df[‘Nationality’]))
```

The result would be a long list of nationalities, starting with the most values and ending with the least.

This dictionary can be filtered using a method called most_common. We'll ask for the ten most common nationalities in this case but, if you wanted to see the least common, you could use the least_common method:

```
print(dict(Counter(df[‘Nationality’]).most_common(10)))
```

Again, the result would be a dictionary containing the ten most common nationalities in order of the most values to the least.

Finally, the bar chart can be generated by calling the bar method on the plt object, passing the dictionary keys and values in:

```
nationality_dict =
dict(Counter(df['Nationality']).most_common(10))

plt.bar(nationality_dict.keys(), nationality_dict.values())

plt.xlabel('Nationality')

plt.ylabel('Frequency')

plt.title('Bar Plot of Ten Most Common Nationalities')

plt.xticks(rotation=90)

plt.show()
```

You can see from the resulting bar chart that the x-axis values overlap, making them difficult to see. If we want to rotate the values, we use the xticks() method, like this:

```
plt.xticks(rotation=90)
```

Using Matplotlib to Generate Pie Charts

Pie charts are an excellent way of visualizing proportions in the data. Using our FIFA19 dataset, we could visualize the player proportions for three countries – England, Germany, and Spain – using a pie chart.

First, we create four new columns, one each for England, Germany, Spain, and Other, which contains all the other nationalities:

```
df.loc[df.Nationality =='England', 'Nationality2'] = 'England'

df.loc[df.Nationality =='Spain', 'Nationality2'] = 'Spain'

df.loc[df.Nationality =='Germany', 'Nationality2'] = 'Germany'

df.loc[~df.Nationality.isin(['England', 'German', 'Spain']),
'Nationality2'] = 'Other'
```

Next, let's create a dictionary that contains the proportion values for each of these:

```
prop = dict(Counter(df['Nationality2']))

for key, values in prop.items():

    prop[key] = (values)/len(df)*100

print(prop)
```

The result shows you a dictionary or list showing Other, Spain, and England, with their player proportions. From here, we can use the Matplotlib pie method to create a pie chart from the dictionary:

```
fig1, ax1 = plt.subplots()

ax1.pie(prop.values(), labels=prop.keys(), autopct='%1.1f%%',

        shadow=True, startangle=90)

ax1.axis('equal')  # Equal aspect ratio ensures that pie is drawn as a
circle.

plt.show()
```

As you can see, these methods all of us very powerful ways of visualizing category proportions in the data.

Visualizing Data with Seaborn

Seaborn is another useful library, built on Matplotlib and offering a powerful plot formatting method. Once you have got to grips with Matplotlib, you should move on to Seaborn to use more complex visualizations.

For example, we can use the set() method from Seaborn to improve how our Matplotlib plots look.

First, Seaborn needs to be imported, and all the figures generated earlier need to be reformatted. Add the following to the top of your script and run it:

```
import seaborn as sns

sns.set()
```

Using Seaborn to Generate Histograms

We can use Seaborn to generate the same visualizations we generated with Matplotlib. First, we'll generate the histogram showing the overall column. This is done on the Seaborn object with distplot:

```
sns.distplot(df['Overall'])
```

We can also reuse our plt object to make additional formats on the axis and set the title:

```
plt.xlabel('Overall')

plt.ylabel('Frequency')

plt.title('Histogram of Overall Rating')

plt.show()
```

As you can see, this is much better looking than the plot generated with Matplotlib.

Using Seaborn to Generate Scatter Plots

We can also use Seaborn to generate scatter plots in a straightforward way, so let's recreate our earlier one:

```
sns.scatterplot(df['Overall'], df['wage_euro'])

plt.title('Overall vs. Wage')

plt.ylabel('Wage')
```

```
plt.xlabel('Overall')

plt.show()
```

Using Seaborn to Generate Heatmaps

Seaborn is perhaps best known for its ability to create correlation heatmaps. These are used for identifying variable independence and, before generating them, the correlation between a specified set of numerical columns must be calculated. We'll do this for four columns:

- age
- overall
- wage_euro
- skill moves

```
corr = df[['Overall', 'Age', 'wage_euro', 'Skill Moves']].corr()

sns.heatmap(corr, annot=True)

plt.title('Heatmap of Overall, Age, wage_euro, and Skill Moves')

plt.show()
```

If we want to see the correlation values, annot can be set to true:

```
sns.heatmap(corr, annot=True)
```

Using Seaborn to Generate Pairs Plot

This is the final tool we'll look at – a method called pairplot – which allows for a matrix of distributions to be generated, along with a scatter plot for a specified numerical feature set. We'll do this with three columns:

- age
- overall

- potential

```
data = df[['Overall', 'Age', 'Potential',]]

sns.pairplot(data)

plt.show()
```

This proves to be one of the quickest and easiest ways to visualize the relationships between the variables and the numerical value distributions using scatter plots.

Matplotlib and Seaborn are both excellent tools, very valuable in the data science world. Matplotlib helps you label, title, and format graphs, one of the most important factors in effectively communicating your results. It also gives many of the fundamental tools we need to produce visualizations, such as scatter plots, histograms, bar charts, and pie charts.

Seaborn is also a very important library because it offers in-depth statistical tools and stunning visuals. As you saw, the Seaborn-generated visuals were much more aesthetically pleasing to look at than the Matplotlib visuals. The Seaborn tools also allow for more sophisticated visuals and analyses.

Both libraries are popular in data visualization, both allowing quick data visualization to tell stories with the data. There is some overlap in their use cases, but it is wise to learn both to ensure you do the best job with your data as you can.

Now buckle up and settle in. The next part is by far the longest as we dive deep into the most important subset of data science – machine learning.

Part Five

An In-Depth Guide to Machine Learning

The term “machine learning” was first coined in 1959 by Arthur Samuel, an AI and computer gaming pioneer who defined it as “a field of study that provides computers with the capability of learning without explicit programming.”

That means that machine learning is an AI application that allows software to learn by experience, continually improving without explicit programming. For example, think about writing a program to identify vegetables based on properties such as shape, size, color, etc.

You could hardcode it all, add a few rules which you use to identify the vegetables. To some, this might seem to be the only way that will work, but there isn’t one programmer who can write the perfect rules that will work for all cases. That’s where machine learning comes in, without any rules – this makes it more practical and robust. Throughout this chapter, you will see how machine learning can be used for this task.

So, we could say that machine learning is a study in making machines exhibit human behavior in terms of decision making and behavior by allowing them to learn with little human intervention – which means no explicit programming. This begs the question, how does a machine gain experience, and where does it learn from? The answer to that is simple – data, the fuel that powers machine learning. Without it, there is no machine learning.

One more question – if machine learning was first mentioned can k in 1959, why has it taken so long to become mainstream? The main reason for this is the significant amount of computing power machine learning requires, not to mention the hardware capable of storing vast amounts of data. It's only in recent years that we have achieved the requirements needed to practice machine learning.

How Do Machine Learning and Traditional Programming Differ?

Traditional programming involves input data being fed into a machine, together with a clean, tested program, to generate some output. With machine learning, the input and the output data are fed to the machine during the initial phase, known as the learning phase. From there, the machine will work the programming out for itself.

Don't worry if this doesn't make sense right now. As you work through this in-depth chapter, it will all come together.

Why We Need Machine Learning

These days, machine learning has more attention than you would think possible. One of its primary uses is to automate tasks that require human intelligence to perform. Machines can only replicate this intelligence via machine learning.

Businesses use machine learning to automate tasks and automate and create data analysis models. Some industries are heavily reliant on huge amounts of data which they use to optimize operations and make the right decisions. Machine learning makes it possible to create models to process and analyze complex data in vast amounts and provide accurate results. Those models are scalable and precise, and they function quickly, allowing businesses to leverage their huge power and the opportunities they provide while avoiding risks, both known and unknown.

Some of the many uses of machine learning in the real world include text generation, image recognition, and so on, this increasing the scope, ensuring machine learning experts become highly sought after individuals.

How Does Machine Learning Work?

Machine learning models are fed historical data, known as training data. They learn from that and then build a prediction algorithm that predicts an output for a completely different set of data, known as testing data, that is given as an input to the system. How accurate the models are is entirely dependent on the quality of the data and how much of it is provided – the more quality data there is, the better the results and accuracy are.

Let's say we have a highly complex issue that requires predictions. Rather than writing complex code, we could give the data to machine learning algorithms. These develop logic and provide predictions, and we'll be discussing the different types of algorithms shortly.

Machine Learning Past to Present

These days, we see plenty of evidence of machine learning in action, such as Natural Language Processing and self-drive vehicles, to name but two. Machine learning has been in existence for more than 70 years, though, all beginning in 1943. Warren McCulloch, a neuropsychologist, and Water Pitts, a mathematician, produced a paper about neurons and the way they work. They went on to create a model with an electrical circuit, producing the first-ever neural network.

Alan Turing created the Turing test in 1950. This test was used to determine whether computers possessed real intelligence, and passing the test required the computer to fool a human into believing it was a human.

Arthur Samuel wrote the very first computer learning program in 1952, a game of checkers that the computer improved at the

more it played. It did this by studying the moves, determining which ones came into the winning strategies and then using those moves in its own program.

In 1957, Frank Rosenblatt designed the perceptron, a neural network for computers that simulates human brain thought processes. Ten years later, an algorithm was created called "Nearest Neighbor." This algorithm allowed computers to begin using pattern recognition, albeit very basic at the time. This was ideal for salesmen to map routes, ensuring all their visits could be completed within the shortest time.

It was in the 1990s that the biggest changes became apparent. Work was moving from an approach driven almost entirely by knowledge to a more data-driven one. Scientists started creating programs that allowed computers to analyze huge swathes of data, learning from the results to draw conclusions.

In 1997, Deep Blue, created by IBM, was the first computer-based chess player to beat a world champion at his own game. The program searched through potential moves, using its computing power to choose the best ones. And just ten years later, Geoffrey Hinton coined the term "deep" learning to describe the new algorithms that allowed computers to recognize text and objects in images and videos.

In 2012, Alex Krizhevzky published a paper with Ilya Sutskever and Geoffrey Hinton. An influential paper, it described a computing model that could reduce the error rate significantly in image recognition systems. At the same time, Google's X Lab came up with an algorithm that could browse YouTube videos autonomously, choosing those that had cats in them.

In 2016, Google DeepMind researchers created AlphaGo to play the ancient Chinese game, Go, and pitted it against the world champion, Lee Sedol. AlphaGo beat the reigning world champion four times out of five.

Finally, in 2020, GPT-3 was released, arguably the most powerful language model in the world. Released by OpenAI, the model could generate fully functioning code, write creative fiction, write good business memos, and so much more, with its use cases only limited by imagination.

Machine Learning Features

Machine learning offers plenty of features, with the most prominent being the following:

- **Automation –** most email accounts have a spam folder where all your spam emails appear. You might question how your email provider knows which emails are spam, and the answer is machine learning. The process is automated after the algorithm is taught to recognize spam emails.

  Being able to automate repetitive tasks is one of machine learning's best features. Many businesses worldwide are using it to deal with email and paperwork automation. Take the financial sector, for example. Every day, lots of data-heavy, repetitive tasks need to be performed, and machine learning takes the brunt of this work, automating it to speed it up and freeing up valuable staff hours for more important tasks.

- **Better Customer Experience** – good customer relationships are critical to businesses, helping to promote brand loyalty and drive engagement. And the best way for a customer to do this is to provide better services and food customer experiences. Machine learning comes into play with both of these. Think about when you last saw ads on the internet or visited a shopping website. Most of the time, the ads are relevant to things you have searched for, and the shopping sites recommend products based on previous searches or purchases. Machine learning is

behind these incredibly accurate recommendation systems and ensures that we can tailor experiences to each user.

In terms of services, most businesses use chatbots to ensure that they are available 24/7. And some are so intelligent that many customers don't even realize they are not chatting to a real human being.

- **Automated Data Visualization –** these days, vast amounts of data are being generated daily by individuals and businesses, for example, Google, Facebook, and Twitter. Think about the sheer amount of data each of those must be generating daily. That data can be used to visualize relationships, enabling businesses to make decisions that benefit them and their customers. Using automated data visualization platforms, businesses can gain useful insights into the data and use them to improve customer experiences and customer service.
- **BI or Business Intelligence –** When machine learning characteristics are merged with data analytics, particularly big data, companies find it easier to find solutions to issues and use them to grow their business and increase their profit. Pretty much every industry uses this type of machine learning to increase operations.

  With Python, you get the flexibility to choose between scripting and object-oriented programming. You don't need to recompile any code – the changes can be implemented, and the results are shown instantly. It is the most versatile of all the programming languages and is multi-platforms, which is why it works so well for machine learning and data science.

<u>Different Types of Machine Learning</u>

Machine learning falls into three primary categories:

- Supervised learning

- Unsupervised learning
- Reinforcement learning

We'll look briefly at each type before we turn our attention to the popular machine learning algorithms.

Supervised Learning

We'll start with the easiest of examples to explain supervised learning. Let's say you are trying to teach your child how to tell the difference between a cat and a dog. How do you do it?

You point out a dog and tell your child, "that's a dog." Then you do the same with a cat. Show them enough and, eventually, they will be able to tell the difference. They may even begin to recognize different breeds in time, even if they haven't seen them before.

In the same way, supervised learning works with two sets of variables. One is a target variable or labels, which is the variable we are predicting, while the other is the features variable, which are the ones that help us make the predictions. The model is shown the features and the labels that go with them and will then find patterns in the data it looks at. To clear this up, below is an example taken from a dataset about house prices. We want to predict a house price based on its size. The target variable is the price, and the feature it depends on is the size.

Number of rooms	Price
1	$100
3	$300
5	$500

There are thousands of rows and multiple features in real datasets, such as the size, number of floors, location, etc.

So, the easiest way to explain supervised learning is to say that the model has x, which is a set of input variables, and y, which is

the output variable. An algorithm is used to find the mapping function between x and y, and the relationship is y = f(x).

It is called supervised learning because we already know what the output will be, and we provide it to the machine. Each time it runs, the algorithm is corrected, and this continues until we get the best possible results. The data set is used to train the algorithm to an optimal outcome.

Supervised learning problems can be grouped as follows:

- **Regression –** predicts future values. Historical data is used to train the model, and the prediction would be the future value of a property, for example.
- **Classification –** the algorithm is trained on various labels to identify things in a particular category, i.e., cats or dogs, oranges or apples, etc.

Unsupervised Learning

Unsupervised learning does not involve any target variables. The machine is given only the features to learn from, which it does alone and finds structures in the data. The idea is to find the distribution that underlies the data to learn more about the data.

Unsupervised learning can be grouped as follows:

- **Clustering –** the input variables sharing characteristics are put together, i.e., all users that search for the same things on the internet
- **Association –** the rules governing meaningful data associations are discovered, i.e., if a person watches this, they are likely to watch that too.

Reinforcement Learning

In reinforcement learning, models are trained to provide decisions based on a reward/feedback system. They are left to

learn how to achieve a goal in complex situations and, whenever they achieve the goal during the learning, they are rewarded.

It differs from supervised learning in that there isn't an available answer, so the agent determines what steps are needed to do the task. The machine uses its own experiences to learn when no training data is provided.

Common Machine Learning Algorithms

There are far too many machine learning algorithms to possibly mention them all, but we can talk about the most common ones you are likely to use. Later, we'll look at how to use some of these algorithms in machine learning examples, but for now, here's an overview of the most common ones.

Naïve Bayes Classifier Algorithm

The Naïve Bayes classifier is not just one algorithm. It is a family of classification algorithms, all based on the Bayes Theorem. They all share one common principle – each features pair being classified is independent of one another.

Let's dive straight into this with an example.

Let's use a fictional dataset describing weather conditions for deciding whether to play a golf game or not. Given a specific condition, the classification is Yes (fit to play) or No (unfit to play.)

The dataset looks something like this:

Outlook Temperature Humidity Windy Play Golf

0 Rainy Hot High False No

1 Rainy Hot High True No

2 Overcast Hot High False Yes

3 Sunny Mild High False Yes

4 Sunny Cool Normal False Yes

5 Sunny Cool Normal True No

6 Overcast Cool Normal True Yes

7 Rainy Mild High False No

8 Rainy Cool Normal False Yes

9 Sunny Mild Normal False Yes

10 Rainy Mild Normal True Yes

11 Overcast Mild High True Yes

12 Overcast Hot Normal False Yes

13 Sunny Mild High True No

This dataset is split in two:

- **Feature matrix –** where all the dataset rows (vectors) are stored – the vectors contain the dependent feature values. In our dataset, the four features are Outlook, Temperature, Windy, and Humidity.
- **Response vector –** where all the class variable values, i.e., prediction or output, are stored for each feature matrix row. In our dataset, the class variable name is called Play Golf.

Assumption

The Naïve Bayes classifiers share an assumption that all features will provide the outcome with an equal and independent contribution. As far as our dataset is concerned, we can understand this as:

- An assumption that there is no dependence between any features pair. For example, a temperature classified as Hot is independent of humidity, and a Rainy outlook is not

related to the wind. As such, we assume the features are independent.

- Each feature has the same importance (weight). For example, if you only know the humidity and the temperature, you cannot provide an accurate prediction. No attribute is considered irrelevant, and all play an equal part in determining the outcome.

Bayes' Theorem

The Bayes' Theorem uses the probability of an event that happened already to determine the probability of another one happening. The Theorem is mathematically stated as:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

A and B are both events and P(B)?0:

- We want to predict the probability of event A, should event B equal true. Event B is also called Evidence.
- P(A) is the prior probability of A, i.e., the probability an event will occur before any evidence is seen. That evidence consists of an unknown instance's attribute value – in our case, it is event B.
- P(A|B) is called a posteriori probability of B, or the probability an event will occur once the evidence is seen.

We can apply the Theorem to our dataset like this:

$$P(y|X) = \frac{P(X|y)P(y)}{P(X)}$$

Where the class variable is y and the dependent features vector, size n, is X, where

$$X = (x_1, x_2, x_3, ..., x_n)$$

Just to clear this up, a feature vector and its corresponding class variable example is:

X = (Rainy, Hot, High, False)

y = No

Here, P(y|X) indicates the probability of not playing, given a set of weather conditions as follows:

- Rainy outlook
- Hot temperature
- High humidity
- No wind

Naïve Assumption

Giving the Bayes Theorem a naïve assumption means providing independence between the features. The evidence can now be split into its independent parts. If any two events, A and B, are independent:

P(A,B) = P(A)P(B)

The result is:

$$P(y|x_1,...,x_n) = \frac{P(x_1|y)P(x_2|y)...P(x_n|y)P(y)}{P(x_1)P(x_2)...P(x_n)}$$

And this is expressed as:

$$P(y|x_1,...,x_n) = \frac{P(y)\prod_{i=1}^{n} P(x_i|y)}{P(x_1)P(x_2)...P(x_n)}$$

For a given input, the denominator stays constant so that term can be removed:

$$P(y|x_1,...,x_n) \propto P(y)\prod_{i=1}^{n} P(x_i|y)$$

The next step is creating a classifier model. We will determine the probability of a given input set for all class variable y's possible

values, choosing the output that offers the maximum probability. This is mathematically expressed as:

$$y = argmax_y P(y) \prod_{i=1}^{n} P(x_i|y)$$

That leaves us having to calculate P(y) and P(x_i | y).

P(y) is also known as the class probability, and P(x_i | y) is the conditional probability.

There are a few Naïve Bayes classifiers, and their main difference is via the assumptions about the P(x_i | y) distribution.

Let's apply this formula to our weather dataset manually. We will need to carry out some precomputations, i.e., for each x_i in X and y_i in y, we must find P(X=x_i | y_i). These calculations can be seen in the following tables:

Outlook

	Yes	No	P(yes)	P(no)
Sunny	2	3	2/9	3/5
Overcast	4	0	4/9	0/5
Rainy	3	2	3/9	2/5
Total	9	5	100%	100%

Temperature

	Yes	No	P(yes)	P(no)
Hot	2	2	2/9	2/5
Mild	4	2	4/9	2/5
Cool	3	1	3/9	1/5
Total	9	5	100%	100%

Humidity

	Yes	No	P(yes)	P(no)
High	3	4	3/9	4/5
Normal	6	1	6/9	1/5
Total	9	5	100%	100%

Wind

	Yes	No	P(yes)	P(no)
False	6	2	6/9	2/5
True	3	3	3/9	3/5
Total	9	5	100%	100%

Play		P(Yes)/P(No)
Yes	9	9/14
No	5	5/14
Total	14	100%

$P(X=x_i \mid y_i)$ is calculated for each x_i in X and y_i in y manually in the above tables. For example, we calculate the probability of playing given a cool temperature – P(temp. = cool | play gold = yes = 3/9.

We also need the (P(y)) class probabilities, calculated in the final table above. For example, the probabilities are calculated as P(play gold = yes) = 9/14.

Now the classifier is ready to go, so we'll test it on a feature set we'll call 'today.'

today = (Sunny, Hot, Normal, False)

Given this, the probability of playing is:

$$P(Yes|today) = \frac{P(SunnyOutlook|Yes)P(HotTemperature|Yes)P(NormalHumidity|Yes)P(NoWind|Yes)P(Yes)}{P(today)}$$

And not playing is:

$$P(No|today) = \frac{P(SunnyOutlook|No)P(HotTemperature|No)P(NormalHumidity|No)P(NoWind|No)P(No)}{P(today)}$$

P(today) appears in both of the probabilities, so we can ignore it and look for the proportional probabilities:

$$P(Yes|today) \propto \frac{2}{9}.\frac{2}{9}.\frac{6}{9}.\frac{6}{9}.\frac{9}{14} \approx 0.0141$$

and

$$P(No|today) \propto \frac{3}{5}.\frac{2}{5}.\frac{1}{5}.\frac{2}{5}.\frac{5}{14} \approx 0.0068$$

Because

$$P(Yes|today) + P(No|today) = 1$$

We can make the sum equal to 1 to convert the numbers into a probability – this is called normalization:

$$P(Yes|today) = \frac{0.0141}{0.0141+0.0068} = 0.67$$

and

$$P(No|today) = \frac{0.0068}{0.0141+0.0068} = 0.33$$

Because

$$P(Yes|today) > P(No|today)$$

In that case, we predict that Yes, golf will be played.

We've discussed a method that we can use for discrete data. Should we be working with continuous data, assumptions need to be made about each feature's value distribution.

We'll talk about one of the popular Naïve Bayes classifiers.

Gaussian Naive Bayes classifier

In this classifier, the assumption is that each feature's associated continuous values are distributed via a Gaussian distribution, also known as a Normal distribution. When this is plotted, you see a curve shaped like a bell, symmetric around the feature values' mean.

The assumption is that the likelihood of the features is Gaussian, which means the conditional probability is provided via:

$$P(x_i|y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} exp\left(-\frac{(x_i-\mu_y)^2}{2\sigma_y^2}\right)$$

Let's see what a Python implementation of this classifier looks like using Scikit-learn:

```
# load the iris dataset
from sklearn.datasets import load_iris
iris = load_iris()
# store the feature matrix (X) and response vector (y)
X = iris.data
y = iris.target


# splitting X and y into training and testing sets
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4,
random_state=1)


# training the model on training set
from sklearn.naive_bayes import GaussianNB
gnb = GaussianNB()
gnb.fit(X_train, y_train)
```

```
# making predictions on the testing set
y_pred = gnb.predict(X_test)


# comparing actual response values (y_test) with predicted
response values (y_pred)
from sklearn import metrics
print("Gaussian Naive Bayes model accuracy(in %):",
metrics.accuracy_score(y_test, y_pred)*100)
```

And the output is:

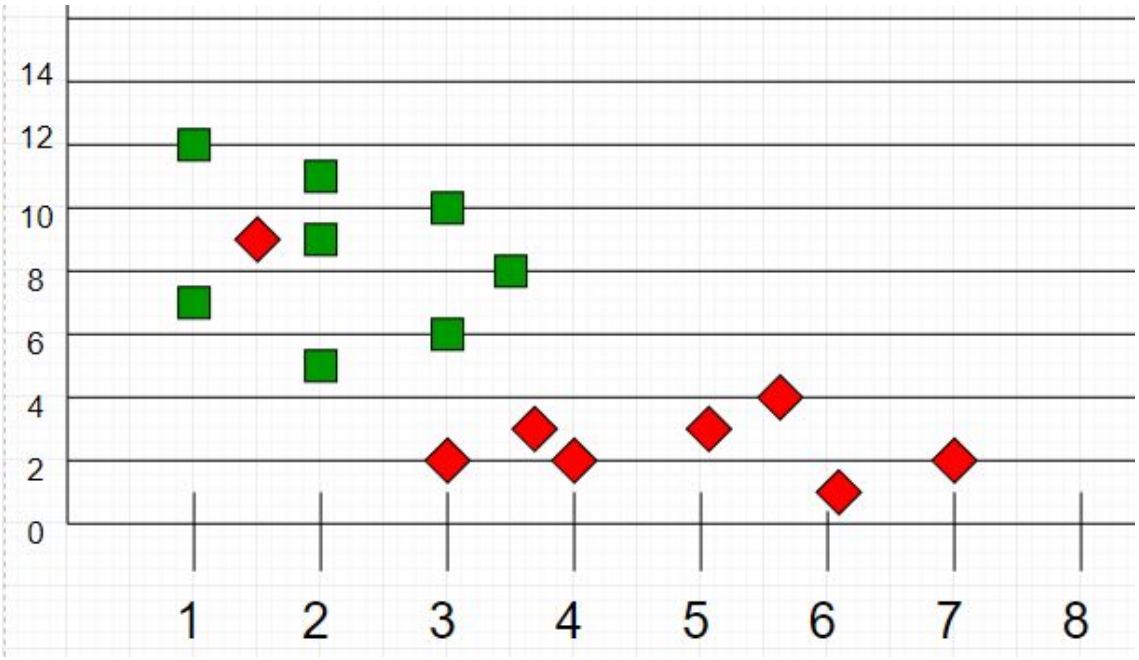
```
Gaussian Naive Bayes model accuracy(in %): 95.0
```

K-Nearest Neighbors

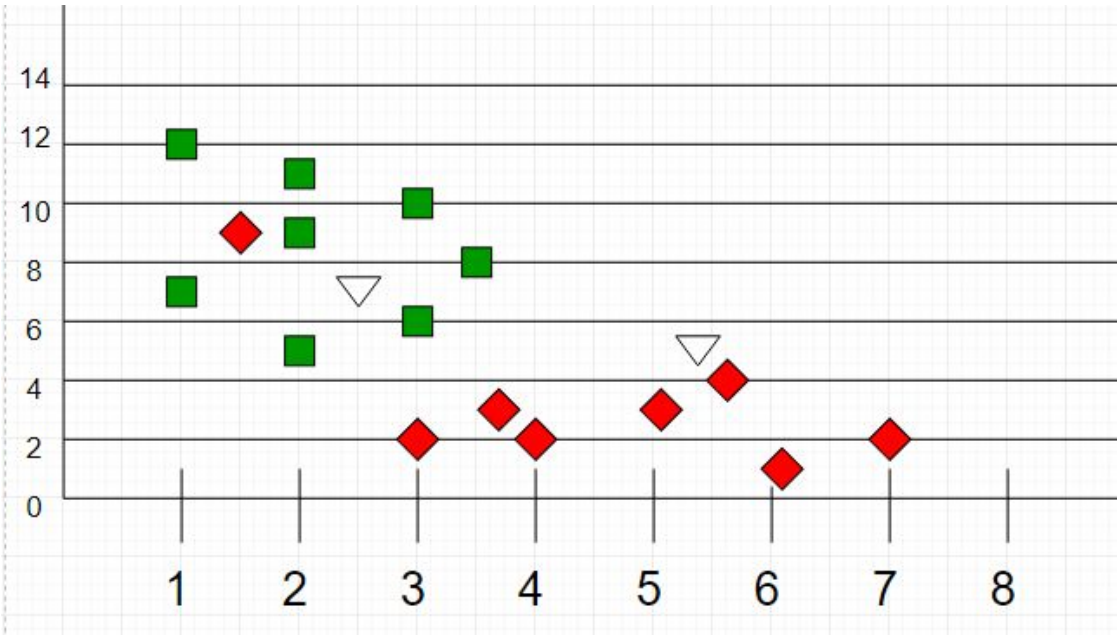
K-Nearest Neighbors, otherwise known as K-NN, is a critical yet simple classification algorithm. It is a supervised learning algorithm used for data mining, pattern detection, and intrusion detection.

In real-life scenarios, it is considered disposable because it is non-parametric, which means it makes no underlying assumptions about the data distribution. Some prior data, the training data, is provided, and this uses attributes to classify the coordinates into specific groups.

Here is an example showing a table with some data points and two features:



With another set of data points, called the testing data, we can analyze the data and allocate the points into a group. Unclassified points on the table are marked ‘White.”



Intuition

If those points were plotted on a graph, we might see some groups or clusters. Unclassified points can be assigned to a group just by seeing which group the unclassified point’s nearest

neighbors are in. That means a point near a cluster classified 'Red' is more likely to be classified as 'Red.'

Using intuition, it's clear that the first point (2.5, 7) should have a classification of 'Green' while the second one (5.5, 4.5) should be 'Red.'

Algorithm

Here, m indicates how many training samples we have, and p is an unknown point. Here's what we want to do:

1. The training samples are stored in arr[], which is an array containing data points. Each of the array elements represents (x, y), which is a tuple.

   For i=0 to m

2. Calculate d(arr[i], p), which is the Euclidean distance

3. Set S of K should be the smallest distances retrieved. Each distance is correspondent to a data point we already classified.

4. The majority label among S is returned.

We can keep K as an odd number, making it easier to work out the clear majority where there are only two possible groups. As K increases, the boundaries across the classifications become more defined and smoother. And, as the number of data points increases in the training set, the better the classifier accuracy becomes.

Here's an example program, written in C++ and Python, where 0 and 1 are the classifiers::

```
// C++ program to find groups of unknown
// Points using K nearest neighbor algorithm.
#include <bits/stdc++.h>
using namespace std;
```

```
struct Point
{
   int val;     // Group of point
   double x, y;     // Co-ordinate of point
   double distance; // Distance from test point
};


// Used to sort an array of points by increasing
// order of distance
bool comparison(Point a, Point b)
{
   return (a.distance < b.distance);
}


// This function finds classification of point p using
// k nearest neighbor algorithm. It assumes only two
// groups and returns 0 if p belongs to group 0, else
// 1 (belongs to group 1).
int classifyAPoint(Point arr[], int n, int k, Point p)
{
   // Fill distances of all points from p
   for (int i = 0; i < n; i++)
      arr[i].distance =
         sqrt((arr[i].x - p.x) * (arr[i].x - p.x) +
             (arr[i].y - p.y) * (arr[i].y - p.y));


   // Sort the Points by distance from p
   sort(arr, arr+n, comparison);


   // Now consider the first k elements and only
   // two groups
   int freq1 = 0;     // Frequency of group 0
   int freq2 = 0;     // Frequency of group 1
   for (int i = 0; i < k; i++)
   {
      if (arr[i].val == 0)
         freq1++;
```

```
        else if (arr[i].val == 1)
            freq2++;
    }

    return (freq1 > freq2 ? 0 : 1);
}

// Driver code
int main()
{
    int n = 17; // Number of data points
    Point arr[n];

    arr[0].x = 1;
    arr[0].y = 12;
    arr[0].val = 0;

    arr[1].x = 2;
    arr[1].y = 5;
    arr[1].val = 0;

    arr[2].x = 5;
    arr[2].y = 3;
    arr[2].val = 1;

    arr[3].x = 3;
    arr[3].y = 2;
    arr[3].val = 1;

    arr[4].x = 3;
    arr[4].y = 6;
    arr[4].val = 0;

    arr[5].x = 1.5;
```

```
arr[5].y = 9;
arr[5].val = 1;


arr[6].x = 7;
arr[6].y = 2;
arr[6].val = 1;


arr[7].x = 6;
arr[7].y = 1;
arr[7].val = 1;


arr[8].x = 3.8;
arr[8].y = 3;
arr[8].val = 1;


arr[9].x = 3;
arr[9].y = 10;
arr[9].val = 0;


arr[10].x = 5.6;
arr[10].y = 4;
arr[10].val = 1;


arr[11].x = 4;
arr[11].y = 2;
arr[11].val = 1;


arr[12].x = 3.5;
arr[12].y = 8;
arr[12].val = 0;


arr[13].x = 2;
arr[13].y = 11;
arr[13].val = 0;
```

```
    arr[14].x = 2;
    arr[14].y = 5;
    arr[14].val = 1;


    arr[15].x = 2;
    arr[15].y = 9;
    arr[15].val = 0;


    arr[16].x = 1;
    arr[16].y = 7;
    arr[16].val = 0;


    /*Testing Point*/
    Point p;
    p.x = 2.5;
    p.y = 7;


    // Parameter to decide group of the testing point
    int k = 3;
    printf ("The value classified to unknown point"
          " is %d.\n", classifyAPoint(arr, n, k, p));
    return 0;
}
```

And the output is:

The value classified to the unknown point is 0.

Support Vector Machine Learning Algorithm

Support vector machines or SVMs are grouped under the supervised learning algorithms used to analyze the data used in regression and classification analysis. The SVM is classed as a discriminative classifier, defined formally by a separating hyperplane. In simple terms, with labeled training data, the

supervised learning part, the output will be an optimal hyperplace that takes new examples and categorizes them.

SVMs represent examples as points in space. Each is mapped so that the separate categories have a wide, clear gap separating them. As well as being used for linear classification, an SVM can also be used for efficient non-linear classification, with the inputs implicitly mapped into high-dimensional feature spaces.

So, what does an SVM do?

When you provide an SVM algorithm with training examples, each one labeled as belonging to one of the two categories, it will build a model that will assign newly provided examples to one of the categories. This makes it a 'non-probabilistic binary linear classification.'

To illustrate this, we'll use an SVM model that uses ML tools, like Scikit-learn, to classify cancer UCI datasets. For this, you need to have NumPy, Pandas, Matplotlib, and Scikit-learn.

First, the dataset must be created:

```
# importing scikit learn with make_blobs
from sklearn.datasets.samples_generator import make_blobs


# creating datasets X containing n_samples
# Y containing two classes
X, Y = make_blobs(n_samples=500, centers=2,
                  random_state=0, cluster_std=0.40)
import matplotlib.pyplot as plt
# plotting scatters
plt.scatter(X[:, 0], X[:, 1], c=Y, s=50, cmap='spring');
plt.show()
```

Support vector machines don't just draw lines between classes. They also consider a region around the line of a specified width. Here's an example:

```
# creating line space between -1 to 3.5
```

```
xfit = np.linspace(-1, 3.5)


# plotting scatter
plt.scatter(X[:, 0], X[:, 1], c=Y, s=50, cmap='spring')


# plot a line between the different sets of data
for m, b, d in [(1, 0.65, 0.33), (0.5, 1.6, 0.55), (-0.2, 2.9, 0.2)]:
   yfit = m * xfit + b
   plt.plot(xfit, yfit, '-k')
   plt.fill_between(xfit, yfit - d, yfit + d, edgecolor='none',
   color='#AAAAAA', alpha=0.4)


plt.xlim(-1, 3.5);
plt.show()
```

Importing Datasets

The SVMs intuition is optimizing linear discriminant models representing the distance between the datasets, which is a perpendicular distance. Let's use our training data to train the model. Before we do that, the cancer datasets need to be imported as a CSV file, and we will train two of all the features:

```
# importing required libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt


# reading csv file and extracting class column to y.
x = pd.read_csv("C:\...\cancer.csv")
a = np.array(x)
y  = a[:,30] # classes having 0 and 1


# extracting two features
x = np.column_stack((x.malignant,x.benign))
```

```
# 569 samples and 2 features
x.shape


print (x),(y)
```

The output would be:

```
[[  122.8   1001.  ]
 [  132.9   1326.  ]
 [  130.    1203.  ]
...,
 [  108.3    858.1 ]
 [  140.1   1265.  ]
 [   47.92   181.  ]]

array([ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,
        0.,  0.,  0.,  0.,  0.,  0.,  1.,  1.,  1.,  0.,  0.,  0.,  0.,
        0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  1.,  0.,
        0.,  0.,  0.,  0.,  0.,  0.,  0.,  1.,  0.,  1.,  1.,  1.,  1.,
        1.,  0.,  0.,  1.,  0.,  0.,  1.,  1.,  1.,  1.,  0.,  1., ....,
        1.])
```

Fitting Support Vector Machines

Next, we need to fit the classifier to the points.

```
# import support vector classifier
# "Support Vector Classifier"
from sklearn.svm import SVC
clf = SVC(kernel='linear')


# fitting x samples and y classes
clf.fit(x, y)
```

Once the model is fitted, it can be used to make predictions on new values:

```
clf.predict([[120, 990]])


clf.predict([[85, 550]])
```

```
array([ 0.])
array([ 1.])
```

The data and the pre-processing methods are analyzed, and Matplotlib is used to produce optimal hyperplanes.

Linear Regression Machine Learning Algorithm

No doubt you have heard of linear regression, as it is one of the most popular machine learning algorithms. It is a statistical method used to model relationships between one dependent variable and a specified set of independent variables. For this section, the dependent variable is referred to as response and the independent variables as features.

Simple Linear Regression

This is the most basic form of linear regression and is used to predict responses from single features. Here, the assumption is that there is a linear relationship between two variables. As such, we want a linear function that can predict the response (y) value as accurately as it can as a function of the independent (x) variable or feature.

Let's look at a dataset with a response value for each feature:

x	0	1	2	3	4	5	6	7	8	9
y	1	3	2	5	7	8	8	9	10	12

We define the following for generality:

- x is defined as a feature vector, i.e. x = [x_1, x_2, …., x_n]
- y is defined as a response vector, i.e. y = [y_1, y_2, …., y_n]

These are defined for n observations, i.e. n=10.

The model's task is to fund the best fitting line to predict responses for new feature values, i.e., those features not in the

dataset already. The line is known as a regression line, and its equation is represented as:

$$h(x_i) = \beta_0 + \beta_1 x_i$$

Here:

- h(x_1) is representing the ith observation's predicted response
- b_0 and b_1 are both coefficients and are representing the y-intercept and the regression line slope, respectively.

Creating the model requires that we estimate or learn the coefficients' values, and once they have been estimated, the model can predict response.

We will be making use of the principle of Least Squares, so consider the following:

$$y_i = \beta_0 + \beta_1 x_i + \varepsilon_i = h(x_i) + \varepsilon_i \Rightarrow \varepsilon_i = y_i - h(x_i)$$

In this, e_i is the ith observation's residual error and we want to reduce the total residual error.

The cost function or squared error is defined as:

$$J(\beta_0, \beta_1) = \frac{1}{2n} \sum_{i=1}^{n} \varepsilon_i^2$$

We want to find the b_0 and b_1 values, where J(b_1, b_1) is the minimum.

Without drawing you through all the mathematical details, this is the result:

$$\beta_1 = \frac{SS_{xy}}{SS_{xx}}$$

$$\beta_0 = \bar{y} - \beta_1 \bar{x}$$

Here, SS_xy represents the sum of y and x (the cross deviations:

$$SS_{xy} = \sum_{i=1}^{n} (x_i - \bar{x})(y_i - \bar{y}) = \sum_{i=1}^{n} y_i x_i - n\bar{x}\bar{y}$$

While SS_xx is representing the sum of x, the squared deviations:

$$SS_{xx} = \sum_{i=1}^{n}(x_i - \bar{x})^2 = \sum_{i=1}^{n} x_i^2 - n(\bar{x})^2$$

Let's look at the Python implementation of our dataset:

```
import numpy as np
import matplotlib.pyplot as plt


def estimate_coef(x, y):
  # number of observations/points
  n = np.size(x)


  # mean of x and y vector
  m_x = np.mean(x)
  m_y = np.mean(y)


  # calculating cross-deviation and deviation about x
  SS_xy = np.sum(y*x) - n*m_y*m_x
  SS_xx = np.sum(x*x) - n*m_x*m_x


  # calculating regression coefficients
  b_1 = SS_xy / SS_xx
  b_0 = m_y - b_1*m_x


  return (b_0, b_1)


def plot_regression_line(x, y, b):
  # plotting the actual points as scatter plot
  plt.scatter(x, y, color = "m",
        marker = "o", s = 30)


  # predicted response vector
  y_pred = b[0] + b[1]*x
```

```
    # plotting the regression line
    plt.plot(x, y_pred, color = "g")


    # putting labels
    plt.xlabel('x')
    plt.ylabel('y')


    # function to show plot
    plt.show()


def main():
    # observations / data
    x = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
    y = np.array([1, 3, 2, 5, 7, 8, 8, 9, 10, 12])


    # estimating coefficients
    b = estimate_coef(x, y)
    print("Estimated coefficients:\nb_0 = {}  \
        \nb_1 = {}".format(b[0], b[1]))


    # plotting regression line
    plot_regression_line(x, y, b)


if __name__ == "__main__":
    main()
```

And the output is:

```
Estimated coefficients:
b_0 = -0.0586206896552
b_1 = 1.45747126437
```

Multiple Linear Regression

Multiple linear regression tries to model relationships between at least two features and one response, and to do this, a linear equation is fit to the observed data. It’s nothing more complex than an extension of simple linear regression.

Let's say we have a dataset containing p independent variables or features and one dependent variable or response. The dataset has n rows (observations).

We define the following:

- X (feature matrix) = a matrix of n X p size where x_{ij} is used to denote the values of the ith observation's jth feature.

So

$$\begin{pmatrix} x_{11} & \cdots & x_{1p} \\ x_{21} & \cdots & x_{2p} \\ \vdots & \ddots & \vdots \\ x_{n1} & \vdots & x_{np} \end{pmatrix}$$

And

- y (response vector) = a vector of n size, where y_{i} denotes the ith observation's response value

$$y = \begin{bmatrix} y_1 \\ y_2 \\ . \\ . \\ y_n \end{bmatrix}$$

The p features regression line is represented as:

$$h(x_i) = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + ... + \beta_p x_{ip}$$

Where h(x_i) is the ith observation's predicted response, and the regression coefficients are b_0, b_1, ⋯, b_p.

We can also write:

$$y_i = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + ... + \beta_p x_{ip} + \varepsilon_i$$
$$or$$
$$y_i = h(x_i) + \varepsilon_i \Rightarrow \varepsilon_i = y_i - h(x_i)$$

Where the ith observation's residual error is represented by e_i.

By presenting the matrix features as the following, the linear model can be generalized a bit further:

$$X = \begin{pmatrix} 1 & x_{11} & \cdots & x_{1p} \\ 1 & x_{21} & \cdots & x_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n1} & \cdots & x_{np} \end{pmatrix}$$

And the linear model can now be expressed as the following in terms of matrices:

$$y = X\beta + \varepsilon$$

where

$$\beta = \begin{bmatrix} \beta_0 \\ \beta_1 \\ . \\ . \\ \beta_p \end{bmatrix}$$

and

$$\varepsilon = \begin{bmatrix} \varepsilon_1 \\ \varepsilon_2 \\ . \\ . \\ \varepsilon_n \end{bmatrix}$$

The estimate of b can now be determined using the Least Squares method, which is used to determine b where the total residual errors are minimized. The result is:

$$\hat{\beta} = (X'X)^{-1}X'y$$

Where ' represents the matrix transpose, and the inverse is represented by -1.

Now that the least-squares estimate, b' is known, we can estimate the multiple linear regression models as:

$$\hat{y} = X\hat{\beta}$$

Where the estimated response vector is y.

Here's a Python example of multiple linear regression on the Boston house price dataset:

```
import matplotlib.pyplot as plt
import numpy as np
from sklearn import datasets, linear_model, metrics


# load the boston dataset
boston = datasets.load_boston(return_X_y=False)


# defining feature matrix(X) and response vector(y)
X = boston.data
y = boston.target


# splitting X and y into training and testing sets
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4,
random_state=1)


# create linear regression object
reg = linear_model.LinearRegression()


# train the model using the training sets
reg.fit(X_train, y_train)


# regression coefficients
print('Coefficients: ', reg.coef_)


# variance score: 1 means perfect prediction
print('Variance score: {}'.format(reg.score(X_test, y_test)))
```

```
# plot for residual error


## setting plot style
plt.style.use('fivethirtyeight')


## plotting residual errors in training data
plt.scatter(reg.predict(X_train), reg.predict(X_train) - y_train,
        color = "green", s = 10, label = 'Train data')


## plotting residual errors in test data
plt.scatter(reg.predict(X_test), reg.predict(X_test) - y_test,
        color = "blue", s = 10, label = 'Test data')


## plotting line for zero residual error
plt.hlines(y = 0, xmin = 0, xmax = 50, linewidth = 2)


## plotting legend
plt.legend(loc = 'upper right')


## plot title
plt.title("Residual errors")


## method call for showing the plot
plt.show()
```

And the output is:

Coefficients:

```
[ -8.80740828e-02   6.72507352e-02   5.10280463e-02
2.18879172e+00
-1.72283734e+01   3.62985243e+00   2.13933641e-03
-1.36531300e+00
2.88788067e-01  -1.22618657e-02  -8.36014969e-01   9.53058061e-
03
-5.05036163e-01]
```

Variance score: 0.720898784611

In this example, the Explained Variance Score is used to determine the accuracy score.

We defined:

explained_variance_score = 1 – Var{y – y'}/Var{y}

In this, the estimated target output is y', the correct target output is y, and the variance is VAR, which is the standard deviation square. 1.0 is the best score, while anything lower is worse.

Assumptions

These are the assumptions made by a linear regression model on the dataset it is applied to:

- **Linear relationship –** there should be a linear relationship between the response and the feature variables. Scatter plots can be used to test the linearity assumption.
- **Little to no multicollinearity** – this occurs when the independent variables are not independent of one another.
- **Little to no auto** - **correlation** – this occurs when there is no independence between the residual errors.
- **Homoscedasticity** – this is when the error term is the same across all the independent variables. This is defined as the random disturbance or noise in the relationship between the independent and dependent variables.

Logistic Regression Machine Learning Algorithm

Logistic regression is another machine learning technique from the field of statistics and is the go-to for solving binary classification, where there are two class values. It is named after the function at the very center of the method, known as the logistic function.

You may also know it as the sigmoid function. It was developed to describe the population growth properties in ecology, quickly rising and maxing out at the environment's carrying capacity. An S curve, it takes numbers (real values) and maps them to values between but not exactly at 0 and 1.

1 / (1 + e^-value)

E is the natural logarithm base, while value is the numerical value you want to be transformed. Let's see how the logistic function is used in logistic regression.

Representation

The representation used by logistic regression is an equation similar to linear regression. Input values, x, are linearly combined using coefficient values or weights to predict y, the output value. It differs from linear regression in that the modeled output value is binary, i.e., 0 or 1, and not a numeric value.

Here's an example of a logistic regression algorithm:

*y = e^(b0 + b1*x) / (1 + e^(b0 + b1*x))*

The predicted output is y, the intercept or bias is b0, and the coefficient for x, the input value, is b1. There is a b coefficient associated with each input data column. This coefficient is a constant real value that the model learns from the training data.

The coefficients you see in the equation are the actual model representation stored in a file or in memory.

Probability Prediction

Logistic regression is used to model the default class's probability. For example, let's say we are modeling sex (male or female) based on a person's height. In that case, the default class might be male, and the model is written for the probability of a male, given a certain height. More formally, that would be:

P(sex=male|height)

We could write this another way and say that the model is for the probability of X, an input belonging to Y=1, the default class. This could be written formally as:

$$P(X) = P(Y=1|X)$$

Be aware that the prediction needs to be transformed to binary values, 0 or 1, actually to make the prediction.

Logistic regression may be linear, but the logistic function is used to transform the predictions. This means that the predictions can no longer be understood as linear combinations of inputs like we do with linear regression. Continuing from earlier, we can state this model as:

$$p(X) = e^{(b0 + b1*X)} / (1 + e^{(b0 + b1*X)})$$

Without going too deeply into the math, the above equation can be turned around as follows, not forgetting that e can be removed from one side with the addition of ln, a natural logarithm, on the other side:

$$ln(p(X) / 1 - p(X)) = b0 + b1 * X$$

This is useful because it allows us to see that the output calculation on the right side is linear, while the left-side input is a log of the default class's probability.

The right-side ratio is known as the default class's odds rather than probability. We calculate odds as a ratio of the event's probability, divided by the probability of the event not happening, for example, 0.8/(1-0.8). The odds of this are 4, so we could write the following instead:

$$ln(odds) = b0 + b1 * X$$

As the odds are log-transformed, the left-side is called the probit or log-odds.

The exponent can be shifted to the right again and written as:

*odds = e^(b0 + b1 * X)*

This helps us see that our model is still linear in its output combination, but the combination relates to the default class's log-odds.

A Logistic Regression Model

You should use your training data to estimate the logistic regression algorithm's coefficients (Beta values b). This is done with the maximum-likelihood estimation. This is one of the more common learning algorithms that many other algorithms use, but it is worth noting that it makes assumptions about your data distribution.

The very best coefficients would give us a model that predicts values as close as possible to 1 for our default class and as close as possible to 0 for the other class. In this case, the maximum-likelihood's intuition for the logistic regression is that a search procedure seeks the coefficient's values that minimize the error in the predicted probabilities to the data. For example, if the data is the primary class, we get a probability of 1.

The math behind maximum-likelihood is out of the scope of this book. Still, it is sufficient to say that minimization algorithms are used to optimize the coefficient's best values for the training data.

Making Predictions with Logistic Regression

This is as easy as inputting the numbers to the equation and calculating your result. Here's an example to make this clearer.

Let's assume we have a model that uses height to predict whether a person is male or female – we'll use a base height of 150cm.

The coefficients have been learned – b0 = -100 and b1 = 1.6. Using the above equation, the probability is calculated that we have a male given a height of 150cm. More formally, this would be

P(male|height=150)

EXP() is used for e:

*y = e^(b0 + b1*X) / (1 + e^(b0 + b1*X))*

*y = exp(-100 + 0.6*150) / (1 + EXP(-100 + 0.6*X))*

y = 0.0000453978687

This gives us a near-zero probability of a person being male.

In practice, the probabilities can be used directly. However, we want a solid answer from this classification problem, so the probabilities can be snapped to binary class values, i.e.

0 if p(male) < 0.5

1 if p(male) >= 0.5

The next step is to prepare the data.

Prepare Data for Logistic Regression

Logistic regression makes assumptions about data distribution and relationships which are similar to those made by linear regression. Much work has gone into the definition of these assumptions, and they are written in precise statistical and probabilistic language. My advice? Use these as rules of thumb or guidelines, and don't be afraid to experiment with preparation schemes.

Here's an example of logistic regression in Python code:

We'll use the Pima Indian Diabetes dataset for this, which you can download from here . First, we load the data using the read CSV function in Pandas:

```
#import pandas
import pandas as pd
col_names = ['pregnant', 'glucose', 'bp', 'skin', 'insulin', 'bmi',
'pedigree', 'age', 'label']
```

```
# load dataset
pima = pd.read_csv("pima-indians-diabetes.csv", header=None,
names=col_names)
pima.head()
```

Next, we want to select the features. The columns can be divided into two – the dependent variable (the target) and the independent variables (the features.)

```
#split dataset in features and target variable
feature_cols = ['pregnant', 'insulin', 'bmi',
'age','glucose','bp','pedigree']
X = pima[feature_cols] # Features
y = pima.label # Target variable
```

The dataset must be divided into a training and test dataset. We do that with the train_test_split function, passing three parameters – features, target, test_set_size. You can also use random_state to choose records randomly:

```
# split X and y into training and testing sets
from sklearn.cross_validation import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.25,ran
dom_state=0)
```

/home/admin/.local/lib/python3.5/site-packages/sklearn/cross_validation.py:41: DeprecationWarning: This module was deprecated in version 0.18 in favor of the model_selection module into which all the refactored classes and functions are moved. Also, note that the interface of the new CV iterators is different from that of this module. This module will be removed in 0.20.

"This module will be removed in 0.20.", DeprecationWarning)

We used a ratio of 75:25 to break the dataset – 75% training data and 25% testing data.

Developing the Model and Making the Prediction

First, import the logistic regression model and use the LogisticRegression() function to create a classifier object. Then you can fit the model using fit() on the training set and use predict() on the test set to make the predictions:

```
# import the class
from sklearn.linear_model import LogisticRegression

# instantiate the model (using the default parameters)
logreg = LogisticRegression()

# fit the model with data
logreg.fit(X_train,y_train)

#
y_pred=logreg.predict(X_test)
```

Using the Confusion Matrix to Evaluate the Model

A confusion matrix is a table we use when we want the performance of our classification model evaluated. You can also use it to visualize an algorithm's performance. A confusion matrix shows a class-wise summing up of the correct and incorrect predictions:

```
# import the metrics class
from sklearn import metrics
cnf_matrix = metrics.confusion_matrix(y_test, y_pred)
cnf_matrix
array([[119,  11],
       [ 26,  36]])
```

The result is a confusion matrix as an array object, and the matrix dimension is 2*2 because this was a binary classification. There are two classes, 0 and 1. Accurate predictions are represented by the diagonal values, while the non-diagonals represent the inaccurate predictions. The output, in this case, is 119 and 36 as the actual predictions and 26 and 11 as the inaccurate predictions.

Using Heatmap to Visualize the Confusion Matrix

This uses Seaborn and Matplotlib to visualize the results in the confusion matrix – we'll be using Heatmap:

```
# import required modules
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
class_names=[0,1] # name  of classes
fig, ax = plt.subplots()
tick_marks = np.arange(len(class_names))
plt.xticks(tick_marks, class_names)
plt.yticks(tick_marks, class_names)
# create heatmap
sns.heatmap(pd.DataFrame(cnf_matrix), annot=True,
cmap="YlGnBu" ,fmt='g')
ax.xaxis.set_label_position("top")
plt.tight_layout()
plt.title('Confusion matrix', y=1.1)
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
Text(0.5,257.44,'Predicted label')
```

Evaluation Metrics

Now we can evaluate the model using precision, recall, and accuracy evaluation metrics:

```
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
print("Precision:",metrics.precision_score(y_test, y_pred))
print("Recall:",metrics.recall_score(y_test, y_pred))
```

The result is:

```
Accuracy: 0.8072916666666666
Precision: 0.7659574468085106
Recall: 0.5806451612903226
```

We got a classification accuracy of 80%, which is considered pretty good.

The precision metric is about precision, as you would expect. When your model makes predictions, you want to know how often it makes the right one. In our case, the model made a 76% prediction accuracy that people would have diabetes.

In terms of recall, the model can correctly identify those with diabetes in the test set 58% of the time.

ROC Curve

The ROC or Receiver Operating Characteristic curve plots the true against the false positive rate and shows the tradeoff between specificity and sensitivity.

```
y_pred_proba = logreg.predict_proba(X_test)[::,1]
fpr, tpr, _ = metrics.roc_curve(y_test,  y_pred_proba)
auc = metrics.roc_auc_score(y_test, y_pred_proba)
plt.plot(fpr,tpr,label="data 1, auc="+str(auc))
plt.legend(loc=4)
plt.show()
```

The AUC score is 0.86. A score of 0.5 indicates a poor classifier, while a score of 1 indicates a perfect classifier. 0.86 is as near to perfect as we can get.

Your ultimate focus in predictive modeling is making predictions that are as accurate as possible, rather than analyzing the results. As such, some assumptions can be broken, so long as you have a strong model that performs well:

- **Binary Output Variable** : you might think this is obvious, given that we mentioned it earlier. Logistic regression is designed for binary classification, and it will predict probabilities of instances belonging to the first or default class and snapping them to a 0 or 1 classification.

- **Removing Noise** : one assumption is that the output variable, y, has no error. You should remove outliers and potentially misclassified data from your training data right from the start.
- **Gaussian Distribution** : because logistic regression is linear but with the output being transformed non-linearly, an assumption is made of a linear relationship between the input and output variables. You get a more accurate model when you transform the input variables to expose the relationship better. For example, univariate transforms, such as Box-cox, log, and root, can do this.
- **Removing Correlated Inputs** : in the same way as linear regression, your model can end up overfitting if you have too many highly correlated inputs. Think about removing the highly correlated inputs and calculating pairwise correlations.
- **Failure to Converge** : it may happen that the likelihood estimation learning the coefficients doesn't converge. This could be because of too many highly correlated inputs in the data or sparse data, which means the input data contains many zeros.

Decision Tree Machine Learning Algorithm

The decision tree is a popular algorithm, undoubtedly one of the most popular, and it comes under supervised learning. It is used successfully with categorical and continuous output variables.

To show you how the decision tree algorithm works, we're going to walk through implementing one on the UCI database, Balance Scale Weight & Distance. The database contains 625 instances – 49 balanced and 288 each to the left and right. There are four numeric attributes and the class name.

Python Packages

You need three Python packages for this:

- **Sklearn –** a popular ML package containing many algorithms, and we'll be using some popular modules, such as train_test_split, accuracy_score, and DecisionTreeClassifier.
- **NumPy –** a numeric module containing useful math functions, NumPy is used to manipulate data and read data in NumPy arrays.
- **Pandas –** commonly used to read/write files and uses DataFrames to manipulate data.

Sklearn is where all the packages are that help us implement the algorithm, and this is installed with the following commands:

```
using pip :
pip install -U scikit-learn
```

Before you dive into this, ensure that NumPy and Scipy are installed and that you have pip installed too. If pip is not installed, use the following commands to get it:

```
python get-pip.py
```

If you are using conda, use this command:

```
conda install scikit-learn
```

Decision Tree Assumptions

The following are the assumptions made for this algorithm:

- The entire training set is considered as the root at the start of the process.
- It is assumed that the attributes are categorical for information gain and continuous for the Gini index.
- Records are recursively distributed, based on attribute values

- Statistical methods are used to order the attributes as internal node or root.

Pseudocode:

- The best attribute must be found and placed on the tree's root node
- The training dataset needs to be divided into subsets, ensuring that each one has the same attribute value
- Leaf nodes must be found in every branch by repeating steps one and two on every subset

The implementation of the algorithm requires the following two phases:

- **The Building Phase –** the dataset is preprocessed, then sklearn is used to split the dataset into test and train sets before training the classifier.
- **The Operational Phase –** predictions are made and the accuracy calculated

Importing the data

Data import and manipulation are done using the Pandas package. The URL we use fetches the dataset directly from the UCI website, so you don't need to download it at the start. That means you will need a good internet connection when you run the code. Because "," is used to separate the dataset, we need to pass that as the sep value parameter. Lastly, because there is no header in the dataset, the Header's parameter must be passed as none. If no header parameter is passed, the first line in the dataset will be used as the Header.

Slicing the Data

Before we train our model, the dataset must be split into two – the training and testing sets. To do this, the train_test_split

module from sklearn is used. First, the target variable must be split from the dataset attributes:

```
X = balance_data.values[:, 1:5]
Y = balance_data.values[:,0]
```

Those two lines of code will separate the dataset. X and Y are both variables – X stores the attributes, and Y stores the dataset's target variable. Next, we need to split the dataset into the training and testing sets:

```
X_train, X_test, y_train, y_test = train_test_split(
X, Y, test_size = 0.3, random_state = 100)
```

These code lines split the dataset into a ratio of 70:30 – 70% for training and 30% for testing. The parameter value for test_size is passed as 0.3. The variable called random_state is a pseudo-random number generator typically used in random sampling cases.

Code Terminology:

Both the information gain and Gini Index methods select the right attribute to place at the internal node or root node.

Gini Index:

- This is a metric used to measure how often elements chosen at random would be identified wrongly.
- Attributes with lower Gini Indexes are preferred
- Sklearn provides support for the "gini" criteria and takes "gini" value by default.

Entropy:

- This is a measure of a random variable's uncertainty, characterizing impurity in an arbitrary example collection.
- The higher entropy is, the more information it contains.

Information Gain:

- Typically, entropy will change when a node is used to divide training instances in a decision tree into smaller sets. Information gain is one measure of change.
- Sklearn has support for the Information Gain's "entropy" criteria. If we need the Information Gain method from sklearn, it must be explicitly mentioned.

Accuracy Score:

- This is used to calculate the trained classifier's accuracy score.

Confusion Matrix:

- This is used to help understand how the trained classifier behaves over the test dataset or help validate the dataset.

Let's get into the code:

```
# Run this program on your local python
# interpreter, provided you have installed
# the required libraries.


# Importing the required packages
import numpy as np
import pandas as pd
from sklearn.metrics import confusion_matrix
from sklearn.cross_validation import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report


# Function importing Dataset
def importdata():
  balance_data = pd.read_csv(
'https://archive.ics.uci.edu/ml/machine-learning-'+
```

```
'databases/balance-scale/balance-scale.data',
    sep= ',', header = None)


    # Printing the dataswet shape
    print ("Dataset Length: ", len(balance_data))
    print ("Dataset Shape: ", balance_data.shape)


    # Printing the dataset obseravtions
    print ("Dataset: ",balance_data.head())
    return balance_data


# Function to split the dataset
def splitdataset(balance_data):


    # Separating the target variable
    X = balance_data.values[:, 1:5]
    Y = balance_data.values[:, 0]


    # Splitting the dataset into train and test
    X_train, X_test, y_train, y_test = train_test_split(
    X, Y, test_size = 0.3, random_state = 100)


    return X, Y, X_train, X_test, y_train, y_test


# Function to perform training with giniIndex.
def train_using_gini(X_train, X_test, y_train):


    # Creating the classifier object
    clf_gini = DecisionTreeClassifier(criterion = "gini",
            random_state = 100,max_depth=3, min_samples_leaf=5)


    # Performing training
    clf_gini.fit(X_train, y_train)
```

```
    return clf_gini


# Function to perform training with entropy.
def tarin_using_entropy(X_train, X_test, y_train):


    # Decision tree with entropy
    clf_entropy = DecisionTreeClassifier(
            criterion = "entropy", random_state = 100,
            max_depth = 3, min_samples_leaf = 5)


    # Performing training
    clf_entropy.fit(X_train, y_train)
    return clf_entropy



# Function to make predictions
def prediction(X_test, clf_object):


    # Prediction on test with giniIndex
    y_pred = clf_object.predict(X_test)
    print("Predicted values:")
    print(y_pred)
    return y_pred


# Function to calculate accuracy
def cal_accuracy(y_test, y_pred):


    print("Confusion Matrix: ",
        confusion_matrix(y_test, y_pred))


    print ("Accuracy : ",
    accuracy_score(y_test,y_pred)*100)
```

```
    print("Report : ",
    classification_report(y_test, y_pred))


# Driver code
def main():


    # Building Phase
    data = importdata()
    X, Y, X_train, X_test, y_train, y_test = splitdataset(data)
    clf_gini = train_using_gini(X_train, X_test, y_train)
    clf_entropy = tarin_using_entropy(X_train, X_test, y_train)


    # Operational Phase
    print("Results Using Gini Index:")


    # Prediction using gini
    y_pred_gini = prediction(X_test, clf_gini)
    cal_accuracy(y_test, y_pred_gini)


    print("Results Using Entropy:")
    # Prediction using entropy
    y_pred_entropy = prediction(X_test, clf_entropy)
    cal_accuracy(y_test, y_pred_entropy)



# Calling main function
if __name__=="__main__":
    main()
```

Here's the data information:

```
Dataset Length:  625
Dataset Shape:  (625, 5)
Dataset:     0  1  2  3  4
0  B  1  1  1  1
1  R  1  1  1  2
2  R  1  1  1  3
```

3 R 1 1 1 4
4 R 1 1 1 5
Using the Gini Index, the results are:

Predicted values:

['R' 'L' 'R' 'R' 'R' 'L' 'R' 'L' 'L' 'L' 'R' 'L' 'L' 'L' 'R' 'L' 'R' 'L'
'L' 'R' 'L' 'R' 'L' 'L' 'R' 'L' 'L' 'L' 'R' 'L' 'L' 'L' 'R' 'L' 'L' 'L'
'L' 'R' 'L' 'L' 'R' 'L' 'R' 'L' 'R' 'R' 'L' 'L' 'R' 'L' 'R' 'R' 'L' 'R'
'R' 'L' 'R' 'R' 'L' 'L' 'R' 'R' 'L' 'L' 'L' 'L' 'L' 'R' 'R' 'L' 'L' 'R'
'R' 'L' 'R' 'L' 'R' 'R' 'R' 'L' 'R' 'L' 'L' 'L' 'L' 'R' 'R' 'L' 'R' 'L'
'R' 'R' 'L' 'L' 'L' 'R' 'R' 'L' 'L' 'L' 'R' 'L' 'R' 'R' 'R' 'R' 'R' 'R'
'R' 'L' 'R' 'L' 'R' 'R' 'L' 'R' 'R' 'R' 'R' 'R' 'L' 'R' 'L' 'L' 'L' 'L'
'L' 'L' 'L' 'R' 'R' 'R' 'R' 'L' 'R' 'R' 'R' 'L' 'L' 'R' 'L' 'R' 'L' 'R'
'L' 'L' 'R' 'L' 'L' 'R' 'L' 'R' 'L' 'R' 'R' 'R' 'L' 'R' 'R' 'R' 'R' 'R'
'L' 'L' 'R' 'R' 'R' 'R' 'L' 'R' 'R' 'R' 'L' 'R' 'L' 'L' 'L' 'L' 'R' 'R'
'L' 'R' 'R' 'L' 'L' 'R' 'R' 'R']

The confusion matrix is:

[[0 6 7]
[0 67 18]
[0 19 71]]

The accuracy is:

73.4042553191

The report:

	precision	recall	f1-score	support
B	0.00	0.00	0.00	13
L	0.73	0.79	0.76	85
R	0.74	0.79	0.76	90
avg/total	0.68	0.73	0.71	188

Using Entropy, the result are:

Predicted values:

['R' 'L' 'R' 'L' 'R' 'L' 'R' 'L' 'R' 'R' 'R' 'R' 'L' 'L' 'R' 'L' 'R' 'L'
'L' 'R' 'L' 'R' 'L' 'L' 'R' 'L' 'R' 'L' 'R' 'L' 'R' 'L' 'R' 'L' 'L' 'L'

```
'L' 'L' 'R' 'L' 'R' 'L' 'R' 'L' 'R' 'R' 'L' 'L' 'R' 'L' 'L' 'R' 'L' 'L'
'R' 'L' 'R' 'R' 'L' 'R' 'R' 'R' 'L' 'L' 'R' 'L' 'L' 'R' 'L' 'L' 'L' 'R'
'R' 'L' 'R' 'L' 'R' 'R' 'R' 'L' 'R' 'L' 'L' 'L' 'L' 'R' 'R' 'L' 'R' 'L'
'R' 'R' 'L' 'L' 'L' 'R' 'R' 'L' 'L' 'L' 'R' 'L' 'L' 'R' 'R' 'R' 'R' 'R'
'R' 'L' 'R' 'L' 'R' 'R' 'L' 'R' 'R' 'L' 'R' 'R' 'L' 'R' 'R' 'R' 'L' 'L'
'L' 'L' 'L' 'R' 'R' 'R' 'R' 'L' 'R' 'R' 'R' 'L' 'L' 'R' 'L' 'R' 'L' 'R'
'L' 'R' 'R' 'L' 'L' 'R' 'L' 'R' 'R' 'R' 'R' 'R' 'L' 'R' 'R' 'R' 'R' 'R'
'R' 'L' 'R' 'L' 'R' 'R' 'L' 'R' 'L' 'R' 'L' 'R' 'L' 'L' 'L' 'L' 'L' 'R'
'R' 'R' 'L' 'L' 'L' 'R' 'R' 'R']
```

The confusion matrix is:

```
[[ 0  6  7]
[ 0 63 22]
[ 0 20 70]]
```

The accuracy is

70.7446808511

And the report is:

	precision	recall	f1-score	support
B	0.00	0.00	0.00	13
L	0.71	0.74	0.72	85
R	0.71	0.78	0.74	90
avg / total	0.66	0.71	0.68	188

Random Forest Machine Learning Algorithm

The random forest classifier is another supervised machine learning algorithm that can be used for regression, classification, and other decision tree tasks. The random forest creates sets of decision trees from random training data subsets. It then makes its decision using votes from the trees.

To demonstrate how this works, we'll use the well-known iris flower dataset to train the model and test it. The model will be built to determine the flower classifications.

First, we load the dataset:

```
# importing required libraries
# importing Scikit-learn library and datasets package
from sklearn import datasets


# Loading the iris plants dataset (classification)
iris = datasets.load_iris()
```

Code: checking our dataset content and features names present in it.

```
print(iris.target_names)
```

Output:

['setosa' 'versicolor' 'virginica']

Code:

```
print(iris.feature_names)
```

Output:

['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']

Code:

```
# dividing the datasets into two parts i.e. training datasets and test datasets
X, y = datasets.load_iris( return_X_y = True)


# Splitting arrays or matrices into random train and test subsets
from sklearn.model_selection import train_test_split
# i.e. 80 % training dataset and 30 % test datasets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size =
0.70)
```

Code: Importing required libraries and random forest classifier module.

```
# importing random forest classifier from assemble module
from sklearn.ensemble import RandomForestClassifier
import pandas as pd
# creating dataframe of IRIS dataset
data = pd.DataFrame({'sepallength': iris.data[:, 0], 'sepalwidth':
iris.data[:, 1],
                     'petallength': iris.data[:, 2], 'petalwidth': iris.data[:, 3],
```

```
                'species': iris.target})
```
Code: Looking at a dataset

```
# printing the top 5 datasets in iris dataset
print(data.head())
```

The output of this is:

	sepallength	sepalwidth	petallength	petalwidth	species
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0

Next, we create the Random Forest Classifier, train it and then test it:

```
# creating a RF classifier
clf = RandomForestClassifier(n_estimators = 100)


# Training the model on the training dataset
# fit function is used to train the model using the training sets as
parameters
clf.fit(X_train, y_train)


# performing predictions on the test dataset
y_pred = clf.predict(X_test)


# metrics are used to find accuracy or error
from sklearn import metrics
print()


# using metrics module for accuracy calculation
print("ACCURACY OF THE MODEL: ", metrics.accuracy_score(y_test,
y_pred))
```

Output:

```
ACCURACY OF THE MODEL: 0.9238095238095239
```

Code: predicting the type of flower from the data set

```
# predicting which type of flower it is.
clf.predict([[3, 3, 2, 2]])
```

And the output is:

```
array([0])
```

The result tells us the flower classification is the setosa type. Note that the dataset contains three types – Setosa, Vericolor, and Virginia.

Next, we want to find the important features in the dataset:

```
# importing random forest classifier from assemble module
from sklearn.ensemble import RandomForestClassifier
# Create a Random forest Classifier
clf = RandomForestClassifier(n_estimators = 100)


# Train the model using the training sets
clf.fit(X_train, y_train)
Code: Calculating feature importance
# using the feature importance variable
import pandas as pd
feature_imp = pd.Series(clf.feature_importances_, index =
iris.feature_names).sort_values(ascending = False)
feature_imp
```

The output is:

```
petal width (cm)     0.458607
petal length (cm)    0.413859
sepal length (cm)    0.103600
sepal width (cm)     0.023933
dtype: float64
```

Each decision tree created by the random forest algorithm has a high variance. However, when they are all combined in parallel, the variance is low. That's because the individual trees are trained perfectly on the specified sample data, and the output is not dependent on just one tree but all of them. In classification problems, a majority voting classifier is used to get the final output. In regression problems, the output is the result of aggregation – the mean of all outputs.

Random forest uses the aggregation and bootstrap technique, also called bagging. The idea is to combine several decision trees to determine the output rather than relying on the individual ones.

The bootstrap part of the technique is where random row and feature sampling forms the sample datasets for each model. The regression technique must be approached as we would any ML technique:

- A specific data or question is designed, and the source is obtained to work out what data we need
- The data must be in an accessible format, or it must be converted to an accessible format
- All noticeable anomalies must be specified, along with missing data points that might be needed to get the right data
- The machine learning model is created, and a baseline model set that you want to be achieved
- The machine learning model is trained on the data
- The test data is used to test the model
- The performance metrics from the test and predicted data are compared

- If the model doesn't do what it should, work on improving the model, changing the data, or using a different modeling technique.
- At the end, the data is interpreted and reported.

Here's another example of how random forest works.

First, the libraries are imported:

```
# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
Next, the dataset is imported and printed:
data = pd.read_csv('Salaries.csv')
print(data)
```

Then all the rows and column 1 are selected from the dataset to x and every row and column 2 as y :

```
x = data.iloc[:, 1:2].values
print(x)
y = data.iloc[:, 2].values
```

The random forest aggressor is fitted to the dataset:

```
# Fitting Random Forest Regression to the dataset
# import the regressor
from sklearn.ensemble import RandomForestRegressor
# create regressor object
regressor = RandomForestRegressor(n_estimators = 100,
random_state = 0)


# fit the regressor with x and y data
regressor.fit(x, y)
```

The result is predicted:

```
Y_pred = regressor.predict(np.array([6.5]).reshape(1, 1))  # test the
output by changing values
```

The result is visualized:

```
# Visualizing the Random Forest Regression results


# arange for creating a range of values
# from min value of x to max
# value of x with a difference of 0.01
# between two consecutive values
X_grid = np.arange(min(x), max(x), 0.01)


# reshape for reshaping the data into a len(X_grid)*1 array,
# i.e. to make a column out of the X_grid value
X_grid = X_grid.reshape((len(X_grid), 1))


# Scatter plot for original data
plt.scatter(x, y, color = 'blue')


# plot predicted data
plt.plot(X_grid, regressor.predict(X_grid),
        color = 'green')
plt.title('Random Forest Regression')
plt.xlabel('Position level')
plt.ylabel('Salary')
plt.show()
```

Artificial Neural Networks Machine Learning Algorithm

Artificial Neural Networks, or ANNs, are paradigms to process information based on how the human brain works. Like a human, an ANN learns by example. They are configured for particular applications, like data classification or pattern recognition, through a learning process that mostly involves adjusting the connections between the neurons.

The human brain is filled with billions and billions of neurons connected to one another via synapses. These are the connections by which neurons send impulses to other neurons. When one neuron sends a signal to another, the signal gets added to the neuron's inputs. Once a specified threshold is exceeded, the target neuron fires a signal forward. That is pretty much how the internal thinking process happens.

In data science, this process is modeled by networks created on computers using matrices. We can understand these networks as neuron abstractions with none of the biological complexities. We could get very complex here, but we won't. We'll create a simple neural network that has two layers and can solve a linear classification problem

Let's say we want to predict the output, using specified inputs and outputs as training examples.

The training process is:

1. **Forward Propagation**

The inputs are multiplied by the weights:

$$Let\ Y = WiIi = W1I1+W2I2+W3I3$$

The result is passed through a sigmoid formula so the neuron's output can be calculated. This function normalizes the result so it falls between 0 and 1.

$$1/1 + e\text{-}y$$

2. **Back Propagation**

The error is calculated. This is the difference between the actual and expected output. Depending on what the error is, the weights can be adjusted. To do this, multiply the error by the input and then do it again with sigmoid curve gradient:

Weight += Error Input Output (1-Output) ,here Output (1-Output) is derivative of sigmoid curve.

The entire process must be repeated for however many iterations there are, potentially thousands.

Let's see what the whole thing looks like coded in Python, using NumPy to do the matrices calculations. Make sure NumPy is already installed and then go ahead with the following code:

```
from joblib.numpy_pickle_utils import xrange
from numpy import *


class NeuralNet(object):
  def __init__(self):
    # Generate random numbers
    random.seed(1)


    # Assign random weights to a 3 x 1 matrix,
    self.synaptic_weights = 2 * random.random((3, 1)) - 1


  # The Sigmoid function
  def __sigmoid(self, x):
    return 1 / (1 + exp(-x))


  # The derivative of the Sigmoid function.
  # This is the gradient of the Sigmoid curve.
  def __sigmoid_derivative(self, x):
    return x * (1 - x)


  # Train the neural network and adjust the weights each time.
  def train(self, inputs, outputs, training_iterations):
    for iteration in xrange(training_iterations):
      # Pass the training set through the network.
      output = self.learn(inputs)
```

```
            # Calculate the error
            error = outputs - output


            # Adjust the weights by a factor
            factor = dot(inputs.T, error *
self.__sigmoid_derivative(output))
            self.synaptic_weights += factor


        # The neural network thinks.


    def learn(self, inputs):
        return self.__sigmoid(dot(inputs, self.synaptic_weights))



if __name__ == "__main__":
    # Initialize
    neural_network = NeuralNet()


    # The training set.
    inputs = array([[0, 1, 1], [1, 0, 0], [1, 0, 1]])
    outputs = array([[1, 0, 1]]).T


    # Train the neural network
    neural_network.train(inputs, outputs, 10000)


    # Test the neural network with a test example.
    print(neural_network.learn(array([1, 0, 1])))
```

The Output

Once 10 iterations have been completed, the NN should predict a value of 0.6598.921. This isn't what we want as the answer should be 1. If the number of iterations is increased to 100, the output is 0.87680541. That means the network is improving the more

iterations it does. So, for 10,000 iterations, we get as near as we can with an output of 0.9897704.

Machine Learning Steps

If only machine learning were as simple as just applying an algorithm to the data and obtaining the predictions. Sadly, it isn't, and it involves many more steps for each project you do:

Step One – Gather the Data

This is the first and most important step in machine learning and is by far the most time-consuming. It involves collecting all the data needed to solve the problem and enough of it. For example, let's say we wanted to build a model that predicted house prices. In that case, we need a dataset with information about previous sales, which we can use to form a tabular structure. We'll solve something similar when we show you how to implement a regression problem later.

Step Two – Preparing the Data

When we have the data, it needs to be put into the right format and pre-processed. Pre-processing involves different steps, including cleaning the data. Should there be abnormal values, such as strings instead of numbers, or empty values, you need to understand how you will deal with them. The simplest way is to drop those rows containing empty values, although there are a few other ways you can do it. Sometimes, you may also have columns that will have no bearing on the results; those columns can also be removed. Usually, data visualization will be done to see the data in the form of charts, graphs, and diagrams. Once these have been analyzed, we can determine the important features.

Step Three – Choosing Your Model

Now the data is ready, and we need a machine-learning algorithm to feed it to. Some people use "machine learning algorithm"

interchangeably with “machine learning model,” but they are not the same. The model is the algorithm’s output. When the algorithm is implemented on the data, we get an output containing the numbers, rules, and other data structures related to the algorithm and needed to make the predictions. For example, we would get an equation showing the best-fit line if we did linear regression on the data. That equation is the model. If we didn’t go down the route of tuning the hyperparameters, your next step would be training the model.

Step Four – Tuning the Hyperparameters

Hyperparameters are critical because they are responsible for controlling how a machine learning model behaves. The goal is to find the best hyperparameter combination to give us the best results. But what are they? Take the K-NN algorithm and the K variable. When K is given different values, the results are predictably different. K’s best value isn’t defined, and each dataset will have its own value. There is also no easy way of knowing what the best values are; all you can do is try different ones and see what results you get. In this case, K is a hyperparameter, and its values need to be tuned to get the right results.

Step Five – Evaluating the Model

Next, we want to know how well the model has performed. The best way to know that is to test it on more data. This is the testing data, and it must be completely different from the data the algorithm was trained on. When you train a model, you do not intend it to learn the values in the training data. Instead, you want it to learn the data’s underlying pattern and use that to make predictions on previously unseen data. We’ll discuss this more in the next section.

Step Six – Making Predictions

The final step is to use the model on real-world problems in the hopes that it will work as intended.

Evaluating a Machine Learning Model

Training a machine learning model is an important step in machine learning. Equally important is evaluating how the model works on previously unseen data and is something you should consider with every pipeline you build. You need to know if your model works and, if it does, whether its predictions can be trusted or not. Is the model memorizing the data you feed it, or is it learning by example? If it is only memorizing data, its predictions on unseen data will be next to useless.

We're going to take some time out to look at some of the techniques you can use to evaluate your model to see how well it works. We'll also look at some of the common evaluation metrics for regression and classification methods with Python.

Model Evaluation Techniques

Handling how your machine learning model performs requires evaluation, an integral part of any data science project you are involved in. When you evaluate the model, you aim to estimate how accurate the model's generalization is on unseen data.

Model evaluation methods fall into two distinct categories – cross-validation and holdout. Both use test sets, which are datasets of completely new, previously unseen data, to evaluate the performance. As you already know by now, using the same data you train a model on to test it is not recommended because the model will not learn anything – it will simply memorize what you trained it on. The predictions will always be correct anywhere in the training set. This is called overfitting.

Holdout

Holdout evaluation tests models on different data to the training data, providing an unbiased learning performance estimate. The dataset will be divided into three random subsets:

- **The training set –** this is a subset of the full dataset using in predictive model building

- **The validation set –** this subset of the dataset is used to assess the model's performance. It gives us a test platform that allows us to fine-tune the parameters for our model and choose the model that performs the best. Be aware that you won't need a validation set for every single algorithm.
- **The test set –** this is the unseen data, a subset of the primary dataset used to tell us how the model may perform in the future. If you find the model fitting the training set better than it does the test set, you probably have a case of overfitting on your hands.

The holdout technique is useful because it is simple, flexible, and fast. However, it does come with an association to high variance. This is because the differences between the training and set datasets can provide significant differences in terms of the accuracy estimation.

Cross-Validation

The cross-validation technique will partition the primary dataset into two – a training set to train the model and another independent set for evaluating the analysis. Perhaps the best-known cross-validation technique is k-fold, which partitions the primary dataset into k number of subsamples, equal in size. Each subsample is called a fold, and the k defines a specified number – typically, 5 or 10 are the preferred options. This process is repeated k number of times, and each time a k subset is used to validate, while the remaining subsets are combined into a training set. The model effectiveness is drawn from an average estimation of all the k trials.

For example, when you perform ten-fold cross-validation, the data is split into ten parts, roughly equal in size. Models are then trained in sequence – the first fold is used as the test set, while the other nine are used for training. This is repeated for each part,

and accuracy estimation is averaged from all ten trials to see how effective the model is.

Each data point is used as the test set once and is part of the training set k number of times, in our case, nine. The reduces the bias significantly as almost all the data is used for fitting. It also reduces variance significantly because almost all the data is used in the test set. This method's effectiveness is increased because the test and training sets are interchanged.

Model Evaluation Metrics

Quantifying the model performance requires the use of evaluation metrics, and which ones you use depend on what kind of task you are doing – regression, classification, clustering, and so on. Some metrics are good for multiple tasks, such as precision-recall. Classification, regression, and other supervised learning tasks make up the largest number of machine learning applications, so we will concentrate on regression and classification

Classification Metrics

We'll take a quick look at the most common metrics for classification problems, including:

- Classification Accuracy
- Confusion matrix
- Logarithmic Loss
- AUC – Area Under Curve
- F-Measure

Classification Accuracy

This is one of the more common metrics used in classification and indicates the correct predictions as a ratio of all predictions. The sklearn module is used for computing classification accuracy, as in the example below:

```
#import modules
import warnings
import pandas as pd
import numpy as np
from sklearn import model_selection
from sklearn.linear_model import LogisticRegression
from sklearn import datasets
from sklearn.metrics import accuracy_score
#ignore warnings
warnings.filterwarnings('ignore')
# Load digits dataset
iris = datasets.load_iris()
# # Create feature matrix
X = iris.data
# Create target vector
y = iris.target
#test size
test_size = 0.33
#generate the same set of random numbers
seed = 7
#cross-validation settings
kfold = model_selection.KFold(n_splits=10, random_state=seed)
#Model instance
model = LogisticRegression()
#Evaluate model performance
scoring = 'accuracy'
results = model_selection.cross_val_score(model, X, y, cv=kfold,
scoring=scoring)
print('Accuracy -val set: %.2f%% (%.2f)' % (results.mean()*100,
results.std()))

#split data
X_train, X_test, y_train, y_test = model_selection.train_test_split(X,
y, test_size=test_size, random_state=seed)
#fit model
model.fit(X_train, y_train)
#accuracy on test set
result = model.score(X_test, y_test)
print("Accuracy - test set: %.2f%%" % (result*100.0))
```

In this case, the accuracy is 88%.

Cross-validation allows us to test the model while training, checking for overfitting, and looking to see how the model will generalize on the test data. Cross-validation can also compare how different models perform on the same data set and helps us choose the best parameter values to ensure the maximum possible model accuracy. This is also known as parameter tuning.

Confusion Matrix

Confusion matrices provide more detail in the breakdown of right and wrong classifications for each individual class. In this case, we will use the Iris dataset for classification and computation of the confusion matrix:

```
#import modules
import warnings
import pandas as pd
from sklearn import model_selection
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt
%matplotlib inline

#ignore warnings
warnings.filterwarnings('ignore')
# Load digits dataset
url = "http://archive.ics.uci.edu/ml/machine-learning-
databases/iris/iris.data"
df = pd.read_csv(url)
# df = df.values
X = df.iloc[:,0:4]
y = df.iloc[:,4]
# print (y.unique())
#test size
test_size = 0.33
#generate the same set of random numbers
seed = 7
#Split data into train and test set.
X_train, X_test, y_train, y_test = model_selection.train_test_split(X,
y, test_size=test_size, random_state=seed)
#Train Model
```

```
model = LogisticRegression()
model.fit(X_train, y_train)
pred = model.predict(X_test)

#Construct the Confusion Matrix
labels = ['Iris-setosa', 'Iris-versicolor', 'Iris-virginica']
cm = confusion_matrix(y_test, pred, labels)
print(cm)
fig = plt.figure()
ax = fig.add_subplot(111)
cax = ax.matshow(cm)
plt.title('Confusion matrix of the classifier')
fig.colorbar(cax)
ax.set_xticklabels([''] + labels)
ax.set_yticklabels([''] + labels)
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()
```

The short way of explaining how a confusion matrix should be interpreted is this:

The elements on the diagonal represent how many points for which the predicted label and true label are equal. Anything not on the diagonal has been mislabeled. The higher the values on the diagonal, the better because this indicates a higher number of correct predictions.

Our classifier was correct in predicting all 13 Setosa and 18 Virginica iris plants in the test data, but it was wrong in classifying four Versicolor plants as Virginica.

Logarithmic Loss

Logarithmic loss is also called logloss, and it is used to measure a classification model's performance where a probability value between 0 and 1 is used as the prediction input. With a divergence between the predicted probability and the actual label, we saw an increase in log loss. That value needs to be

minimized by the machine learning model – the smaller the logloss, the better, and the best models have a log loss of 0.

```
#Classification LogLoss
import warnings
import pandas
from sklearn import model_selection
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import log_loss

warnings.filterwarnings('ignore')
url =
"https://raw.githubusercontent.com/jbrownlee/Datasets/master/p
ima-indians-diabetes.data.csv"
dataframe = pandas.read_csv(url)
dat = dataframe.values
X = dat[:,:-1]
y = dat[:,-1]
seed = 7
#split data
X_train, X_test, y_train, y_test = model_selection.train_test_split(X,
y, test_size=test_size, random_state=seed)
model.fit(X_train, y_train)
#predict and compute logloss
pred = model.predict(X_test)
accuracy = log_loss(y_test, pred)
print("Logloss: %.2f" % (accuracy))
```

Area Under Curve (AUC)

This performance metric measures how well binary classifiers discriminate between positive classes and negative ones.:

```
#Classification Area under curve
import warnings
import pandas
from sklearn import model_selection
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_auc_score, roc_curve

warnings.filterwarnings('ignore')
```

```
url =
"https://raw.githubusercontent.com/jbrownlee/Datasets/master/p
ima-indians-diabetes.data.csv"
dataframe = pandas.read_csv(url)
dat = dataframe.values
X = dat[:,:-1]
y = dat[:,-1]
seed = 7
#split data
X_train, X_test, y_train, y_test = model_selection.train_test_split(X,
y, test_size=test_size, random_state=seed)
model.fit(X_train, y_train)

# predict probabilities
probs = model.predict_proba(X_test)
# keep probabilities for the positive outcome only
probs = probs[:, 1]

auc = roc_auc_score(y_test, probs)
print('AUC - Test Set: %.2f%%' % (auc*100))

# calculate roc curve
fpr, tpr, thresholds = roc_curve(y_test, probs)
# plot no skill
plt.plot([0, 1], [0, 1], linestyle='--')
# plot the roc curve for the model
plt.plot(fpr, tpr, marker='.')
# show the plot
plt.show()
```

In this example, we have an AUC greater than 0.5 and close to 1. The perfect classifier has a ROC curve along the y-axis and then along the x-axis.

F-Measure

Also called F-Score, this measures the accuracy of a test, using the test's recall and precision to come up with the score. Precision indicates correct positive results divided by the total number of predicted positive observations. Recall indicates correct positive

results divided by the total actual positives or the total of all the relevant samples:

```
import warnings
import pandas
from sklearn import model_selection
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import log_loss
from sklearn.metrics import precision_recall_fscore_support as
score, precision_score, recall_score, f1_score

warnings.filterwarnings('ignore')

url =
"https://raw.githubusercontent.com/jbrownlee/Datasets/master/p
ima-indians-diabetes.data.csv"
dataframe = pandas.read_csv(url)
dat = dataframe.values
X = dat[:,:-1]
y = dat[:,-1]
test_size = 0.33
seed = 7

model = LogisticRegression()
#split data
X_train, X_test, y_train, y_test = model_selection.train_test_split(X,
y, test_size=test_size, random_state=seed)
model.fit(X_train, y_train)
precision = precision_score(y_test, pred)
print('Precision: %f' % precision)
# recall: tp / (tp + fn)
recall = recall_score(y_test, pred)
print('Recall: %f' % recall)
# f1: tp / (tp + fp + fn)
f1 = f1_score(y_test, pred)
print('F1 score: %f' % f1)
```

Regression Metrics

There are two primary metrics used to evaluate regression problems – Root Mean Squared Error and Mean Absolute Error.

Mean Absolute Error, also called MAE, tells us the sum of the absolute differences between the actual and predicted values. Root Mean Squared Error, also called RMSE, measures an average of the error magnitude by "taking the square root of the average of squared differences between prediction and actual observation."

Here's how we implement both metrics:

```
import pandas
from sklearn import model_selection
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error,
mean_squared_error
from math import sqrt
url =
"https://raw.githubusercontent.com/jbrownlee/Datasets/master/h
ousing.data"
dataframe = pandas.read_csv(url, delim_whitespace=True)
df = dataframe.values
X = df[:,:-1]
y = df[:,-1]
seed = 7
model = LinearRegression()
#split data
X_train, X_test, y_train, y_test = model_selection.train_test_split(X,
y, test_size=test_size, random_state=seed)
model.fit(X_train, y_train)
#predict
pred = model.predict(X_test)
print("MAE test score:", mean_absolute_error(y_test, pred))
print("RMSE test score:", sqrt(mean_squared_error(y_test, pred)))
```

In an ideal world, a model's estimated performance will tell us if the model performs well on unseen data or not. Most of the time, we will be building models to predict on future data and, before a metric is chosen, you must understand the context – each model will use different datasets and different objectives to solve the problem.

Implementing Machine Learning Algorithms with Python

By now, you should already have all the software and libraries you need on your machine – we will mostly be using Scikit-learn to solve two problems – regression and classification.

Implementing a Regression Problem

We want to solve the problem of predicting house prices given specific features, like the house size, how many rooms there are, etc.

- **Gather the Data –** we don't need to collect the required data manually because someone has already done it for us. All we need to do is import that dataset. It's worth noting that not all available datasets are free, but the ones we are using are.

We want the Boston Housing dataset, which contains records describing suburbs or towns in Boston. The data came from the Boston SMSA (Standard Metropolitan Statistical Area) in 1970, and the attributes have been defined as:

1. CRIM: the per capita crime rate in each town
2. ZN: the proportion of residential land zoned for lots of more than 25,000 sq.ft.
3. INDUS: the proportion of non-retail business acres in each town
4. CHAS: the Charles River dummy variable (= 1 if the tract bounds river; 0 if it doesn't)
5. NOX: the nitric oxide concentrations in parts per 10 million
6. RM: the average number of rooms in each property
7. AGE: the proportion of owner-occupied units built before 1940

8. DIS: the weighted distances to five employment centers in Boston
9. RAD: the index of accessibility to radial highways
10. TAX: the full-value property tax rate per $10,000
11. PTRATIO: the pupil-teacher ratio by town
12. B: 1000(Bk−0.63)2 where Bk is the proportion of black people in each town
13. LSTAT: % lower status of the population
14. MEDV: Median value of owner-occupied homes in $1000s

You can download the dataset here .

Download the file and open it to see the data on the house sales. Note that the dataset is not in tabular forms, and there are no names for the columns. Commas separate all the values.

The first step is to place the data in tabular form, and Pandas will help us do that. We give Pandas a list with all the column names and a delimiter of '\s+,' meaning that every entry can be differentiated when single or multiple spaces are encountered.

We'll start by importing the libraries we need and then the dataset in CSV format. All this is imported into a Pandas DataFrame.

```
import numpy as np
import pandas as pd
column_names = ['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE',
'DIS', 'RAD', 'TAX','PTRATIO', 'B', 'LSTAT', 'MEDV']
bos1 = pd.read_csv('housing.csv', delimiter=r"\s+",
names=column_names)
```

- **Preprocess the Data**

Next, the data must be pre-processed. When we look at the dataset, we can immediately see that it doesn't contain any NaN values. We also see that the data is in numbers, not strings, so we don't need to worry about errors when we train the model.

We'll divide the data into two – 70% for training and 30% for testing.

```
bos1.isna().sum()

from sklearn.model_selection import train_test_split
X=np.array(bos1.iloc[:,0:13])
Y=np.array(bos1["MEDV"])
#testing data size is of 30% of entire data
x_train, x_test, y_train, y_test =train_test_split(X,Y, test_size = 0.30,
random_state =5)
```

- **Choose Your Model**

To solve this problem, we want two supervised learning algorithms for solving regression. Later, the results from both will be compared. The first is K-NN and the second is linear regression, both of which were demonstrated earlier in this part.

```
from sklearn.linear_model import LinearRegression
from sklearn.neighbors import KNeighborsRegressor
#load our first model
lr = LinearRegression()
#train the model on training data
lr.fit(x_train,y_train)
#predict the testing data so that we can later evaluate the model
pred_lr = lr.predict(x_test)
#load the second model
Nn=KNeighborsRegressor(3)
Nn.fit(x_train,y_train)
pred_Nn = Nn.predict(x_test)
```

- **Tune the Hyperparameters**

All we are going to do here is tune the ok K value in K-NN – we could do more, but this is just for starters, so I don't want you to

get too overwhelmed. We'll use a for loop and check on the results from k, from 1 to 50. K-NN works fast on small datasets, so this won't take too much time.

```
import sklearn
for i in range(1,50):
   model=KNeighborsRegressor(i)
   model.fit(x_train,y_train)
   pred_y = model.predict(x_test)
   mse = sklearn.metrics.mean_squared_error(y_test,
pred_y,squared=False)
   print("{} error for k = {}".format(mse,i))
```

The output will be a list of errors for k = 1 right through to k = 50. The least error is for k = 3, justifying why k = 3 was used when the model was being trained.

- **Evaluate the Model**

We will use MSE (mean_squared_error) method in Scikit-learn to evaluate our model. The 'squared' parameter must be set as False, otherwise you won't get the RMSE error:

```
#error for linear regression
mse_lr= sklearn.metrics.mean_squared_error(y_test,
pred_lr,squared=False)
print("error for Linear Regression = {}".format(mse_lr))
#error for linear regression
mse_Nn= sklearn.metrics.mean_squared_error(y_test,
pred_Nn,squared=False)
print("error for K-NN = {}".format(mse_Nn))
```

The output is:

```
error for Linear Regression = 4.627293724648145
error for K-NN = 6.173717334583693
```

We can conclude from this that Linear Regression is much better than K-NN on this dataset at any rate. This won't always be the case as it is dependent on what data you are working with.

- **Make the Prediction**

At last, we can predict the house prices using our models and the predict function. Ensure that you get all the features that were in the data you used to train the model.

Here's the entire program from start to finish:

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.neighbors import KNeighborsRegressor
column_names = ['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE',
'DIS', 'RAD', 'TAX', 'PTRATIO', 'B', 'LSTAT', 'MEDV']
bos1 = pd.read_csv('housing.csv', delimiter=r"\s+",
names=column_names)
X=np.array(bos1.iloc[:,0:13])
Y=np.array(bos1["MEDV"])
#testing data size is of 30% of entire data
x_train, x_test, y_train, y_test =train_test_split(X,Y, test_size = 0.30,
random_state =54)
#load our first model
lr = LinearRegression()
#train the model on training data
lr.fit(x_train,y_train)
#predict the testing data so that we can later evaluate the model
pred_lr = lr.predict(x_test)
#load the second model
Nn=KNeighborsRegressor(12)
Nn.fit(x_train,y_train)
pred_Nn = Nn.predict(x_test)
#error for linear regression
mse_lr= sklearn.metrics.mean_squared_error(y_test,
pred_lr,squared=False)
print("error for Linear Regression = {}".format(mse_lr))
#error for linear regression
mse_Nn= sklearn.metrics.mean_squared_error(y_test,
pred_Nn,squared=False)
print("error for K-NN = {}".format(mse_Nn))
```

Implementing a Classification Problem

The problem we are solving here is the Iris population classification problem. The Iris dataset is one of the most popular and common for beginners. It has 50 samples of each of three Iris species, along with other properties of the flowers. One is linearly separable from two species, but those two are not linearly separable from one another. This dataset has the following columns:

Different species of iris

- SepalLengthCm
- SepalWidthCm
- PetalLengthCm
- PetalWidthCm
- Species

This dataset is already included in Scikit-learn, so all we need to do is import it:

```
from sklearn.datasets import load_iris
iris = load_iris()
X=iris.data
Y=iris.target
print(X)
print(Y)
```

The output is a list of features with four items – these are the features. The bottom part is a list of labels, all transformed to numbers because the model is unable to understand strings – each name must be coded as a number.

Here's the whole thing:

```
from sklearn.model_selection import train_test_split
#testing data size is of 30% of entire data
x_train, x_test, y_train, y_test =train_test_split(X,Y, test_size = 0.3,
random_state =5)
```

```
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
#fitting our model to train and test
Nn = KNeighborsClassifier(8)
Nn.fit(x_train,y_train)
#the score() method calculates the accuracy of model.
print("Accuracy for K-NN is ",Nn.score(x_test,y_test))
Lr = LogisticRegression()
Lr.fit(x_train,y_train)
print("Accuracy for Logistic Regression is ",Lr.score(x_test,y_test))
```

The output is:

```
Accuracy for K-NN is 1.0
Accuracy for Logistic Regression  is 0.9777777777777777
```

Advantages and Disadvantages of Machine Learning

Everything has its advantages and disadvantages, and machine learning is no different:

Advantages

- **Makes Identifying Trends and Patterns in the Data Easy**

With machine learning, huge amounts of data can be reviewed, looking for trends and patterns the human eye wouldn't see. For example, Flipkart, Amazon, and other e-commerce websites use machine learning to help understand how their customers browse and what they buy to send them relevant deals, product details, and reminders. The machine learning results are used to show relevant ads to each customer based on their browsing and purchasing history.

- **Continuous Improvement**

New data is continuously being generated, and providing machine learning models with the new data helps them upgrade over time, become more accurate, and continuously improve

their performance. This leads to continually better decisions being made.

- **They can Handle Multi-Variety and Multidimensional Data**

Machine learning algorithms can handle multi-variety and multidimensional data easily in uncertain or dynamic environments.

- **They are Widely Used**

It doesn't matter what industry you are in, machine learning can work for you. Wherever it is applied, it can give businesses the means and answers they need to make better decisions and provide their customers with a more personal user experience.

Disadvantages

- **It Needs Huge Amounts of Data**

Machine learning models need vast amounts of data to learn, but the quantity of day is only a small part. It must be high-quality, unbiased data, and there must be enough of it. At times, you may even need to wait until new data has been generated before you can train the model.

- **It Takes Time and Resources**

Machin learning needs sufficient time for the algorithms to learn and develop to ensure they can do their job with a significant level of relevancy and accuracy. They also require a high level of resources which can mean using more computer power than you might have available.

- **Interpretation of Results**

One more challenge is being able to interpret the results that the algorithms' output accurately. You must also carefully choose the algorithms based on the task at hand and what it is intended to do. You won't always choose the right algorithm for the first time, even though your analysis points you to a specific one.

- **Susceptible to Errors**

Although machine learning is autonomous, it does have a high susceptibility to errors. Let's say you train an algorithm with such small datasets that they are not inclusive. The result would be biased predictions because your training set is biased. This would lead to a business showing their customers' irrelevant ads or purchase choices. With machine learning, this type of thing can kick-start a whole list of errors that may not be detected for some time and, when they are finally detected, its time consuming to find where the issues started and put it right.

Conclusion

Thank you for taking the time to read my guide. Data science is one of the most used buzzwords of the current time, which will not change. With the sheer amount of data being generated daily, companies rely on data science to help them move their business forward, interact intelligently with their customers, and ensure they provide exactly that their customers need when they need it.

Machine learning is another buzzword, a subset of data science that has only really become popular in the last couple of decades. Machine learning is all about training machines to think like humans, to make decisions like a human would and, although true human thought is still some way off for machines, it is improving by the day.

We discussed several data science techniques and machine learning algorithms that can help you work with data and draw meaningful insights from it. However, we've really only scraped the surface, and there is so much more to learn. Use this book as your starting point and, once you've mastered what's here, you can take your learning further. Data science and machine learning are two of the most coveted job opportunities in the world these days and, with the rise in data, there will only be more positions available. It makes sense to learn something that can potentially lead you to a lucrative and highly satisfying career.

Thank you once again for choosing my guide, and I wish you luck in your data science journey.

References

"A Comprehensive Guide to Python Data Visualization with Matplotlib and Seaborn." *Built In* , builtin.com/data-science/data-visualization-tutorial.

"A Guide to Machine Learning Algorithms and Their Applications." *Sas.com* , 2019, www.sas.com/en_gb/insights/articles/analytics/machine-learning-algorithms.html.

Analytics Vidhya. "Scikit-Learn in Python - Important Machine Learning Tool." *Analytics Vidhya* , 5 Jan. 2015, www.analyticsvidhya.com/blog/2015/01/scikit-learn-python-machine-learning-tool/.

"Difference between Data Science and Machine Learning: All You Need to Know in 2021." *Jigsaw Academy* , 9 Apr. 2021, www.jigsawacademy.com/what-is-the-difference-between-data-science-and-machine-learning/.

Leong, Nicholas. "Python for Data Science — a Guide to Pandas." *Medium* , 11 June 2020, towardsdatascience.com/python-for-data-science-basics-of-pandas-5f8d9680617e.

"Machine Learning Algorithms with Python." *Data Science | Machine Learning | Python | C++ | Coding | Programming | JavaScript* , 27 Nov. 2020, thecleverprogrammer.com/2020/11/27/machine-learning-algorithms-with-python/.

Mujtaba, Hussain. "Machine Learning Tutorial for Beginners | Machine Learning with Python." *GreatLearning* , 18 Sept. 2020, www.mygreatlearning.com/blog/machine-learning-tutorial/.

Mutuvi, Steve. "Introduction to Machine Learning Model Evaluation." *Medium* , Heartbeat, 16 Apr. 2019,

heartbeat.fritz.ai/introduction-to-machine-learning-model-evaluation-fa859e1b2d7f.

Python, Real. “Pandas for Data Science (Learning Path) – Real Python.” *Realpython.com* , realpython.com/learning-paths/pandas-data-science/.

“The Ultimate NumPy Tutorial for Data Science Beginners.” *Analytics Vidhya* , 27 Apr. 2020, www.analyticsvidhya.com/blog/2020/04/the-ultimate-numpy-tutorial-for-data-science-beginners/.