


JOHN S. CODE



PYTHON PROGRAMMING

FOR BEGINNERS

A HANDS-ON EASY GUIDE FOR BEGINNERS TO LEARN PYTHON
PROGRAMMING FAST, CODING LANGUAGE, DATA
ANALYSIS WITH TOOLS AND TRICKS



PYTHON PROGRAMMING FOR BEGINNERS: A HANDS-ON EASY GUIDE FOR BEGINNERS TO LEARN PYTHON PROGRAMMING FAST, CODING LANGUAGE, DATA ANALYSIS WITH TOOLS AND TRICKS.

JOHN S. CODE

© Copyright 2019 - All rights reserved.

The content contained within this book may not be reproduced, duplicated or transmitted without direct written permission from the author or the publisher.

Under no circumstances will any blame or legal responsibility be held against the publisher, or author, for any damages, reparation, or monetary loss due to the information contained within this book. Either directly or indirectly.

Legal Notice:

This book is copyright protected. This book is only for personal use. You cannot amend, distribute, sell, use, quote or paraphrase any part, or the content within this book, without the consent of the author or publisher.

Disclaimer Notice:

Please note the information contained within this document is for educational and entertainment purposes only. All effort has been executed to present accurate, up to date, and reliable, complete information. No warranties of any kind are declared or implied. Readers acknowledge that the author is not engaging in the rendering of legal, financial, medical or professional advice. The content within this book has been derived from various sources. Please consult a licensed professional before attempting any techniques outlined in this book.

By reading this document, the reader agrees that under no circumstances is the author responsible for any losses, direct or indirect, which are incurred as a result of the use of information contained within this document, including, but not limited to, — errors, omissions, or inaccuracies.

Table of Contents

[Introduction](#)

[Chapter 1 Mathematical Concepts](#)

[Chapter 2 What Is Python](#)

[Chapter 3 Writing The First Python Program](#)

[Chapter 4 The Python Operators](#)

[Chapter 5 Basic Data Types In Python](#)

[Chapter 6 Data Analysis with Python](#)

[Chapter 7 Conditional Statements](#)

[Chapter 8 Loops – The Never-Ending Cycle](#)

[Chapter 9 File handling](#)

[Chapter 10 Exception Handling](#)

[Chapter 11 Tips and Tricks For Success](#)

[Conclusion](#)

Introduction

Python is an awesome decision on machine learning for a few reasons. Most importantly, it's a basic dialect at first glance. Regardless of whether you're not acquainted with Python, getting up to speed is snappy in the event that you at any point have utilized some other dialect with C-like grammar.

Second, Python has an incredible network which results in great documentation and inviting and extensive answers in Stack Overflow (central!).

Third, coming from the colossal network, there are a lot of valuable libraries for Python (both as "batteries included" an outsider), which take care of essentially any issue that you can have (counting machine learning).

History of Python

Python was invented in the later years of the 1980s. Guido van Rossum, the founder, started using the language in December 1989. He is Python's only known creator and his integral role in the growth and development of the language has earned him the nickname "Benevolent Dictator for Life". It was created to be the successor to the language known as ABC.

The next version that was released was Python 2.0, in October of the year 2000 and had significant upgrades and new highlights, including a cycle-distinguishing junk jockey and back up support for Unicode. It was most fortunate, that this particular version, made vast improvement procedures to the language turned out to be more straightforward and network sponsored.

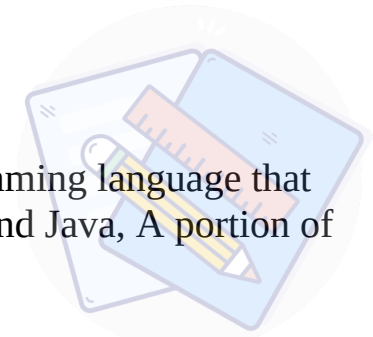
Python 3.0, which initially started its existence as Py3K. This version was rolled out in December of 2008 after a rigorous testing period. This particular version of Python was hard to roll back to previous compatible versions which are the most unfortunate. Yet, a significant number of its real highlights have been rolled back to versions 2.6 or 2.7 (Python), and rollouts of Python 3 which utilizes the two to three utilities, that helps to automate the interpretation of the Python script.

Python 2.7's expiry date was originally supposed to be back in 2015, but for unidentifiable reasons, it was put off until the year 2020. It was known that there was a major concern about data being unable to roll back but roll FORWARD into the new version, Python 3. In 2017, Google declared that there would be work done on Python 2.7 to enhance

execution under simultaneously running tasks.

Basic features of Python

Python is an unmistakable and extremely robust programming language that is object-oriented based almost identical to Ruby, Perl, and Java, A portion of Python's remarkable highlights:



- Python uses a rich structure, influencing, and composing projects that can be analyzed simpler.
- It accompanies a huge standard library that backs tons of simple programming commands, for example, extremely seamless web server connections, processing and handling files, and the ability to search through text with commonly used expressions and commands.
- Python's easy to use interactive interface makes it simple to test shorter pieces of coding. It also comes with IDLE which is a "development environment".

The Python programming language is one of many different types of coding languages out there for you. Some are going to be suited the best to help out with websites. There are those that help with gaming or with specific projects that you want to handle. But when it comes to finding a great general-purpose language, one that is able to handle a lot of different tasks all at once, then the Python coding language is the one for you.

There are a lot of different benefits to working with the Python language. You will find that Python is easy enough for a beginner to learn how to work with. It has a lot of power behind it, and there is a community of programmers and developers who are going to work with this language to help you find the answers that you are looking for. These are just some of the benefits that we get to enjoy with the Python language, and part of the reason why we will want to get started with this language as soon as possible!

The Python programming language is a great general-purpose language that is able to take care of all your computing and programming needs. It is also freely available and can make solving some of the bigger computer programs that you have as easy as writing out some of the thoughts that you have about that solution. You are able to write out the code once, and then, it is able to run on almost any kind of program that you would like without you needing

to change up the program at all.

How is Python used?

Python is one of the best programming languages that is a general-purpose and is able to be used on any of the modern operating systems that you may have on your system. You will find that Python has the capabilities of processing images, numbers, text, scientific data, and a lot of other things that you would like to save and use on your computer.

Python may seem like a simple coding language to work with, but it has a lot of the power and more that you are looking for when it is time to start with programming. In fact, many major businesses, including YouTube, Google, and more, already use this coding language to help them get started on more complex tasks.

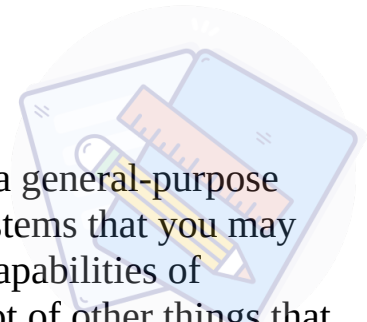
Python is also known as a type of interpreted language. This means that it is not going to be converted into code that is readable by the computer before the program is run. Instead, this is only going to happen at runtime. Python and other programming languages have changed the meaning of this kind of coding and have ensured that it is an accepted and widely used coding method for many of the projects that you would like to handle.

There are a lot of different tasks that the Python language is able to help you complete. Some of the different options that you are able to work with include:

1. Programming any of the CGI that you need on your web applications.
2. Learning how to build up your own RSS reader
3. Working with a variety of files.
4. Creating a calendar with the help of HTML
5. Being able to read from and write in MySQL
6. Being able to read from and write to PostgreSQL

The Benefits of Working with Python

When it comes to working with the Python language, you will find that there are a lot of benefits with this kind of coding language. It is able to help you to complete almost any kind of coding process that you would like and can still

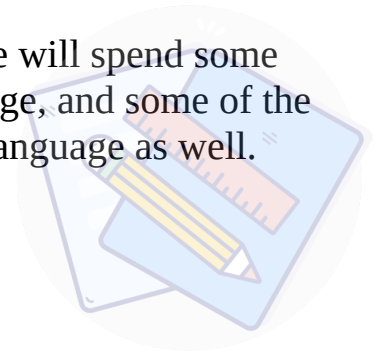


have some of the ease of use that you are looking for. Let's take a quick look at some of the benefits that come with this kind of coding language below:

- Beginners can learn it quickly. If you have always wanted to work with a coding language, but you have been worried about how much work it is going to take, or that it will be too hard for you to handle, then Python is the best option. It is simple to use and has been designed with the beginner in mind.
- It has a lot of power to enjoy. Even though Python is easy enough for a beginner to learn how to use, that doesn't mean that you are going to be limited to the power that you are able to get with some of your codings. You will find that the Python language has the power and more that you need to get so many projects done.
- It can work with other coding languages. When we get to work on data science and machine learning, you will find that this is really important. There are some projects where you will need to combine Python with another language, and it is easier to do than you may think!
- It is perfect for simple projects all the way up to more complex options like machine learning and data analysis. This will help you to complete any project that you would like.
- There are a lot of extensions and libraries that come with the Python language, which makes it the best option for you to choose for all your projects. There are a lot of libraries that you are able to add to Python to make sure that it has the capabilities that you need.
- There is a large community that comes with Python. This community can answer your questions, show you some of the different codes that you can work with, and more. As a beginner, it is always a great idea to work with some of these community members to ensure that you are learning as much as possible about Python.

When it comes to handling many of the codes and more that you would like in your business or on other projects, nothing is going to be better than

working with the Python language. In this guidebook, we will spend some time exploring the different aspects of the Python language, and some of the different things that you are able to do with this coding language as well.



Chapter 1 Mathematical Concepts

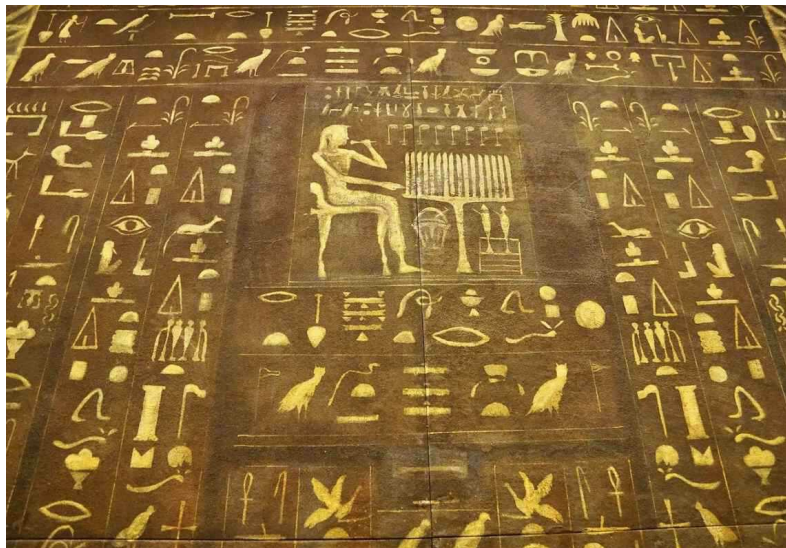
As we have stated before, computers are physical manifestations of several mathematical concepts. Mathematics are the scientific language of solving problems. Over the centuries, mathematicians have theoretically solved many complex issues. Mathematics includes concepts like algebra and geometry.

Number Systems

Mathematics is a game of number manipulation which makes number systems at the center stage of mathematical concepts. There are several different types of number systems. Before we take a look at the number systems, we have to understand the concept of coding.

Coding

A way to represent values using symbols is called coding. Coding is as old as humans. Before the number systems we use today, there were other systems to represent values and messages. An example of coding from ancient times is the Egyptian hieroglyphs.



Number systems are also examples of coding because values are represented using special symbols.

There are different types of number systems, and we are going to discuss a few relevant ones.

Binary System

A binary system has only two symbols, 1 and 0 which are referred to as bits. All the numbers are represented by combining these two symbols. Binary

systems are ideal for electronic devices because they also have only two states, on or off. In fact, all electronic devices are based on the binary number system. The number system is positional which means the position of symbols determines the final value. Since there are two symbols in this system, the system has a base of 2.

The sole purpose of input and output systems is to convert data to and from binary system to a form that makes better sense to the user. The first bit from the left side is called Most Significant Bit (MSB) while the first bit from the right is called the Least Significant Bit (LSB).

Here is the binary equivalent code of “this is a message”:

```
01110100 01101000 01101001 01110011 00100000 01101001 01110011
00100000 01100001 00100000 01101101 01100101 01110011 01110011
01100001 01100111 01100101
```

Decimal System

The decimal system has ten symbols, the numbers 0 through 9. This is also a positional number system where the position of symbols changes the value it represents. All the numbers in this system are created with different combinations of the initial ten symbols. This system has a base 10.

This is also called the Hindu-Arabic number system. Decimals make more sense to humans and are used in daily life. There are two reasons for that.

- Creating large numbers from the base symbols follows a consistent pattern

- Performing arithmetic operations in a decimal system is easier compared to other systems

Hexadecimal System

The hexadecimal number system is the only one that has letters as symbols. It has the 10 symbols of the decimal system plus the six alphabets A, B, C, D, E and F. This is also a positional number system with a base 16.

Hexadecimal system is extensively used to code instructions in assembly language.

Number System Conversion

We can convert the numbers from one system to another. There are various online tools to do that. Python also offers number conversion, but it is better

to learn how it is done manually.

Binary to Decimal

Here's a binary number 01101001, let's convert it to a decimal number.

$$(01101001)_2 = 0 \times 2^7 + 1 \times 2^6 + 1 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$$

$$(01101001)_2 = 0 + 64 + 32 + 0 + 8 + 0 + 0 + 1$$

$$(01101001)_2 = (105)_{10}$$

Decimal to Binary

To convert a decimal number to binary, we have to repeatedly divide the number by two until the quotient becomes one. Recording the remainder generated at each division step gives us the binary equivalent of the decimal number.

2	105	
2	52	1
2	26	0
2	13	0
2	6	1
2	3	0
	1	1

$(105)_{10} = (1101001)_2$

An interesting thing to note here is that $(01101001)_2$ and $(1101001)_2$ represent the same decimal number $(105)_{10}$. It means that just like decimal number system, leading zeros can be ignored in the binary number system.

Binary to Hexadecimal

Binary numbers can be converted to hexadecimal equivalents using two methods.

1. Convert the binary number to decimal, then decimal to

hexadecimal number

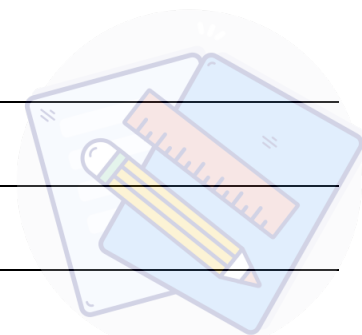
2. Break binary number in groups of four bits and convert each to its hexadecimal equivalent, keeping the groups' positions in the original binary number intact.

Let's convert $(1101001)_2$ to a hexadecimal number using the second method. The first step is to break the binary number into different groups each of four bits. If the MSB group has less than four bits, make it four by adding leading zeros. Grouping starts from the LSB. So, $(1101001)_2$ will give us $(1001)_2$ and $(0110)_2$. Now, remembering their position in the original binary number, we are going to convert each group to a hexadecimal equivalent.

Here is the table of hexadecimal equivalents of four-bit binary numbers.

Binary	Hexadecimal
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9
1010	A

1011	B
1100	C
1101	D
1110	E
1111	F



From the table, we can see $(1001)_2$ is $(9)_{16}$ and $(0110)_2$, the MSB group, is $(6)_{16}$.

Therefore, $(1101001)_2 = (01101001)_2 = (69)_{16}$

Hexadecimal to binary

We can use the above given table to quickly convert hexadecimal numbers to binary equivalents. Let's convert $(4EA9)_{16}$ to binary.

$$(4)_{16} = (0100)_2$$

$$(E)_{16} = (1110)_2$$

$$(A)_{16} = (1010)_2$$

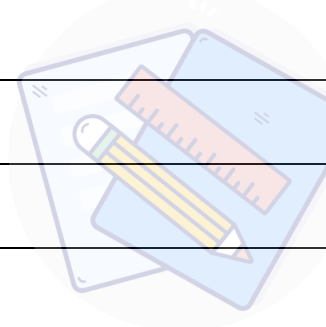
$$(9)_{16} = (1001)_2$$

$$\text{So, } (4EA9)_{16} = (0100111010101001)_2 = (100111010101001)_2$$

Decimal to Hexadecimal

You can say hexadecimal is an extended version of decimals. Let's convert $(45781)_{10}$ to decimal. But, first, we have to remember this table.

Decimal	Hexadecimal
0	0
1	1



2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
10	A
11	B
12	C
13	D
14	E
15	F

We are going to divide the decimal number repeatedly by 16 and record the remainders. The final hexadecimal equivalent is formed by replacing remainder decimals with their correct hexadecimal symbols.

16	45781	
16	2861	5
16	178	13
	11	2

$(45781)_{10} = (B2D5)_2$



Hexadecimal to Decimal

Let's convert $(4EA9)_{16}$ to its decimal equivalent.

$$(4EA9)_{16} = 4 \times 16^3 + 14 \times 16^2 + 10 \times 16^1 + 9 \times 16^0$$

$$(4EA9)_{16} = 16384 + 3584 + 160 + 9$$

$$(4EA9)_{16} = (20137)_{10}$$

There's another number system, the octal system, where the number of unique symbols include 0, 1, 2, 3, 4, 5, 6, along with 7. These were developed for small scale devices that worked on small values with limited resources. With the rapid advancements in storage and other computer resources, octal system became insufficient and thus was discarded in favor of hexadecimal number system. You might still find an old octal based computer system.

Fractions (Floating Points)

Decimal number system supports a decimal point '.' to represent portion/slices of a value. For example, if we want to say half of milk bag is empty using numbers, we can write 0.5 or $\frac{1}{2}$ of milk bag is empty. Do other number systems support decimal point? Yes, they do. Let's see how to convert $(0.75)_{10}$ or $(\frac{3}{4})_{10}$ to binary.

$$\frac{3}{4} \times 2 = 6/4 = 1. (2/4)$$

$$2/4 \times 2 = 4/4 = 1$$

$$(0.75)_{10} = (\frac{3}{4})_{10} = (.11)_2$$

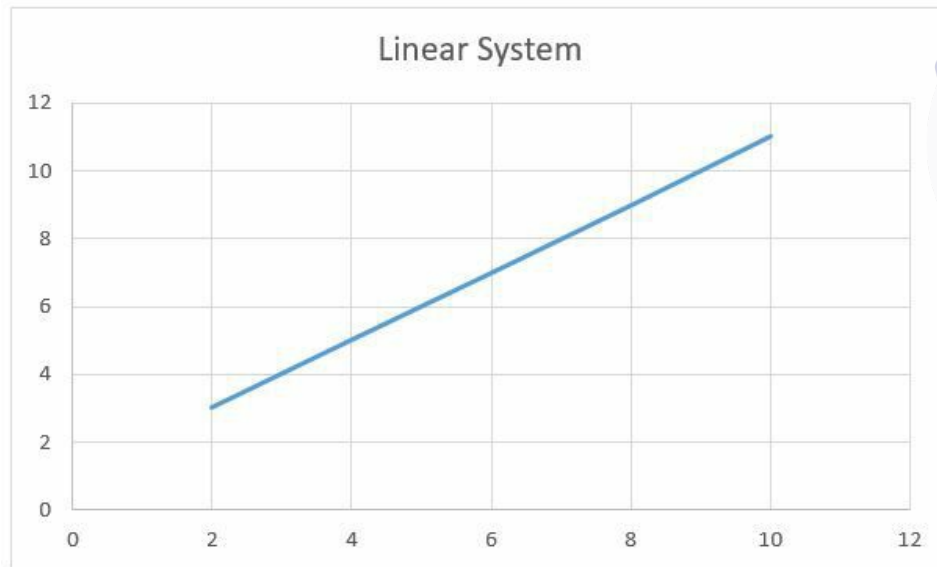
Negatives

In the decimal system, a dash or hyphen '-' is placed before a number to declare it as a negative. There are different ways to denote negative numbers in the binary system. The easiest is to consider the MSB as a sign bit, which means if MSB is 1, the number is negative and if the MSB is 0, the number is positive. Determining if a hexadecimal number is negative or positive is a bit tricky. The easiest way is to convert the number into binary and perform the checks for negatives in binary system.

Linear Algebra

Did you hate algebra in school? I have some bad news for you! Linear algebra is heavily involved in programming because it's one of the best mathematical ways to solve problems. According to Wikipedia, algebra is the study of mathematical symbols and the rules for manipulating these symbols. The field advanced thanks to the works of Muhammad ibn Musa al-Khwarizmi who introduced the reduction and balancing methods and treated algebra as an independent field of mathematics. During that era, the concept of 'x' and 'y' etc. variable notation wasn't widespread but during the Islamic Golden Age, Arabs had a fondness of lengthy "layman terms" descriptions of problems and solutions and that is what Khwarizmi explained algebra concepts in his book. The book dealt with many practical real-life problems including the fields of finance, planning, and legal.

So, we know what algebra is. But, where does "linear" comes from? For that, we have to understand what a linear system is. It is a mathematical model where the system attributes (variables) have a linear relation among themselves. The easiest way to explain this is if the plot between system attributes is a straight line, the system is linear. Linear systems are much simpler than the nonlinear systems. The set of algebraic concepts that relate to linear systems is referred to as linear algebra. Linear algebra helps resolve system problems such as missing attribute values. The first step is to create linear equations to establish the relationship between the system variables.



Statistics

Another important field of mathematics that is crucial in various computer science applications. Data analysis and machine learning wouldn't be what they are without the advancements made in statistical concepts during the 20th century. Let's see some concepts related to statistics.

Outlier

Outlier detection is very important in statistical analysis. It helps in homogenizing the sample data. After detecting the outliers, what to do with them is crucial because they directly affect the analysis results. There are many possibilities including:

Discarding Outlier

Sometimes it's better to discard outliers because they have been recorded due to some error. This usually happens where the behavior of the system is already known.

System Malfunction

But, outliers can also indicate a system malfunction. It is always better to investigate the outliers instead of discarding them straightaway.

Average

Finding the center of a data sample is crucial in statistical analysis because it reveals a lot of system characteristics. There are different types of averages, each signifying something important.

Mean

Mean is the most common average. All the data values are added and divided by the number of data values added together. For example, you sell shopping bags to a well-renowned grocery store and they want to know how much each shopping bag can carry. You completely fill 5 shopping bags with random grocery items and weigh them. Here are the readings in pounds.

5.5, 6.0, 4.95, 7.1, 5.0

You calculate the mean as $(5.5 + 6 + 4.95 + 7.1 + 5) / 5 = 5.71$. You can tell the grocery store your grocery bags hold 5.71 lbs on average.

Median

Median is the center value with respect to the position of data in a sample data when it's sorted in ascending order. If sample data has odd members, the median is the value with an equal number of values on both flanks. If sample data has an even number of values, the median is calculated by finding the mean of two values in the middle with equal number of items on both sides.

Mode

Mode is the most recurring value in a dataset. If there is no recurring value in the sample data, there is no mode.

Variance

To find how much each data value in a sample data changes with respect to the average of the sample data, we calculate the variance. Here is a general formula to calculate variance.

sum of (each data point - mean of sample points)² / number of data points in the sample.

If the variance is low in a sample data, it means there are no outliers in the data.

Standard Deviation

We take the square root of variance to find standard deviation. This relates the mean of sample data to the whole of sample data.

Probability

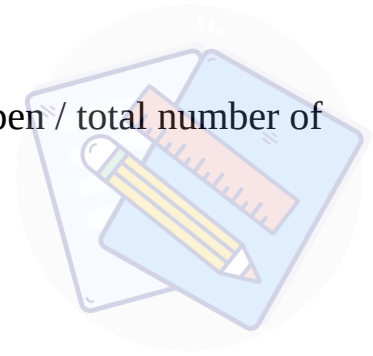
No one can accurately tell what will happen in the future. We can only predict what is going to happen with some degree of certainty. The

probability of an event is written mathematically as,

Probability = number of possible ways an event can happen / total number of possibilities

A few points:

1. Probability can never be negative
2. Probability ranges between one and zero
3. To calculate probability, we assume that the set of events we are working with occur independently without any interference.



Finding the probability of an event can change the probability of something happening in a subsequent event. It depends upon how we are interacting with the system to find event probabilities.

Distribution

There are many types of distributions. In this book, whenever we talk about distribution, we refer to probability distribution unless explicitly stated otherwise. Let's take an example of flipping coins and see what is the distribution of such events.

H H H

H H T

H T H

T T T

T H H

H T T

T H T

T T H

This is a very simple event with only a handful of possible outcomes. We can easily determine the probability of different outcomes. But, this is almost impossible in complex systems with thousands or millions of possible outcomes. Distributions work much better in such cases by visually representing the probability curve. It makes more sense than looking at a huge table of fractions or small decimal numbers.

We call a probability distribution discrete if we know all the possible outcomes beforehand.



Chapter 2 What Is Python

Python is going to be a programming language that is known as interpreted, general-purpose, high-level, and multiparadigm. Python is going to allow the programmers who use it some different styles of programming in order to create simple or even some complex programs, get quicker results than before, and write out code almost as if they are speaking in a human language, rather than like they are talking to a computer.

This language is so popular that you will find a lot of major companies that are already using it for their systems, including Google App Engine, Google Search, YouTube, iRobot machines and more. And as the knowledge about this language continues to increase, it is likely that we are going to see even more of the applications and sites that we rely on each day working with this language as well.

The initial development of Python was started by Guido van Rossum in the late 1980s. Today, it is going to be developed and run by the Python Software Foundation. Because of the features behind this language, programmers of Python can accomplish their tasks with many different styles of programming. Python can be used for a variety of things as well including serial port access, game development, numeric programming, and web development to name a few.

There are going to be a few different attributes that we can look at that show us why the development time that we see in Python is going to be faster and more efficient than what we are seeing with some of the other programming languages out there. These will include:

1. Python is an interpreted language. This means that there is no need for us to compile the code before we try to execute the program because Python will not need this compilation in the background. It is also going to be a more high-level language that will abstract many sophisticated details from the programming code. Much of this abstraction is going to be focused so that the code could be understood even by those who are just getting started with coding.
2. Python codes are often going to be shorter than similar codes in other languages. Although it does offer some fast times for

development, keep in mind that the execution time is going to lag a little bit. Compared to some of the other languages, such as fully compiling options like C and C++ < Python is going to execute at a slower rate. Of course, with the processing speeds that we see in most computers today, the differences in the speed are not really going to be noticed by the people who are using the system.

There are a lot of different things that we are able to do when it comes to the Python language, and we are going to spend some time looking at how we can do some of the different parts as well. From some of the basics that we are able to do with the help of Python all the way to some of the more complex things that we are able to do with data science and machine learning that we can talk about a little later.

And all of this can be done with the ease of the Python language. You will find that when we spend some time focusing on this language and some of the basics that we are able to do will help to prepare us for some of the more complicated things that we are able to do with this code as well.

A. WHY TO LEARN PYTHON

Learning the ABCs of anything in this world, is a must. Knowing the essentials is winning half the battle before you get started. It's easier to proceed when you are equipped with the fundamentals of what you are working on.

In the same manner that before you embark on the other aspects of python let us level off the basic elements first. You need to learn and understand the basics of python as a foundation in advancing to the more complicated components. This fundamental information will greatly help you as you go on and make the learning experience easier and enjoyable.

Familiarize yourself with the Python Official Website

<https://www.python.org/>. Knowing well the website of python would give you the leverage in acquiring more information and scaling up your knowledge about python. Also, you can get the needed links for your work

Learn from Python collections. Locate python collections such as records, books, papers, files, documentations and archives and learn from it. You can pick up a number of lessons from these, and expand your knowledge about Python. There are also tutorials, communities and forums at your disposal.

Possess the SEO Basics. Acquire some education on Search Engine Optimization so you can interact with experts in the field and improve your python level of knowledge. That being said, here are the basic elements of Python.

B. DIFFERENT VERSIONS OF PYTHON

With Guido van Rossum at the helm of affairs, Python has witness three versions over the years since its conception in the '80s. These versions represent the growth, development, and evolution of the scripting language over time, and cannot be done without in telling the history of Python.

The Versions of Python Include The Following;

• Python 0.9.0:

The first-ever version of Python released following its implementation and in-house releases at the Centrum Wiskunde and Informatica (CWI) between the years 1989 and 1990, was tagged version 0.9.0. This early version which was released on alt. sources had features such as exception handling, functions, and classes with inheritance, as well as the core data types of list, str, dict, among others in its development. The first release came with a module system obtained from Module-3, which Van Rossum defined as one of the central programming units used in the development of Python.

Another similarity the first release bore with Module-3 is found in the exception model which comes with an added else clause. With the public release of this early version came a flurry of users which culminated in the formation of a primary discussion forum for Python in 1994. The group was named comp. lang. python and served as a milestone for the growing popularity of Python users.

Following the release of the first version in the 29th of February, 1991, there were seven other updates made to the early version 0.9.0. These updates took varying tags under the 0.9.0 version and were spread out over nearly three years (1991 to 1993). The first version update came in the form of Python 0.9.1, which was released in the same month of February 1991 as its predecessor. The next update came in the autumn period of the release year, under the label Python 0.9.2. By Christmas Eve of the same year (1991) python published its third update to the earliest version under the label Python 0.9.4. By January of the succeeding year, the 2nd precisely, a gift update under the label Python 0.9.5 was released. By the 6th of April, 1992, a

sixth update followed named, Python 0.9.6. It wasn't until the next year, 1993, that a seventh update was released under the tag Python 0.9.8. The eighth and final update to the earliest version came five months after the seventh, on the 29th of July, 1993, and was dubbed python 0.9.9.

These updates marked the first generation of python development before it transcended into the next version label.

• Python 1.0

After the last update to Python 0.9.0, a new version, Python 1.0, was released in January of the following year. 1994 marked the addition of key new features to the Python programming language. Functional programming tools such as map, reduce, filter, and lambda were part of the new features of the version 1 release. Van Rossum mentioned that the obtainment of map, lambda, reduce and filter was made possible by a LISP hacker who missed them and submitted patches that worked. Van Rossum's contract with CWI came to an end with the release of the first update version 1.2 on the 10th of April, 1995. In the same year, Van Rossum went on to join CNRI (Corporation for National Research Initiatives) in Reston, Virginia, United States, where he continued to work on Python and published different version updates.

Nearly six months following the first version update, version 1.3 was released on the 12th of October, 1995. The third update, version 1.4, came almost a year later in October of 1996. By then, Python had developed numerous added features. Some of the typical new features included an inbuilt support system for complex numbers and keyword arguments which, although inspired by Modula-3, shared a bit of a likeness to the keyword arguments of Common Lisp. Another included feature was a simple form hiding data through name mangling, although it could be easily bypassed.

It was during his days at CNRI that Van Rossum began the CP4E (Computer Programming for Everybody) program which was aimed at making more people get easy access to programming by engaging in simple literacy of programming languages. Python was a pivotal element to van Rossum's campaign, and owing to its concentration on clean forms of syntax; Python was an already suitable programming language. Also, since the goals of ABC and CP4E were quite similar, there was no hassle putting Python to use. The

program was pitched to and funded by DARPA, although it did become inactive in 2007 after running for eight years. However, Python still tries to be relatively easy to learn by not being too arcane in its semantics and syntax, although no priority is made of reaching out to non-programmers again.

The year 2000 marked another significant step in the development of Python when the python core development team switched to a new platform — BeOpen where a new group, BeOpen PythonLabs team was formed. At the request of CNRI, a new version update 1.6 was released on the 5th of September, succeeding the fourth version update (Python 1.5) on the December of 1997. This update marked the complete cycle of development for the programming language at CNRI because the development team left shortly afterward. This change affected the timelines of release for the new version Python 2.0 and the version 1.6 update; causing them to clash. It was only a question of time before Van Rossum, and his crew of PythonLabs developers switched to Digital Creations, with Python 2.0 as the only version ever released by BeOpen.

With the version 1.6 release caught between a switch of platforms, it didn't take long for CNRI to include a license in the version release of Python 1.6. The license contained in the release was quite more prolonged than the previously used CWI license, and it featured a clause mentioning that the license was under the protection of the laws applicable to the State of Virginia. This intervention sparked a legal feud which led The Free Software Foundation into a debate regarding the "choice-of-law" clause being incongruous with that of the GNU General Public License. At this point, there was a call to negotiations between FSF, CNRI, and BeOpen regarding changing to Python's free software license which would serve to make it compatible with GPL. The negotiation process resulted in the release of another version update under the name of Python 1.6.1. This new version was no different from its predecessor in any way besides a few new bug fixes and the newly added GPL-compatible license.

- **Python 2.0:**

After the many legal dramas surrounding the release of the second-generation Python 1.0 which corroborated into the release of an unplanned update (version 1.6.1), Python was keen to put all behind and forge ahead. So, in October of 2000, Python 2.0 was released. The new release featured new additions such as list comprehensions which were obtained from other

functional programming languages Haskell and SETL. The syntax of this latest version was akin to that found in Haskell, but different in that Haskell used punctuation characters while Python stuck to alphabetic keywords.

Python 2.0 also featured a garbage collection system which was able to collect close reference cycles. A version update (Python 2.1) quickly followed the release of Python 2.0, as did Python 1.6.1. However, due to the legal issue over licensing, Python renamed the license on the new release to Python Software Foundation License. As such, every new specification, code or documentation added from the release of version update 2.1 was owned and protected by the PSF (Python Software Foundation) which was a nonprofit organization created in the year 2001. The organization was designed similarly to the Apache Software Foundation. The release of version 2.1 came with changes made to the language specifications, allowing support of nested scopes such as other statically scoped languages. However, this feature was, by default, not in use and unrequired until the release of the next update, version 2.2 on the 21st of December, 2001.

Python 2.2 came with a significant innovation of its own in the form of a unification of all Python's types and classes. The unification process merged the types coded in C and the classes coded in Python into a single hierarchy. The unification process caused Python's object model to remain totally and continuously object-oriented. Another significant innovation was the addition of generators as inspired by Icon. Two years after the release of version 2.2, version 2.3 was published in July of 2003. It was nearly another two years before version 2.4 was released on the 30th of November in 2004. Version 2.5 came less than a year after Python 2.4, in September of 2006. This version introduced a "with" statement containing a code block in a context manager; as in obtaining a lock before running the code block and releasing the lock after that or opening and closing a file. The block of code made for behavior similar to RAI (Resource Acquisition Is Initialization) and swapped the typical "try" or "finally" idiom.

The release of version 2.6 on the 1st of October, 2008 was strategically scheduled such that it coincided with the release of Python 3.0. Besides the proximity in release date, version 2.6 also had some new features like the "warnings" mode which outlined the use of elements which had been omitted from Python 3.0. Subsequently, in July of 2010, another update to Python 2.0 was released in the version of python 2.7. The new version updates shared

features and coincided in release with version 3.1 — the first version update of python 3. At this time, Python drew an end to the release of Parallel 2.x and 3.x, making python 2.7 the last version update of the 2.x series. Python went public in 2014, November precisely, to announce to its username that the availability of python 2.7 would stretch until 2020. However, users were advised to switch to python 3 in their earliest convenience.

- **Python 3.0:**

The fourth generation of Python, Python 3.0, otherwise known as Py3K and python 3000, was published on the 3rd of December 2008. This version was designed to fix the fundamental flaws in the design system of the scripting language. A new version number had to be made to implement the required changes which could not be run while keeping the stock compatibility of the 2.x series that was by this time redundant. The guiding rule for the creation of python 3 was to limit the duplication of features by taking out old formats of processing stuff. Otherwise, Python three still followed the philosophy with which the previous versions were made. Albeit, as Python had evolved to accumulate new but redundant ways of programming alike tasks, python 3.0 was emphatically targeted at quelling duplicative modules and constructs in keeping with the philosophy of making one "and

preferably only one" apparent way of doing things. Regardless of these changes, though, version 3.0 maintained a multi-paradigm language, even though it didn't share compatibility with its predecessor.

The lack of compatibility meant Python 2.0 codes were unable to be run on python 3.0 without proper modifications. The dynamic typing used in Python as well as the intention to change the semantics of specific methods of dictionaries, for instance, made a perfect mechanical conversion from the 2.x series to version 3.0 very challenging. A tool, name of 2to3, was created to handle the parts of translation which could be automatically done. It carried out its tasks quite successfully, even though an early review stated that the tool was incapable of handling certain aspects of the conversion process. Proceeding the release of version 3.0, projects that required compatible with both the 2.x and 3.x series were advised to be given a singular base for the 2.x series. The 3.x series platform, on the other hand, was to produce releases via the 2to3 tool.

For a long time, editing the Python 3.0 codes were forbidden because they

required being run on the 2.x series. However, now, it is no longer necessary. The reason being that in 2012, the recommended method was to create a single code base which could run under the 2.x and 3.x series through compatibility modules. Between the December of 2008 and July 2019, 8 version updates have been published under the python 3.x series. The current version as at the 8th of July 2019 is the Python 3.7.4. Within this timeframe, many updates have been made to the programming language, involving the addition of new features mentioned below:

1. Print which used to be a statement was changed to an inbuilt function, making it relatively easier to swap out a module in utilizing different print functions as well as regularizing the syntax. In the late versions of the 2.x series, (python 2.6 and 2.7), print is introduced as inbuilt, but is concealed by a syntax of the print statement which is capable of being disabled by entering the following line of code into the top of the file:
`from __future__ import print_function`
2. The [input] function in the Python 2.x series was removed, and the [raw_input] function to [input] was renamed. The change was such that the [input] function of Python 3 behaves similarly to the [raw_input] function of the python 2.x series; meaning input is typically outputted in the form of strings instead of being evaluated as a single expression.
3. [reduce] was removed with the exemption of [map] and [filter] from the in-built namespace into [functools]. The reason behind this change is that operations involving [reduce] are better expressed with the use of an accumulation loop.
4. Added support was provided for optional function annotations which could be used in informal type declarations as well as other purposes.
5. The [str]/[unicode] types were unified, texts represented, and immutable bytes type were introduced separately as well as a mutable [bytearray] type which was mostly corresponding; both of which indicate several arrays of bytes.
6. Taking out the backward-compatibility features such as implicit

relative imports, old-style classes, and string exceptions.

7. Changing the mode of integer division functionality. For instance, in the Python 2.x series, $5/2$ equals 2. Note that in the 3.x series, $5/2$ equals 2.5. From the recent versions of the 2.x series beginning from version 2.2 up until python 3: $5//2$ equals 2.

In contemporary times, version releases in the version 3.x series have all been equipped with added, substantial new features; and every ongoing development on Python is being done in line with the 3.x series.

C. HOW TO DOWNLOAD AND INSTALL PYTHON

In this time and age, being techy is a demand of the times, and the lack of knowledge, classifies one as an outback. This can result to being left out from the career world, especially in the field of programming.

Numerous big shot companies have employed their own programmers for purposes of branding, and to cut back on IT expenses.

In the world of programming, using Python language is found to be easier and programmer-friendly, thus, the universal use.

Discussed below are information on how to download python for MS Windows. In this particular demo, we have chosen windows because it's the most common worldwide – even in not so progressive countries. We want to cater to the programming needs of everyone all over the globe.


Python 2.7.17 version was selected because this version bridges the gap between the old version 2 and the new version 3.

Some of the updated functions/applications of version 3 are still not compatible with some devices, so 2.7.17 is a smart choice.

Steps in downloading Python 2.7.17, and installing it on Windows

1. Type python on your browser and press the Search button to display the search results.

Scroll down to find the item you are interested in. In this instance, you are looking for python. click “python releases for windows”, and a new page opens. See image below:



python

All Images Videos Books News More Settings Tools

About 590,000,000 results (0.55 seconds)

Welcome to Python.org

<https://www.python.org>

The official home of the **Python** Programming Language.

Search python.org

Downloads

Python 3.7.4 - Windows - Mac OS X - Python 3.7.3 - Python 2.7.16

Mac OS X

Python Releases for Mac OS X.
Latest Python 3 Release ...

Python For Beginners

BeginnersGuide/Download - Python for Programmers - Book

Applications

Applications for Python. Python is used in many application ...

The Python Tutorial

1. Whetting Your Appetite - 5. Data Structures - 9. Classes - Glossary

Jobs

Locations - Developer / Engineer - Telecommute - Back end - Types

2. Select the Python version, python 2.7.17, and click, or you can select the version that is compatible to your device or OS.

Python Releases for Windows

- [Latest Python 3 Release - Python 3.8.1](#)
- [Latest Python 2 Release - Python 2.7.17](#)



Stable Releases

- [Python 3.8.1 - Dec. 18, 2019](#)

Note that Python 3.8.1 *cannot* be used on Windows XP or earlier.

- Download [Windows help file](#)
- Download [Windows x86-64 embeddable zip file](#)
- Download [Windows x86-64 executable installer](#)
- Download [Windows x86-64 web-based installer](#)
- Download [Windows x86 embeddable zip file](#)
- Download [Windows x86 executable installer](#)
- Download [Windows x86 web-based installer](#)

- [Python 3.7.6 - Dec. 18, 2019](#)

Note that Python 3.7.6 *cannot* be used on Windows XP or earlier.

- Download [Windows help file](#)
- Download [Windows x86-64 embeddable zip file](#)
- Download [Windows x86-64 executable installer](#)
- Download [Windows x86-64 web-based installer](#)
- Download [Windows x86 embeddable zip file](#)

3. The new page contains the various python types. Scroll down and select an option: in this instance, select Windows x86 MSI installer and click.

Files

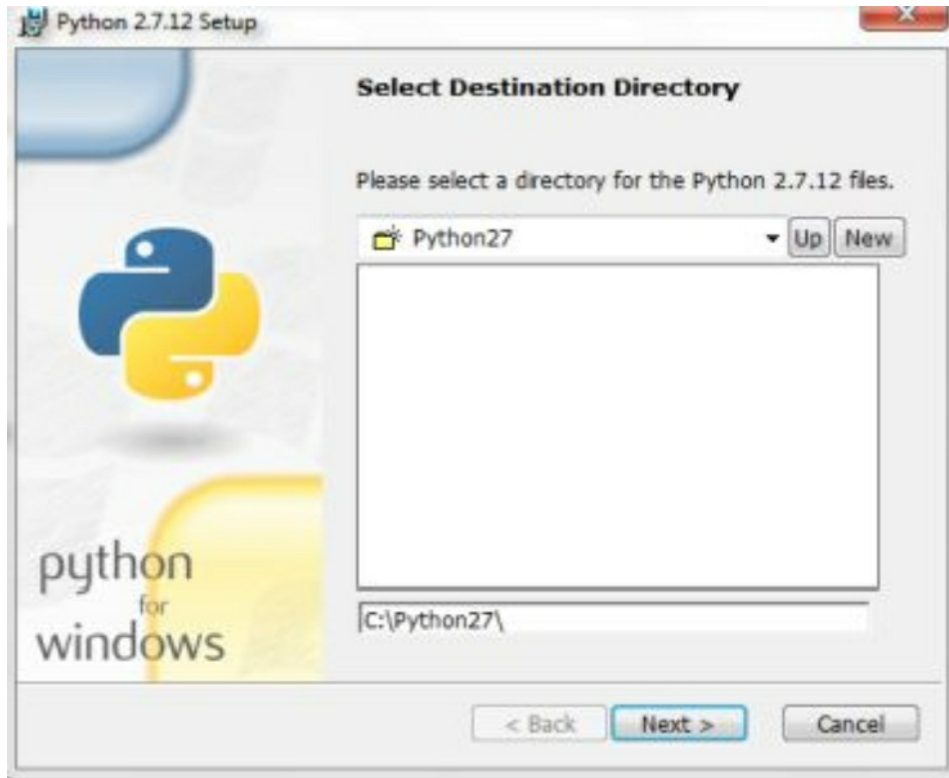
Version	Operating System	Description	MD5 Sum	File Size	GPG
Stipped source tarball	Source release		88d61f02e3616a+be952828309*109d	16935960	SIG
XZ compressed source tarball	Source release		57dfce9ce8bb2ab5f82af1d8+9a69	12390820	SIG
Mac OS X 32-bit (i386)/PPC installer	Mac OS X	for Mac OS X 10.5 and later	3adbedcc935a0db1ab08aa41fec4e33	24214628	SIG
Mac OS X 64-bit/32-bit installer	Mac OS X	for Mac OS X 10.6 and later	86beddc2b0cd37335d27aw2df84952e1	22355024	SIG
Windows debug information files	Windows		1751598e16431be04e1f4f24ca52b53a	24678566	SIG
Windows debug information files for 64-bit binaries	Windows		c5+33a7fca9edefe52835bd40e+0aa9d	25481362	SIG
Windows help file	Windows		7bc4e15eca8ede7c85e122f0a6d5f27	6224175	SIG
Windows x86-64 MSI installer	Windows	for AMD64/EM64T/x64, not Itanium processors	8fa13925db17638aa472a3e794caee3	19820544	SIG
Windows x86 MSI installer	Windows		fe0ef5b8fd02722f327f284324934ff9d	18907136	SIG

4. Press the Python box at the bottom of your screen.

Click the “Run” button, and wait for the new window to appear.

5. Select the user options that you require and press “NEXT”.

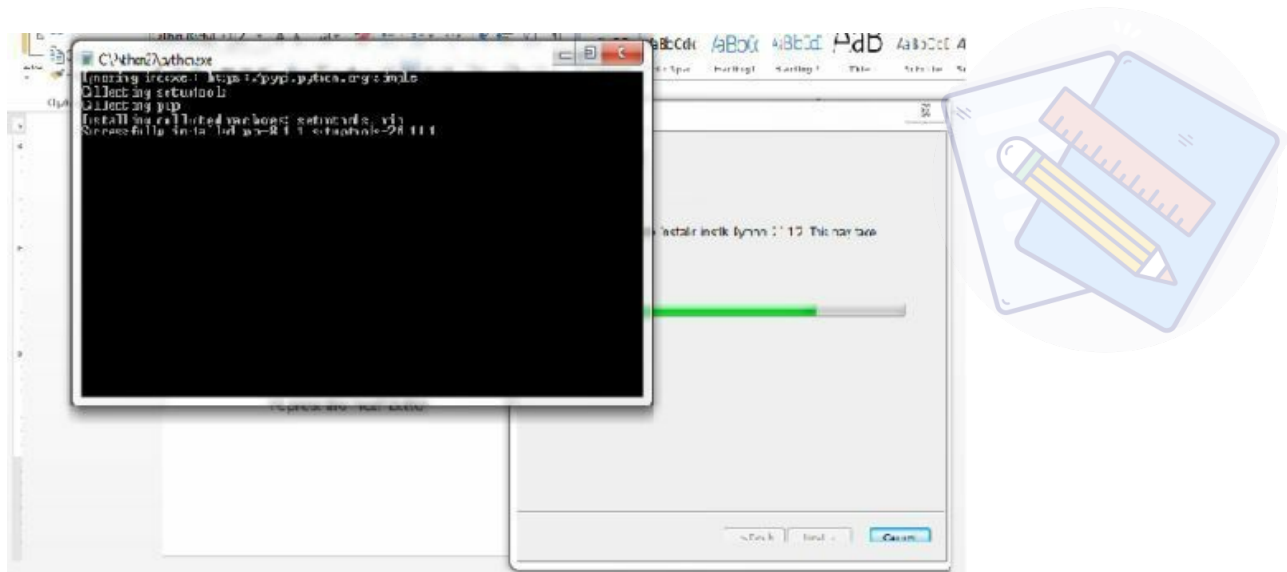
Your screen will display the hard drive where your python will be located.



6. Press the “NEXT” button.

7. Press yes, and wait for a few minutes. Sometimes it can take longer for the application to download, depending on the speed of your internet.

8. After that, click the FINISHED button to signify that the installation has been completed



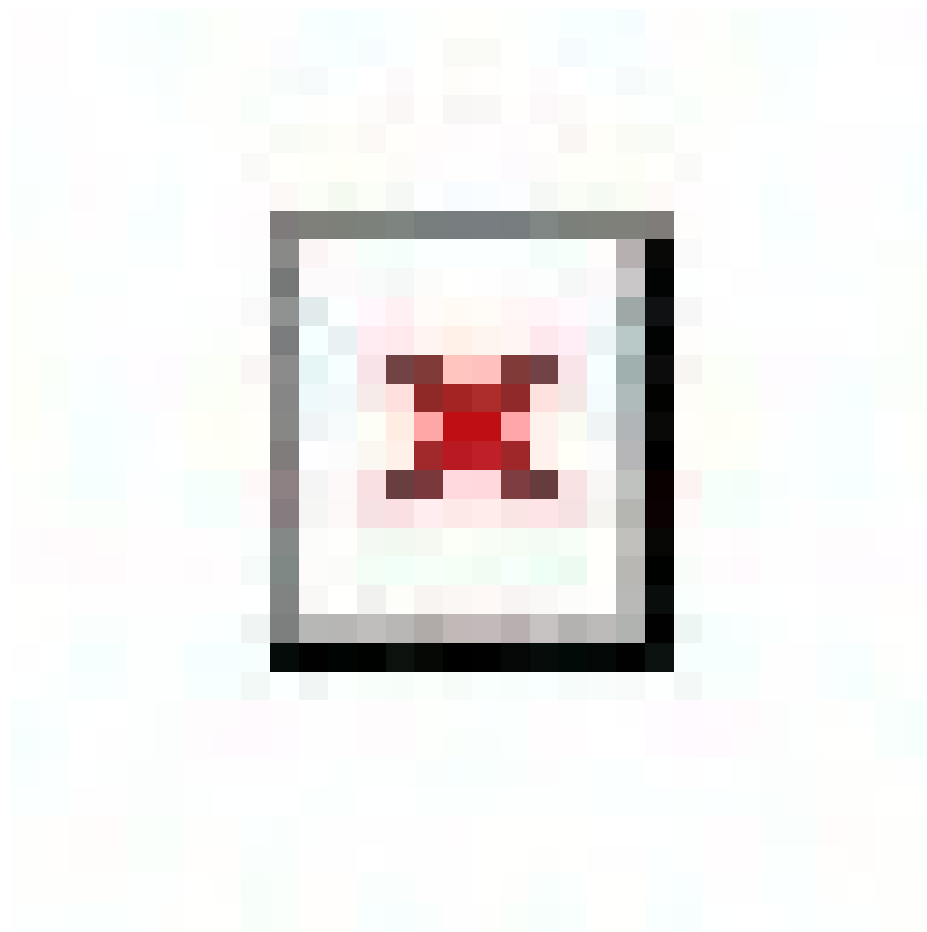
Your python has been installed in your computer and is now ready to use. Find it in drive C, or wherever you have saved it.

There can be glitches along the way, but there are options which are presented in this article. If you follow it well, there is no reason that you cannot perform this task.

It's important to note that there's no need to compile programs. Python is an interpretive language and can execute quickly your commands.

You can also download directly from the Python website, by selecting any of these versions – 3.8.1 or 2.7.17. and clicking 'download.'


See image below:



Follow the step by step instructions prompted by the program itself. Save and run the program in your computer.

For Mac

To download Python on Mac, you can follow a similar procedure, but this time, you will have to access the “Python.mpkg” file, to run the installer.



The image shows the top section of the Python.org website. It features a dark blue header with white navigation links: About, Downloads, Documentation, Community, Success Stories, News, and Events. Below the header is a large banner with a dark blue background. On the left, it says "Download the latest version for Mac OS X" in yellow, followed by a yellow button that says "Download Python 3.8.1". Below this, there are links for "Looking for Python with a different OS?" and "Want to help test development versions of Python?". On the right side of the banner, there is an illustration of two parachutes with yellow and white striped canopies, each carrying a brown cardboard box. To the right of the banner, there is a faint, circular watermark containing a stylized illustration of a blue notepad, a yellow pencil, and a red ruler.

Download the latest version for Mac OS X

[Download Python 3.8.1](#)

Looking for Python with a different OS? Python for [Windows](#), [Linux/UNIX](#), [Mac OS X](#), [Other](#)

Want to help test development versions of Python? [Prereleases](#), [Docker images](#)

Looking for Python 2.7? See below for specific releases

Looking for a specific release?

Python releases by version number:

Release version	Release date		Click for more
Python 3.8.1	Dec. 18, 2019	Download	Release Notes
Python 3.7.6	Dec. 18, 2019	Download	Release Notes
Python 3.6.10	Dec. 18, 2019	Download	Release Notes
Python 3.5.9	Nov. 2, 2019	Download	Release Notes

For Linux

For Linux, Python 2 and 3 may have been installed by default. Hence, check first your operating system. You can check if your device has already a Python program, by accessing your command prompt and entering this: `python—version`, or `python3—version`.

If Python is not installed in your Linux, the result “command not found” will be displayed. You may want to download both Python 2.7.17 and any of the versions of Python 3 for your Linux. This is due to the fact that Linux can have more compatibility with Python 3.

For windows users, now that you have downloaded the program, you’re ready to start.

And yes, congratulations! You can now begin working and having fun with your Python programming system.

Chapter 3 Writing The First Python Program

Beginners may find it difficult to start using Python. It's a given and nothing's wrong about that. However, your desire to learn will make it easier for you to gradually become familiar with the language.

Here are the specific steps you can follow to start using Python.

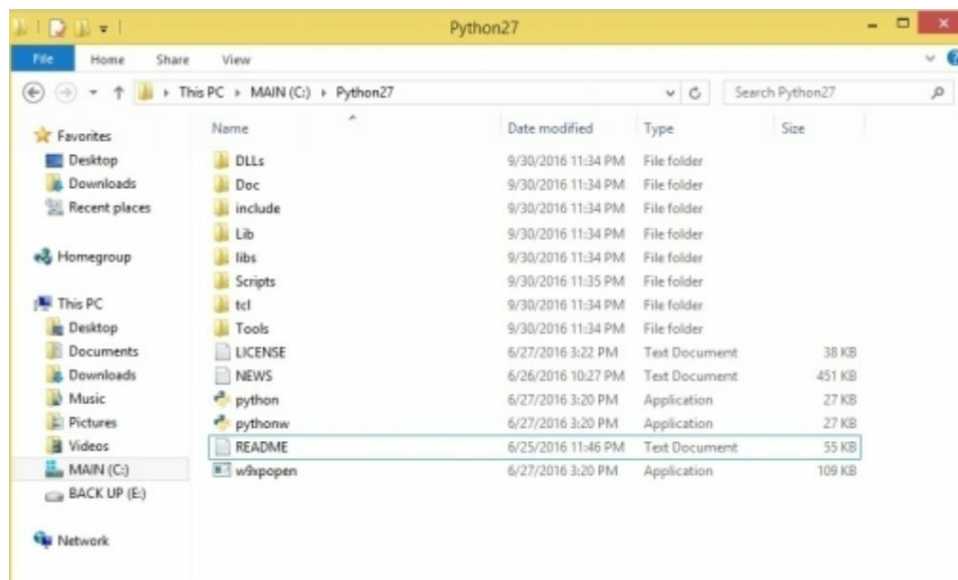
Steps in using Python

Step #1–Read all about Python.

Python has included a README information in your downloaded version. It's advisable to read it first, so you will learn more about the program.

You can start using your Python through the command box (black box), or you can go to your saved file and read first the README file by clicking it.

See image below:



This box will appear.



You can read the content completely, if you want to understand more what the program is all about, the file-setup, and similar information.

This is a long data that informs you of how to navigate and use Python. Also, Python welcomes new contributions for its further development.

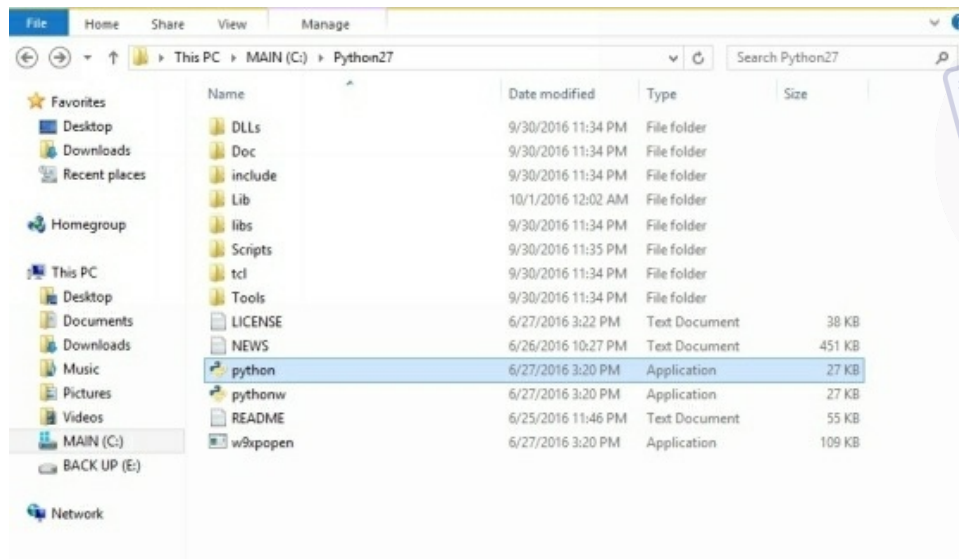
You can copy paste the content of the box into a Window document for better presentation.

If you don't want to know all the other information about Python and you're raring to go, you can follow these next steps.

Step #2–Start using Python.

First open the Python file you have saved in your computer. Click on Python as show below. In some versions, you just click 'python'for the shell to appear.

See image below:



You can start using Python by utilizing the simplest function, which is ‘print’. It’s the simplest statement or directive of python. It prints a line or string that you specify.

For Python 2, print command may or may not be enclosed in parenthesis or brackets, while in Python 3 you have to enclose print with brackets.

Example for Python 2:

print “Welcome to My Corner.”

Example for Python 3:

print (“Welcome to My Corner”)

The image below shows what appears when you press ‘enter’.



```
C:\Python27\python.exe
Python 2.7.12 (v2.7.12:d33e0cf91556, Jun 27 2016, 15:19:22) [MSC v.1500 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> print "Welcome to My Corner."
Welcome to My Corner.
>>> _
```

You may opt to use a Python shell through idle. If you do, this is how it would appear:

```
Python 2.7.12 Shell
File Edit Shell Debug Options Window Help
Python 2.7.12 (v2.7.12:d33e0cf91556, Jun 27 2016, 15:19:22) [MSC v.1500 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> print "Welcome to My Corner."
Welcome to My Corner.
>>> |
```

In the Python 3.5.2 version, the text colors are: function (purple), string (green) and the result (blue). (The string is composed of the words inside the bracket ("Welcome to My Corner"), while the function is the command word outside the bracket (print).

Take note that the image above is from the Python 2.7.12 version.

You have to use indentation for your Python statements/codes. The standard Python code uses four spaces. The indentations are used in place of braces or blocks.

In some programming languages, you usually use semi-colons at the end of the commands—in python, you don't need to add semi-colons at the end of the whole statement.

In Python, semi-colons are used in separating variables inside the brackets.

For version 3, click on your downloaded Python program and save the file in your computer. Then Click on IDLE (Integrated DeveLopment Environment), your shell will appear. You can now start using your Python. It's preferable to use idle, so that your codes can be interpreted directly by

idle.

Alternative method to open a shell (for some versions).

An alternative method to use your Python is to open a shell through the following steps:

Step #1– Open your menu.

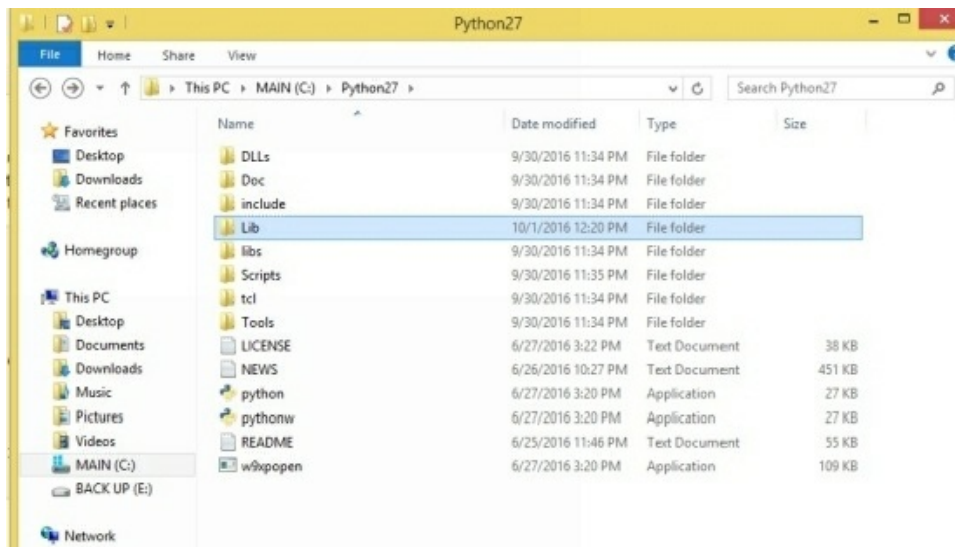
After downloading and saving your Python program in your computer, open your menu and find your saved Python file. You may find it in the downloaded files of your computer or in the files where you saved it.

Step #2–Access your Python file.

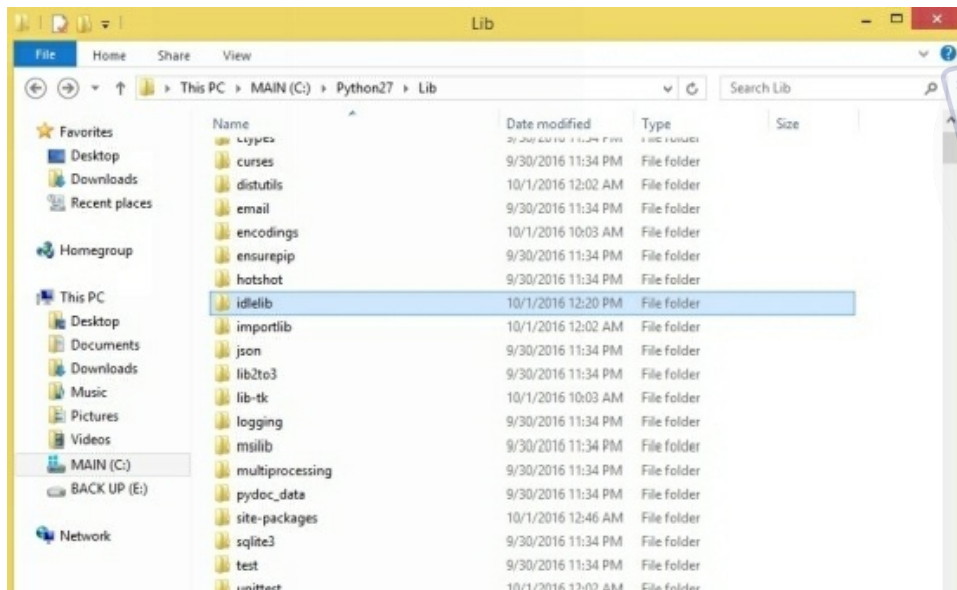
Open your saved Python file (Python 27) by double clicking it. The contents of Python 27 will appear. Instead of clicking on Python directly (as shown above), click on Lib instead.



See image below.

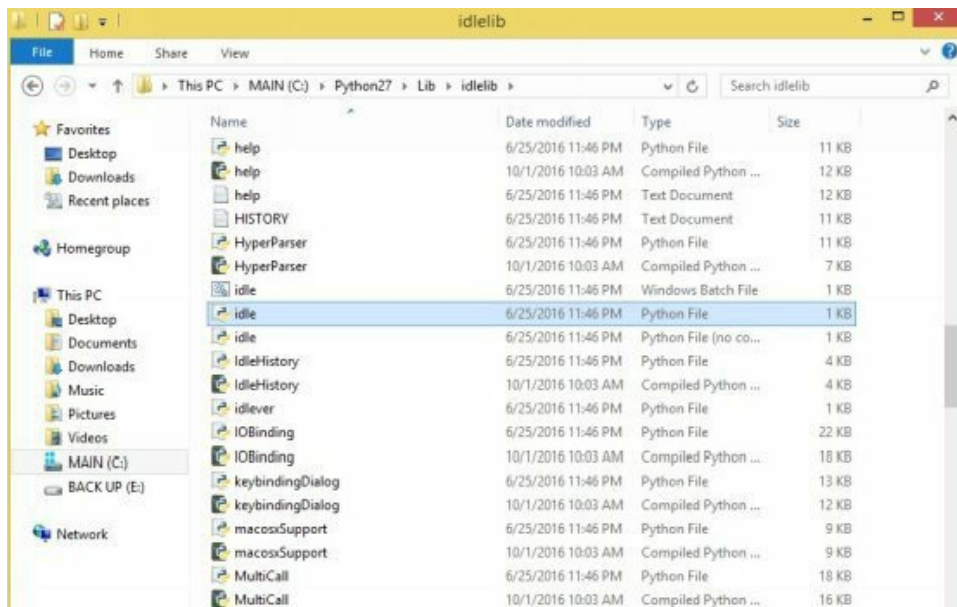


This will appear:



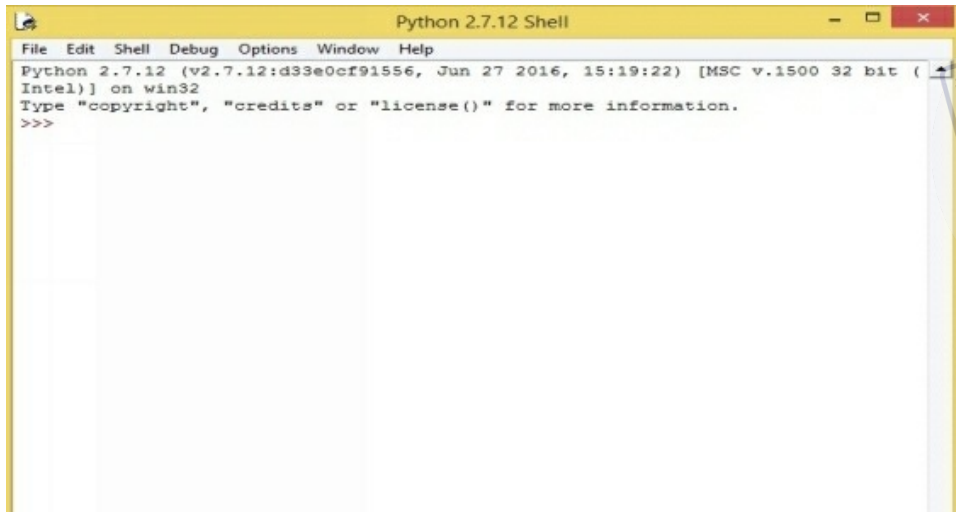
Step #3–Click on ‘idlelib’.

Clicking the ‘idlelib’ will show this content:



Step #4–Click on idle to show the Python shell.

When you click on any of the ‘idle’ displayed on the menu, the ‘white’ shell will be displayed, as shown below:



The differences between the three 'idle' menu, is that the first two 'idle' commands have the black box (shell) too, while the last 'idle' has only the 'white' box (shell). I prefer the third 'idle' because it's easy to use.

Step #5–Start using your Python shell.

You can now start typing Python functions, using the shell above.

You may have noticed that there are various entries to the contents of each of the files that you have opened. You can click and open all of them, as you progress in learning more about your Python programming.

Python is a programming language that has been studied by students for several days or months. Thus, what's presented in this book are the basics for beginners.

The rest of illustrations will assume you are running the python programs in a Windows environment.

1. Start IDLE
2. Navigate to the File menu and click New Window
3. Type the following: `print ("Hello World!")`
4. On the file, menu click Save. Type the name of `myProgram1.py`
5. Navigate to Run and click Run Module to run the program.

The first program that we have written is known as the "Hello World!" and is

used to not only provide an introduction to a new computer coding language but also test the basic configuration of the IDE. The output of the program is “Hello World!” Here is what has happened, the Print() is an inbuilt function, it is prewritten and preloaded for you, is used to display whatever is contained in the () as long as it is between the double quotes. The computer will display anything written within the double quotes.

Practice Exercise: Now write and run the following python programs:

- ✓ `print(“I am now a Python Language Coder!”)`
- ✓ `print(“This is my second simple program!”)`
- ✓ `print(“I love the simplicity of Python”)`
- ✓ `print(“I will display whatever is here in quotes such as owyhen2589gdbnz082”)`

Now we need to write a program with numbers but before writing such a program we need to learn something about Variables and Types.

Remember python is object-oriented and it is not statically typed which means we do not need to declare variables before using them or specify their type. Let us explain this statement, an object-oriented language simply means that the language supports viewing and manipulating real-life scenarios as groups with subgroups that can be linked and shared mimicking the natural order and interaction of things. Not all programming languages are object oriented, for instance, Visual C programming language is not object-oriented. In programming, declaring variables means that we explicitly state the nature of the variable. The variable can be declared as an integer, long integer, short integer, floating integer, a string, or as a character including if it is accessible locally or globally. A variable is a storage location that changes values depending on conditions.

For instance, number1 can take any number from 0 to infinity. However, if we specify explicitly that `int number1` it then means that the storage location will only accept integers and not fractions for instance. Fortunately or unfortunately, python does not require us to explicitly state the nature of the storage location (declare variables) as that is left to the python language itself to figure out that.

Before tackling types of variables and rules of writing variables, let us run a simple program to understand what variables when coding a python program

are.

- ✓ Start IDLE
- ✓ Navigate to the File menu and click New Window
- ✓ Type the following:

```
num1=4
num2=5
sum=num1+num2
print(sum)
```

- ✓ On the file, menu click Save. Type the name of myProgram2.py
- ✓ Navigate to Run and click Run Module to run the program.

The expected output of this program should be “9” without the double quotes.

Discussion

At this point, you are eager to understand what has just happened and why the `print(sum)` does not have double quotes like the first programs we wrote. Here is the explanation.

The first line `num1=4` means that variable `num1`(our shortened way of writing `number1`, first number) has been assigned 4 before the program runs.

The second line `num2=5` means that variable `num2`(our shortened way of writing `number2`, second number) has been assigned 5 before the program runs.

The computer interprets these instructions and stores the numbers given

The third line `sum=num1+num2` tells the computer that takes whatever `num1` has been given and add to whatever `num2` has been given. In other terms, sum the values of `num1` and `num2`.

The fourth line `print(sum)` means that display whatever `sum` has. If we put double quotes to `sum`, the computer will simply display the word `sum` and not the sum of the two numbers! Remember that cliché that computers are garbage in and garbage out. They follow what you give them!

Note: `+` is an operator for summing variables and has other uses.

Now let us try out three exercises involving numbers before we explain types



of variables and rules of writing variables so that you get more freedom to play with variables. Remember variables values vary for instance num1 can take 3, 8, 1562, 1.

Follow the steps of opening Python IDE and do the following:

✓ The output should be 54

```
num1=43
```

```
num2=11
```

```
sum=num1+num2
```

```
print(sum)
```

✓ The output should be 167

```
num1=101
```

```
num2=66
```

```
sum=num1+num2
```

```
print(sum)
```

✓ The output should be 28

```
num1=9
```

```
num2=19
```

```
sum=num1+num2
```

```
print(sum)
```

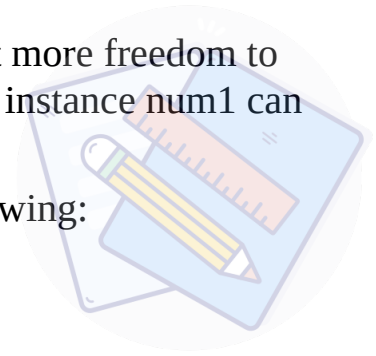
1. Variables

We have used num1, num2, and sum and the variable names were not just random, they must follow certain rules and conventions. Rules are what we cannot violate while conventions are much like the recommended way. Let us start with the rules:

The Rules of When Naming Variables in Python

1. Variable names should always start with a letter or an underscore, i.e.

```
num1
```



`_num1`

2. The remaining part of the variable name may consist of numbers, letters, and underscores, i.e.

`number1`

`num_be_r`

3. Variable names are case sensitive meaning that capital letters and non-capital letters are treated differently.

`Num1` will be treated differently with `num1`.



Practice Exercise

Write/suggest five variables for:

- ✓ Hospital department.
- ✓ Bank.
- ✓ Media House.

Given `scri=75`, `scr4=9`, `sscr2=13`, `Scr=18`

- ✓ The variable names in above are supposed to represents scores of students. Rewrite the variables to satisfy Python variable rules and conventions.

2. Conventions When Naming Variables in Python

As earlier indicated, conventions are not rules per se are the established traditions that add value and readability to the way we name variables in Python.

- ❖ Uphold readability. Your variables should give a hint of what they are handling because programs are meant to be read by other people other than the person writing them.

`number1` is easy to read compared to `n1`. Similarly, `first_name` is easy to read compared to `firstname` or `firstName` or `fn`. The implication of all these is that both are valid/acceptable variables in python but the convention is forcing us to write them in an easy to read form.

❖ Use descriptive names when writing your variables. For instance, number1 as a variable name is descriptive compared yale or mything. In other words, we can write yale to capture values for number1 but the name does not outrightly hint what we are doing. Remember when writing programs; assume another person will maintain them. The person should be able to quickly figure out what the program is all about before running it.

❖ Due to confusion, avoid using the uppercase 'O', lowercase letter 'l' and the uppercase letter 'I' because they can be confused with numbers. In other terms, using these letters will not be a violation of writing variables but their inclusion as variable names will breed confusion.

Practice Exercise 1

Re-write the following variable names to (1) be valid variable names and follow (2) conventions of writing variable names.

- ✓ 23doctor
- ✓ line1
- ✓ Option3
- ✓ Mydesk
- ✓ #cup3

Practice Exercise 2

Write/Suggest variable names that are (1) valid and (2) conventional.

- ✓ You want to sum three numbers.
- ✓ You want to store the names of four students.
- ✓ You want to store the names of five doctors in a hospital.

3. Keywords and Identifiers in Python Programming Language

At this point, you have been wondering why you must use print and str in that manner without the freedom or knowledge of why the stated words have to be written in that manner. The words print and str constitute a special type of words that have to be written that way always. Each programming language has its set of keywords. In most cases, some keywords are found across several programming languages. Keywords are case sensitive in python

meaning that we have to type them in their lowercase form always. Keywords cannot be used to name a function (we will explain what it is later), name of a variable.

There are 33 keywords in Python and all are in lowercase save for None, False, and True. They must always be written as they appear below:

Note: The print() and str are functions, but they are inbuilt/preloaded functions in Python. Functions are a set of rules and methods that act when invoked. For instance, the print function will display output when activated/invoked/called. At this point, you have not encountered all of the keywords, but you will meet them gradually. Take time to skim through, read and try to recall as many as you can.

Practice Exercise

Identify what is wrong with the following variable names (The exercise requires recalling what we have learned so far)

- ✓ for=1
- ✓ yield=3
- ✓ 34ball
- ✓ m

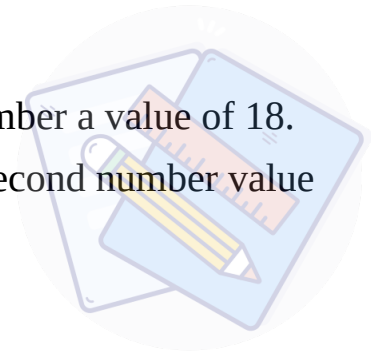
4. Comments and Statements

Statements in Python

A statement in Python refers to instructions that a Python interpreter can work on/execute. An example is str='I am a Programmer' and number1=3. A statement having an equal sign(=) is known as an assignment statement. They are other types of statements such as the if, while, and for which will be handled later.

Practice Exercise

- ✓ Write a Python statement that assigns the first number a value of 18.
- ✓ Write a programming statement that assigns the second number value of 21.
- ✓ What type of statements are a. and b. above?



5. Multi-Line Python Statement

It is possible to spread a statement over multiple lines. Such a statement is known as a multi-line statement. The termination of a programming statement is denoted by new line character. To spread a statement over several lines, in Python, we use the backslash (\) known as the line continuation character. An example of a multi-line statement is:

```
sum=3+6+7+\n    9+1+3+\n    11+4+8
```

The example above is also known as an explicit line continuation. In Python, the square brackets [] denotes line continuation similar to parenthesis/round brackets (), and lastly braces {}.

The above example can be rewritten as

```
sum=(3+6+7+\n    9+1+3+\n    11+4+8)
```

Note: We have dropped the backslash(\) known as the line continuation character when we use the parenthesis(round brackets) because the parenthesis is doing the work that the line continuation \ was doing.

Question: Why do you think multi-line statements are necessary we can simply write a single line and the program statement will run just fine?

Answer: Multi-line statements can help improve formatting/readability of the entire program. Remember, when writing a program always assume that it is other people who will use and maintain it without your input.

Practice Exercise:

Rewrite the following program statements using multi-line operators such as the `\`, `[]`, `()` or `{}` to improve readability of the program statements.

- `total=2+9+3+6+8+2+5+1+14+5+21+26+4+7+13+31+24`
- `count=13+1+56+3+7+9+5+12+54+4+7+45+71+4+8+5`

Semicolons are also used when creating multiple statements in a single line. Assume we have to assign and display the age of four employees in a python program. The program could be written as:

`employee1=25; employee2=45; employee3=32; employee4=43.`

6. Indentation in Python

Indentation is used for categorization program lines into a block in Python. The amount of indentation to use in Python depends entirely on the programmer. However, it is important to ensure consistency. By convention, four whitespaces are used for indentation instead of using tabs. For example:

Note: We will explain what kind of program of this is later.

Indentation in Python also helps make the program look neat and clean. Indentation creates consistency. However, when performing line continuation indentation can be ignored. Incorrect indentation will create an indentation error. Correct python programs without indentation will still run but they might be neat and consistent from human readability view.

7. Comments in Python

When writing python programs and indeed any programming language, comments are very important. Comments are used to describe what is happening within a program. It becomes easier for another person taking a look at a program to have an idea of what the program does by reading the comments in it. Comments are also useful to a programmer as one can forget the critical details of a program written. The hash (`#`) symbol is used before writing a comment in Python. The comment extends up to the newline character. The python interpreter normally ignores comments. Comments are

meant for programmers to understand the program better.

Example

- Start IDLE
- Navigate to the File menu and click New Window
- Type the following:

```
#This is my first comment
```

```
#The program will print Hello World
```

```
Print('Hello World') #This is an inbuilt function to display
```

- On the file, menu click Save. Type the name of myProgram5.py

Navigate to Run and click Run Module to run the program

Practice Exercise

This exercise integrates most of what we have covered so far.

- ✓ Write a program to sum two numbers 45, and 12 and include single line comments at each line of code.
- ✓ Write a program to show the names of two employees where the first employee is “Daisy” and the second employee is “Richard”. Include single comments at each line of code.
- ✓ Write a program to display the student registration numbers where the student names and their registration are: Yvonne=235, Ian=782, James=1235, Juliet=568.

- **Multi-Line Comments**

Just like multi-line program statements we also have multi-line comments. There are several ways of writing multi-line comments. The first approach is to type the hash (#) at each comment line starting point.

For Example

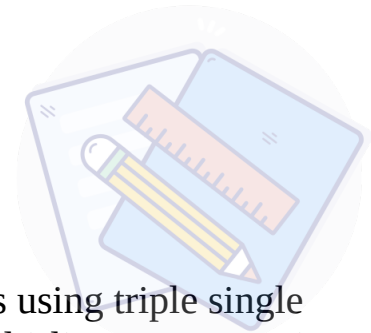
Start IDLE.

Navigate to the File menu and click New Window.



Type the following:

```
#I am going to write a long comment line  
#the comment will spill over to this line  
#and finally end here.
```



The second way of writing multi-line comments involves using triple single or double quotes: `'''` or `"""`. For multi-line strings and multi-line comments in Python, we use the triple quotes. Caution: When used in docstrings they will generate extra code but we do not have to worry about this at this instance.

Example:

Start IDLE.

Navigate to the File menu and click New Window.

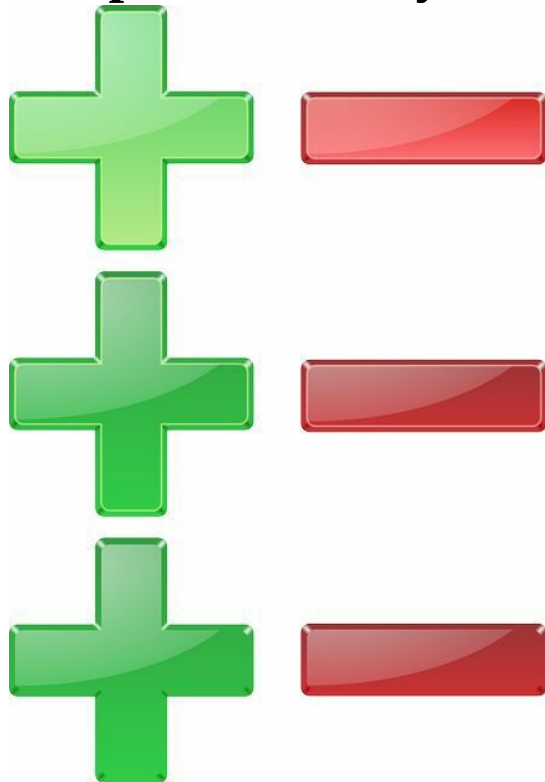
Type the following:

```
'''This is also a great i  
illustration of  
a multi-line comment in Python'''
```

- **Summary**

Variables are storage locations that a user specifies before writing and running a python program. Variable names are labels of those storage locations. A variable holds a value depending on circumstances. For instance, doctor1 can be Daniel, Brenda or Rita. Patient1 can be Luke, William or Kelly. Variable names are written by adhering to rules and conventions. Rules are a must while conventions are optional but recommended as they help write readable variable names. When writing a program, you should assume that another person will examine or run it without your input and thus should be well written. In programming, declaring variables means that we explicitly state the nature of the variable. The variable can be declared as an integer, long integer, short integer, floating integer, a string, or as a character including if it is accessible locally or globally. A variable is a storage location that changes values depending on conditions. Use descriptive names when writing your variables.

Chapter 4 The Python Operators

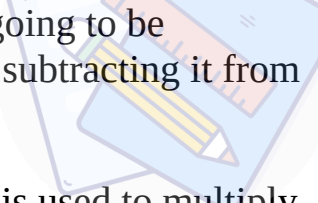


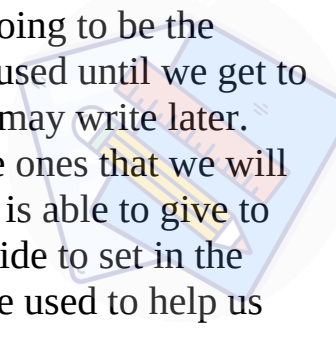
While we are here, we want to look at the topic of the Python operators and what these are able to do for some of our codings. As we go through some of the codings that need to be done as a beginner throughout this guidebook, you will find that these operators are pretty common and we are going to use them on a regular basis. There are actually a number of operators, but we are able to split them up into a few different types based on our needs. Some of the different operators that we are able to work with will include:

1. *The arithmetic operators.* These are the ones that will allow you to complete some mathematical equations inside of your code, such as addition or subtraction. You can simply add together two operands in the code, or two parts of the code together, subtract them, multiply them, and divide them and work from there. These can be used in many of the different codes that you would want to write along the way. Some of the options that you will be able to use when it comes to the arithmetic operators will include:

1. (+): this is the addition operator and it is responsible for adding

together both of your values.

- 
2. (-): this is the subtraction operator and it is going to be responsible for taking the right operand and subtracting it from the left.
 3. (*): this is the multiplication operator and it is used to multiply two or more values in the equation.
 4. (/): this is the division operator and it is going to divide the value of the left operand from that on the right and gives you the answer.
 2. *Comparison operators:* We are also able to work with the comparison operators. These are going to be a good option when we would like to take two or more parts of the code, such as the statements or the values, and then compare them with one another. This one is going to rely on the Boolean expressions to help us get things done because it will allow us to look at true or false answers along the way. so, the statements that you are comparing will either be true or they will be false based on how they compare.
 1. (>=): this one means to check if the operand on the left is greater than or equal to the value of the one on the right.
 2. (<=): this one means to check if the value of the operand on the left side is less than or the same/equal to the one on the right.
 3. (>): this one means to check whether the values of the left side are greater than the value on the right side of the code.
 4. (<): this one means to check whether the values of the left side are less than the values that are on the right side.
 5. (!=): this is not equal to operator.
 6. (==): this one is equal to operator.

- 
3. *The logical operators:* The next option is going to be the logical operators. These ones are often not used until we get to some of the more advanced codes that you may write later. These types of operators are going to be the ones that we will need to use to evaluate the input that a user is able to give to you with any of the conditions that you decide to set in the code, and there are three of them that can be used to help us with this goal.
 1. Or: with this one, the compiler is going to evaluate x and if false, it will then go over and evaluate y. If x ends up being true, the compiler is going to return the evaluation of x.
 2. And: if x ends up being the one that is false, the compiler is going to evaluate it. If x ends up being true, it will move on and evaluate y.
 3. Not: it ends up being false, the compiler is going to return True. But if x ends up being true, the program will return false.
 4. *Assignment operators:* And finally, we are going to take a look at the assignment operators. These are going to be ones that are used all of the time, and we have actually already seen them in some coding. The assignment operator is simply going to be an equal sign because it takes a value and assigns it over to one of your variables. You can use any variable or value that you want here, and do the assigning as much as you would like, just make sure that the equal sign is in the right place.

These operators are going to be important to the codes that you write for many reasons, and it often depends on what you are hoping to accomplish out of some of the codes that you are writing along the way. Take some time to look these over and learn a bit more about these operators and how you are able to use them in some of your own codings along the way for the best results.

Chapter 5 Basic Data Types in Python

What is Data Types?

We know that variables are used to store the data we use in our Python programs. But, how do these variables actually work?

Each system has a Random-Access Memory (RAM), which is used to store temporary data. When we declare a variable as such:

```
x = 10
```

What happens in the back is that Python reserves a small segment of our RAM for us and it is used to store that value.

But, how does the Python interpreter decide how much space to allocate for our variable? This allocation of space is dependent on the size of the variable, which in turn, is dependent on the *data type* of the variable.

To make our program more efficient and faster, we need to assign memory carefully and use the right data types for our variables.

Booleans

One of the simplest data types in Python and other programming languages is the bool or the Boolean data type.

A Boolean variable can only have two possible values; True or False. Such variables are used commonly to test for conditions.

For example, if you wish to know the state of the web server on in your Python program, you could write something like:

1. `webServerOpen = True`
2. `lockdownState = False`
3. `underMaintenance = False`

If you are unsure about the data type of a variable, Python allows you to easily access the data type using the **type()** function.

If you run the following code next,

1. `print(type(webServerOpen))`
2. `print(type(lockdownState))`



You will get the following output:

```
>>> type(webServerOpen)
<class 'bool'>
>>> type(lockdownState)
<class 'bool'>
>>> |
```



The function returns the data type of the variable which is sent inside its parenthesis. The values inside the parenthesis are called *arguments*. Here, 'bool' represents a Boolean variable.

Strings

Remember, in our first program, when we printed "Hello World!". We called it a phrase or text which is not built into Python but can be used to convey messages from the user. This text or similar phrases are called **Strings** in programming languages.

Strings are nothing more than a series of characters. Whenever you see something enclosed with double quotation marks or single quotation marks, the text inside is considered a string.

For example, these two variables that I've just declared are both Strings and perfectly valid.

1. `a = "This is an example of a string and I'm using double quotation marks"`
2. `b = 'This is also an example of strings and now, I am using single quotation marks'`

What does this flexibility offer? If you take a look at the variable 'a'. I've used a contraction and a single quotation mark inside my string (which uses a double quotation mark).

Here are a few examples of quotes and sentences where I've used contractions:

1. `string1 = "I'm an amazing person"`
2. `string2 = 'I told him, "Work hard!"'`

String Methods

Methods are simple actions that we can ask the Python interpreter to take for

us. Methods require variables or a piece of data on which they can work and produce the desired result. Let's first write some code and we'll explore it as we go.

Changing the Case of Strings:

Create a new file called `stringMethods.py` (or alternatively use IDLE) and run the following code:

1. `username = 'john doe'`
2. `print(username.title())`

If you run the code, you should get this output:

John Doe

In the first line, we declare our variable and assign a string to it. In the second line, we start by printing something. In the parenthesis of the print function, we use the method **title()** on our variable.

Methods let us perform an action on our data. In this case, the title method displays each word of our variable in capital letters. Since the method acts on our variable, we connect the variable to the method using a period or '.' symbol.

Methods are identified by the set of parentheses and take in additional information to act on the variables they are connected to. Since the title method doesn't need any other information, the parenthesis is empty.

To deal with character casing, several other methods are available. Let's test the **upper() and lower() method** by adding the following code to your file. Then, run it:

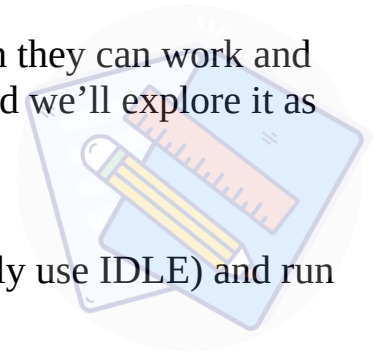
```
username2 = 'John Doe'
print(username.upper())
print(username.lower())
```

The output of these methods should be:

JOHN DOE

john doe

So, the upper method capitalizes each word, whereas, the lower method converts all characters to lowercase letters.



Concatenating and Combining Strings:

At times your program might require users to input their first name, middle name, and then the last name. But, when you display it back, these names are combined together.

So, how do we concatenate strings? It's simple. The "+" symbol is called the concatenation operator and is used to combine strings together.

Take a look at the following code:

```
1. firstName = "John"
2. middleName = "Adam"
3. lastName = "Doe"
4. fullName = firstName + middleName + lastName
5. print(fullName)
```

What's the output? **John Adam Doe**

If you want to run some other methods on the output, feel free. Here's a program which runs some tests on the variables:

```
firstName = "John"
middleName = "Adam"
lastName = "Doe"
fullName = firstName + middleName + lastName
print(fullName.lower())
message = "The manager, " + fullName.title() + ", is a good person."
```

What would be the output? For the first print statement:

john adam doe – All lowercase letters. For the second print statement:

The manager, John Adam Doe, is a good person. – Concatenated a string on which a method was applied.

So, you can use these methods on every string in your program and it will output just fine.

Adding Whitespaces:

Whitespaces refer to the characters which are used to produce spacing in between words, characters, or sentences using tabs, spaces, and line breaks. It's better to use properly use whitespaces so the outputs are readable for users.

To produce whitespaces using tabs, use this combination of characters `'\t'`. Here's an example to show you the difference with and without the tab spacing.

1. `print("This is without a tab spacing!")`
2. `print("\t This is with a tab spacing!")`

Notice the difference in the output? Here's the output of the two lines above.

This is without a tab spacing

This is with a tab spacing

To add new text to new lines, you can add a line break using the combination `'\n'`. Here's an example to show you the difference a line break adds:

1. `print('I love the following weathers: SummerWinter')`
2. `print('I love the following weathers:\nSummer\nWinter')`

Once again, which output is more readable? Take a look at the outputs.

1. **I love the following weathers: SummerWinter**
2. **I love the following weathers:**

Summer

Winter

You can use both tabs, spaces, and newlines together and it would work just fine. Here's an example program to cater to both new lines and tabs being used together.

```
print("I love these weathers:\n\tSummer\n\tWinter")
```

Here's the output of the command:

I love these weathers:

Summer

Winter

Removing or Stripping the Whitespace:

To you, the phrases 'code' and 'code' look the same right? To the computer, however, they are two very different strings. It treats this extra space as the part of the string; until, told otherwise.

Where does this produce a problem? When you are about to compare two strings to check if 'code' and 'code' are equal, you will think it's True, but the interpreter will always return a False.

If you think your string has extra space to the right side of the string, use the **rstrip()** method. Let's take a look at an example. I'll declare a variable, show you the whitespace, and then remove it to see the difference. Here's the code:

```
favoriteWeather = "Summer "  
print(favoriteWeather + "is lovely")  
favoriteWeather.rstrip()  
print(favoriteWeather + "is lovely")
```

If you run these statements, you might expect the following output:

Summeris lovely

Summer is lovely

Did you expect this output? Here's the actual output:

Summeris lovely

Summeris lovely

This is because the effect of **rstrip()** is temporary. Sure, it does cause the removal of whitespaces, but, to ensure it is permanent. You need to reassign it to the original variable. Here's the same example, continued to explain this concept:

```
favoriteWeather = "Summer "  
print(favoriteWeather + " is lovely")
```



```
favoriteWeather = favoriteWeather.rstrip()  
print(favoriteWeather + " is lovely")
```

In line 3, rather than using the method only, I assigned it to the original variable which causes the permanent removal of whitespaces. Here's the new output:

Summeris lovely

Summer is lovely – After the removal of spaces

If you think your string has extra space to the left side of the string, use the **lstrip()** method. Here's a small program to test the **lstrip()** method:

```
weather = "  cold"  
print("It is " + weather)  
weather = weather.lstrip()  
print("It is " + weather)
```

Here's the output of the lines:

It is cold

It is cold

If you wish to remove whitespaces from both ends, use the **strip()** method. Here's an example to show you the removal of whitespaces from both ends:

```
message = "  It's so cold in London  "  
print(message)  
message = message.strip()  
print(message)
```

Here's the output of these statements:

It's so cold in London(end)

It's so cold in London (end)

If you want to temporarily notice the effects of strip, you can use the method in a print function and see the effects. These strip methods are extremely useful in real-life applications when user data has to be cleaned and manipulated.

Numbers



Numbers are an integral part of our daily lives and computers make use of the two binary digits; 0 and 1, a lot. Python has several different ways to treat numbers. Here are a few common types we'll be looking at:

Integers

Integers are numbers that are written without fractional parts. In Python, these numbers have the type **int** (just like bool for Booleans).

Here are a few integers and their declaration in Python:

- `a = 2424`
- `b = 10101`
- `c = 9040`

Floats

If you declare a number with a decimal point, Python will automatically consider it a floating-point number or a float.

All operations you could perform on integers, they can also be performed on floating point numbers. If you run the type function on floats, you'll get a type **'float'**.

Here's a program to show some operations on floats:

```
w = 1.2 * 0.3
print(w)
x = 10.4 + 1.6
print(x)
y = 10.555 + 22.224
print(y)
z = 0.004 + 0.006
print(z)
```

Here is the output to all four print statements:

0.36

12.0

32.778999999999996

0.01

Type Casting: What Is It?

Another fact about Python is that it is a dynamically-typed language.

A weakly-typed or dynamically-typed language doesn't associate a data type with your variable at the time you are typing your code. Rather, the type is associated with the variable at run-time. Not clear? Let's see an example.

```
x = 10
```

```
x = "I am just a phrase"
```

```
x = 10.444
```

```
x = True
```

When you run this code, it'll perform its desired actions correctly. Let's see what happens with the variable 'x' though. We've written four statements and assigned different values to 'x'.

On run-time (when you interpret or run your program), on line 1, 'x' is an integer. On line 2, it's a string. On line 3, it's a float, and finally, it's a Boolean value.

However, through typecasting, we can manually change the types of each variable. The functions we'll be using for that purpose are **str()**, **int()**, and **float()**.

Let's expand the same example:

```
x = 10
```

```
x = float(x)
```

```
print(type(x))
```

```
x = "I am just a phrase"
```

```
print("x: " + x)
```

```
print(type(x))
```

```
x = 10.444
```

```
x = int(x)
```

```
print(type(x))
```

```
x = False
```

```
x = int(x)
print(x)
```



In this program, we've used everything covered in the last few lessons. All data types and converted them using our newly learned function.

In the first case, x is converted into a float and the type function does verify that for us. Secondly, the string is still a string since it can't be converted into numbers, int or float. Thirdly, we convert the float into an integer.

As an added exercise, if you print the newly changed value of the third case, you'll see that the value of x is: **10**. This is because the type is now changed and the values after the decimal point are discarded.

In the fourth case, we print the value of x which is False. Then, we change its value to an integer. Here, something else comes up. The output? **0**.

It's because, in Python, the value of True is usually any non-zero number (typically 1) and for False, it's 0. So, their integer conversions yield 1 and 0 for True and False respectively.

Comments

Comments are text phrases that are put in the code to make it understandable and readable for other coders, readers, and programmers.

Why Are Comments Important?

Comments are very important, especially when you're working with other programmers and they'll be reviewing your code sooner or later. Through comments, you can write a small description of the code and tell them what it does.

Also, if you have other details or personal messages which are relevant to the code, you can put them there, since the interpreter doesn't catch them.

How to Write Comments?

In python, there are two ways to write comments, and we'll be exploring both of them. For our first method, you can use the “#” symbol in front of the line you wish to comment. Here, take a look at this code:

```
# This line is a comment
# Count is a simple variable
count = 15
```

```
print(count)
```

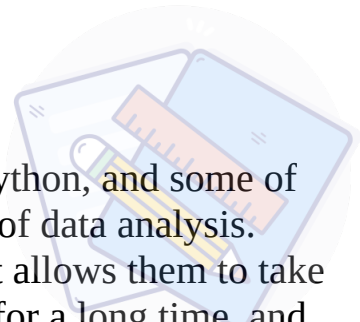
If you run this code, the output will be **15**. This is because the comments lines (starting with #) are not run at all.

Now, this method is fine if your commented lines are less i.e. do not span over multiple lines. But, if they do, hashing all of them is a waste of time. For our second comment, we'll enclose our commented lines is **three single quotation marks (‘’)** and close it with **three quotation marks as well**. Here's an example:

1. '''
2. This comment spans
3. on multiple lines.
4. '''
5. count = 15
6. print(count)

Notice, we have to close our multi-line comment, unlike the single line comment.

Chapter 6 Data Analysis with Python



Another topic that we need to explore a bit here is how Python, and some of the libraries that come with it, can work with the process of data analysis. This is an important process for any businesses because it allows them to take all of the data and information they have been collecting for a long time, and then can put it to good use once they understand what has been said within the information. It can be hard for a person to go through all of this information and figure out what is there, but for a data analyst who is able to use Python to complete the process, it is easy to find the information and the trends that you need.

The first thing that we need to look at here though is what data analysis is all about. Data analysis is going to be the process that companies can use in order to extract out useful, relevant, and even meaningful information from the data they collect, in a manner that is systematic. This ensures that they are able to get the full information out of everything and see some great results in the process. There are a number of reasons that a company would choose to work on their own data analysis, and this can include:

1. Parameter estimation, which helps them to infer some of the unknowns that they are dealing with.
2. Model development and prediction. This is going to be a lot of forecasting in the mix.
3. Feature extraction which means that we are going to identify some of the patterns that are there.
4. Hypothesis testing. This is going to allow us to verify the information and trends that we have found.
5. Fault detection. This is going to be the monitoring of the process that you are working on to make sure that there aren't any biases that happen in the information.

One thing that we need to make sure that we are watching out for is the idea of bias in the information that we have. If you go into the data analysis with

the idea that something should turn out a certain way, or that you are going to manipulate the data so it fits the ideas that you have, there are going to be some problems. You can always change the data to say what you would like, but this doesn't mean that you are getting the true trends that come with this information, and you may be missing out on some of the things that you actually need to know about.

This is why a lot of data analysts will start this without any kind of hypothesis at all. This allows them to see the actual trends that come with this, and then see where the information is going to take you, without any kind of slant with the information that you have. This can make life easier and ensures that you are actually able to see what is truly in the information, rather than what you would like to see in that information.

Now, there are going to be a few different types of data that you can work with. First, there is going to be the deterministic. This is going to also be known as the data analysis that is non-random. And then there is going to be the stochastic, which is pretty much any kind that is not going to fit into the category of deterministic.

1. The Data Life Cycle

As we go through this information, it is important to understand some of the different phases that come with the data life cycle. Each of these comes together to ensure that we are able to understand the information that is presented to us and that we are able to use all of the data in the most efficient and best way possible.

There are a few stages that are going to come with this data life cycle, and we are going to start out with some of the basics to discuss each one to help us see what we are able to do with the data available to us. First, we work with data capture. The first experience that an individual or a company should have with a data item is to have it pass through the firewalls of the enterprise. This is going to be known as the Data Capture, which is basically going to be the act of creating values of data that do not exist yet and have never actually existed in that enterprise either. There are three ways that you can capture the

data including:

1. Data acquisition: This is going to be the ingestion of data that is already existing that was produced by the organization but outside of the chosen enterprise.
2. Data entry: This is when we are dealing with the creation of new data values to help with the enterprise and it is done by devices or human operators that can help to generate the data needed.
3. Signal reception: This is where we are going to capture the data that a device has created with us, typically in the control system, but can be found in the Internet of Things if we would like.

The next part is going to be known as Data Maintenance. This is going to be where you supply the data to points at which data synthesis and data usage can occur in the next few steps. And it is best if you are able to work out the points so that they are going to be ready to go in this kind of phase.

What we will see during the data maintenance is that we are working to process the data, without really working to derive any value out of it yet. This is going to include integration changed data, cleansing, and making sure that the data is in the right format and as complete as possible before we get started. This ensures that no matter what method or algorithm you choose to work with here, you are going to be able to have the data ready to go.

Once you have been able to maintain the data and get it all cleaned up, it is time to work on the part known as data synthesis. This is a newer phase in the cycle and there are some places where you may not see this happen. This is going to be where we create some of the values of data through inductive logic, and using some of the data that we have from somewhere else as the input. The data synthesis is going to be the arena of analytics that is going to use modeling of some kind to help you get the right results in the end.

Data usage comes next. This data usage is going to be the part of the process

where we are going to apply the data as information to tasks that the enterprise needs to run and then handle the management on its own. This would be a task that normally falls outside of your life cycle for the data. However, data is becoming such a central part of the model for most businesses and having this part done can make a big difference.

For example, the data itself can be a service or a product, or at least part of this service or product. This would then make it a part of the data usage as well. The usage of the data is going to have some special challenges when it comes to data governance. One of these is whether it is legal to use the data in the ways that most people in business would like. There could be some issues like contractual or regulatory constraints on how we can use this data and it is important that these are maintained as much as possible.

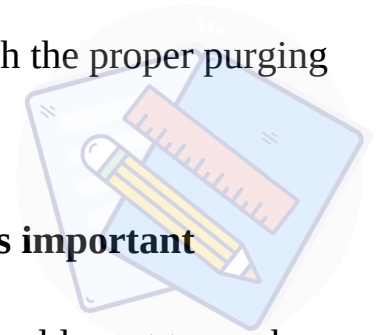
Once we have figured out the data usage, it is time to move on to data publication implying that that has been sent outside of the firm or enterprise.

Next on the list is the data archival. We will see that the single data value that we are working with can sometimes experience a lot of different rounds of usage and then publication, but eventually, it is going to reach the very end of its life. The first part of this means that we need to be able to take the value of the data and archive it. When we work on the process of Data Archival, it is going to mean that we are copying the data to an environment where it is stored in case we need it again, in an active production environment, and then we will remove the data from all of those active environments as well.

This kind of archive for the data is simply going to be a place where the data is stored, but where no publication, usage, or maintenance is going to happen. If necessary, it is possible to take any of the data that is in the archive and bring it back out to use again.

And finally, we reach the part of data purging. This is going to be the end that comes with our single data value and the life cycle that it has gone through. Data purging is going to be when we remove every copy of data from the enterprise. If possible, you will reach this information through the archive. If there is a challenge from Data Governance at this point, it is just there to

prove that the information and the data have gone through the proper purging procedure at that time.



2. Working with data analysis and why it is important

With this in mind, we need to pay attention to why we would want to work on data analysis to start with? Do we really need to be able to look through all of this information to find the trends, or is there another method? Let's look at an example of what can happen when we do this data analysis and why you would want to use it.

Let's consider that we are looking at a set of data that includes information about the weather that occurred across the globe between the years 2015 to 2018. We are also going to have information that is base don the country between these years as well. So, there is going to be a percentage of ran within that country and we are going to have some data that concerns this in our set of data as well.

Now, what if you would like to go through all of that data, but you would like to only take a look at the data that comes with one specific country. Let's say that you would like to look at America and you want to see what percentage of rain it received between 2016 and 2017. Now, how are you going to get this information in a quick and efficient manner?

What we would need to do to make sure that we were able to get ahold of this particular set of data is to work with the data analysis. There are several algorithms, especially those that come from machine learning, that would help you to figure out the percentage of rain that America gets between 2016 to 2017. And this whole process is going to be known as what data analysis is really all about.

3. The Python Panda Library

When it comes to doing some data analysis in Python, the best extension that

you can use is Pandas. This is an open-sourced library that works well with Python and it is going to provide you with a high level of performance, data structures that are easy for even a beginner to use, and tools to make data analysis easy with Python. There are a lot of things to enjoy about this language, and if you want to be able to sort through all of the great information that you have available with the help of Python, then this is the library that you need to work with.

There are a lot of things that you can enjoy when it comes to working on the Python library. First off, this is one of the most popular and easy to use libraries when it comes to data science and it is going to work on top of the NumPy library. The one thing that a lot of coders are going to like about working with Pandas is that it is able to take a lot of the data that you need, including a SQL database or a TSV and CSV file, and will use it to create an object in Python. This object is going to have columns as well as rows called the data frame, something that looks very similar to what we see with a table in statistical software including Excel.

There are many different features that are going to set Pandas apart from some of the other libraries that are out there. Some of the benefits that you are going to enjoy the most will include:

1. There are some data frames or data structures that are going to be high level compared to some of the others that you can use.
2. There is going to be a streamlined process in place to handle the tabular data, and then there is also a functionality that is rich for the time series that you want to work with.
3. There is going to be the benefit of data alignment, missing data-friendly statistics, merge, join, and groupby methods to help you handle the data that you have.
4. You are able to use the variety of structures for data in Pandas, and you will be able to freely draw on the functions that are present in SciPy and NumPy to help make sure manipulation and other work can be done the way that you want.

Before we move on from here, we also need to have a good look at what some of the types of data are when it comes to Pandas. Pandas is going to be well suited when it comes to a large amount of data and will be able to help you sort through almost any kind of data that you would like. Some of the data types that are going to be the most suitable for working with the Pandas library with Python will include:

1. Any kind of tabular data that is going to have columns that are heterogeneously typed.
2. Arbitrary matrix data that has labels for the columns and rows.
3. Unordered and ordered time-series data
4. Any other kinds of sets of data that are statistical and observational.

Working with the Pandas library is one of the best ways to handle some of the Python codings that you want to do with the help of data analysis. As a company, it is so important to be able to go through and not just collect data, but also to be able to read through that information and learn some of the trends and the information that is available there. being able to do this can provide your company with some of the insights that it needs to do better, and really grow while providing customer service.

There are a lot of different methods that you can use when it comes to performing data analysis. And some of them are going to work in a different way than we may see with Python or with the Pandas library. But when it comes to efficiently and quickly working through a lot of data, and having a multitude of algorithms and more that can sort through all of this information, working with the Python Pandas is the best option.

Chapter 7 Conditional Statements

If Statements

More often than not, you're faced with a situation where you have to decide something and then, a few things happen in response to your decision.

Similarly, programming languages also allow you to write conditional tests, with which, you can check a condition and make responses according to it.

Let's take a real-life example, and then code it. **If** the stove is on, **then** close it. **If** it is not on, **then** do nothing.

If you take a look, we use the keyword '**if**' when we're trying to put forth a condition. If this, then that. Likewise, Python uses the **if** statement to allow you to make a decision based on something. That 'something' in our example was, whether or not the stove was on. Let's code it.

1. # Whether or not the stove is on.
2. stoveOn = True
3. # Is the stove on?
4. if stoveOn == True:
5. print("The stove is on! Close it.")

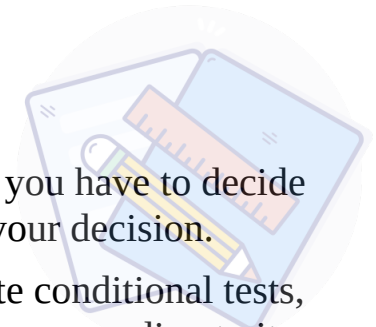
Firstly, we declared our variable and said, yes, the stove is on. Now, using the **if statement**, we see, if the value of our variable when compared to the Boolean value, True, yields a True, i.e. the stove is actually on and both of their values are the same.

Now to the syntactical part. Using if statement is simple. You can either write your condition after the **if keyword** or you could wrap it in a pair of parentheses. Like this:

```
if(stoveOn == True):
```

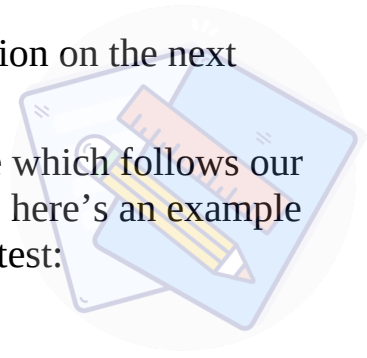
You can put in as many conditions as you want and use logical operators (and, or) to separate them and make a decision based on their values. Thirdly, after the condition, we use the **“:”** symbol to continue. Next, we tell what should our program do, if, the result of our conditional test is True. If not, it will completely ignore the code which follows this test. But, is it going to be the first following line only? Let's see.

Since in our case, it is True, we simply print a statement to close it right



away. But, here's something odd. Do you see an indentation on the next line?

This indentation is how we recognize the part of the code which follows our if statement. In this case, there's only one statement. But, here's an example with multiple lines of code and quite a few conditions to test:



1. # Whether or not the stove is on.
2. stoveOn = True
3. lightsOn = False
4. # Is the light on? Just check the stove.
5. if(lightsOn == True):
6. if(stoveOn == True):
7. print("The lights are on, just close the stove.")
8. # Is the light off? Open it, then, check the stove.
9. if(lightsOn == False):
10. print("First, open the lights!")
11. if(stoveOn == True):
12. print("Close the stove!")

Now, we test a few conditions and then make our decision. We added another variable which helps us make a decision.

Firstly, check if the lights are on. If their values are correct, I indented the code in the next line, which means, lines 7-8 are to be executed only if the statement on line 6 is True.

If not, all that code is neglected. If, however, line 6 yields a True but line 7 returns a False. You see, I indented the code on line 8 again. Which means, this is now, a part of the if statement on line 7 and will not be executed in this case.

Now, on to the second conditional test. If the lights aren't on, print a statement to turn the lights on. Then check for the stoves. See, how the code on line 12-13 is indented once. It means they belong to the first if statement and will be executed by the interpreter if the statement on line 11 yields a True.

Similarly, code on line 14 only executes if the if statement on line 13 is True. So, the **indentation in Python is important and must be well taken care of. If you don't indent your code properly, either your conditions won't output a result, or, you will get an error.**

'If-else' Statements

Now, what was our example again? Here's that sentence. **If** the stove is on, **then** close it. **If** it is not on, **then** do nothing.

You see, we catered the part where it says, if the stove is on, do this and that. But, what about the part where it isn't on? Let's take a look at this code. We expand on our first example.

```
1. # Whether or not the stove is on.  
2. stoveOn = False  
3.  
4. # Is the stove on?  
5. if stoveOn == True:  
6.     print("The stove is on! Close it.")  
7. else:  
8.     print("The stove is off!")
```

Now, we put a False in our variable and continued our tests. If the stove is on (True output from the if condition), just print that it is on. If it is not, we use the **else** clause, to say, do this instead.

Like our real-life situations. Do this, else do that. So, what happens here? When the result from the if statement on line 5 is False, it neglects all indented code (which was to follow if that if statement was True) and executes the else clause.

Now, logically, it states, **If the stoveOn is False, do this.** And, we simply print, that the stove is already off!

```
The stove is off!
```

'if-elif-else' Statements

Usually, when asked to write multiple conditions, you might write them in a similar fashion: If this, then do this. Else if this, then do that. Else (none of these), do something completely different.

See, how all these if-elseif-else clauses are linked to one single conditional statement? This is where the **elif or Else If** block comes in. If your variable, the one you wish to use for your conditional test, has many values, and needs to output differently for those values, you can put them in an if-elif-else block. This way, on the first true, no other condition gets executed. Or, elif gets executed, or the else clause.

We did just the same. We said, If the lights and stove are on, you just close the stove. **Else if (elif in Python)**, the lights are closed but the stove is on, turn the lights, close the stove. And further, continue with the else statement.

1. # Whether or not the stove is on.
2. stoveOn = True
3. lightsOn = False
4. # Is the light on? Just check the stove.
5. if(lightsOn == True and stoveOn == True):
6. print("The lights are on, just close the stove.")
7. elif (lightsOn == False and stoveOn == True):
8. print("Turn the lights on, close the stove!")
9. else:
10. print("Both the lights and the stove is on.")

```
Turn the lights on, close the stove!
```

The example also shows how you can run multiple conditions in an if-elif-else statement and base the output on all those.

Chapter 8 Loops – The Never-Ending Cycle

Imagine you are creating a program which asks the user to guess a number. The code should ideally run for three times before it could let the user know that they consumed their three chances and failed. Similarly, the program should be smart enough to know if the user guessed the right number, in which case, it would end the execution of the program by displaying “You guessed the right number!”

We use loops to address such situations. Loops are when an entire block of code continues to run over and over again, until the condition set is no longer valid. If you forget to set a condition, or if a condition is not properly defined, you may start an endless loop that will never cease, causing the program to crash completely.

Do not worry, your system will not crash. You can end the program by using the red/pink stop button that always magically appears after you hit the green run button.

There are essentially two types of loops we use in Python. The first one is the ‘while’ loop, and the second one is the ‘for’ loop.

The ‘While’ Loop

This type of loop runs a specific block of code for as long as the given condition remains true. Once the given condition is no longer valid, or turns to false, the block of code will end right away.

This is quite a useful feature as there may be codes which you may need to rely on to process information quickly. To give you an idea, suppose, you are to guess a number. You have three tries. You want the prompt to ask the user to guess the number. Once the user guesses the wrong number, it will reduce the maximum number of tries from three to two, inform the user that the number is wrong and then ask to guess another time. This will continue until either the user guesses the right number or the set number of guesses are utilized and the user fails to identify the number.

Imagine just how many times you would have to write the code over and over again. Now, thanks to Python, we just type it once underneath the ‘while’ loop and the rest is done for us.

Here’s how the syntax for the ‘while’ loop looks like:

while condition:

code

code

...

You begin by typing in the word 'while' followed by the condition. We then add a colon, just like we did for the 'if' statement. This means, whatever will follow next, it will be indented to show that the same is working underneath the loop or the statement.

Let us create a simple example from this. We start by creating a variable. Let's give this variable a name and a value like so:

```
x = 0
```

Nothing fun here, so let us add something to make it more exciting. Now, we will create a condition for a while loop. The condition would state that as long as x is equal to or less than 10, the prompt will continue to print the value of x. Here's how you would do that:

```
x = 0
```

```
while x <= 10:
```

```
    print(x)
```

Now try and run that to see what happens!

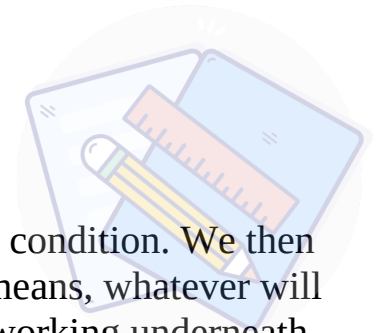
Your console is now bombarded with a never-ending loop of zeros. Why did that happen? If you look close enough at the code, we only assigned one value to our variable. There is no code to change the value or increase it by one or two, or any of that.

In order for us to create a variable that continues to change variable after it has printed the initial value, we need to add one more line to the code. Call it as the increment code, where x will increase by one after printing out a value. The loop will then restart, this time with a higher value, print that and then add one more. The loop will continue until x is equal to 10. The second it hits the value of 11, the interpreter will know that the condition no longer remains true or valid, and hence we will jump out of the loop.

```
x = 0
```

```
while x <= 10:
```

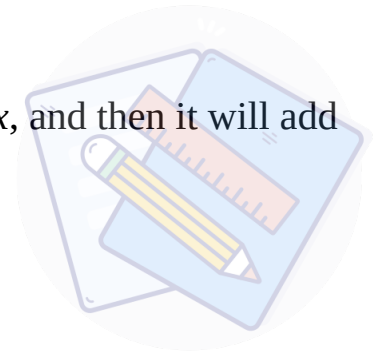
```
    print(x)
```



$x = x + 1$

The last line will execute and recall the current value of x , and then it will add one to the value. The result would look like this.

1
2
3
4
5
6
7
8
9
10



If you do not like things to add just like that, add a little print statement to say “The End” and that should do the trick.

I almost forgot! If you intend to add a print statement at the end, make sure you hit the backspace key to delete the indentation first.

Let’s make things a little more fun now, and to do that, we will be creating our very first basic game.

Let me paint the scenario first. If you like, pick up a pen and a paper, or just open notepad on your computer. Try and write down what you think is the possible solution for this.

The game has a secret number that the end-user cannot see. Let’s assume that the number is set to 19. We will allow the user to have three attempts to guess the number correctly. The game completes in a few possible ways:

1. The user guesses the number correctly before running out of lives.
2. The user runs out of the three chances and is unable to guess the number.
3. The user guesses the number on the final attempt.

Use your imagination and think what can be the possible code. Once ready, let us proceed to the actual coding for this game and see how this works out to be.

Hint: Use both a 'while' loop and an 'if' statement!

Well done for those who tried. There is no shame in failing to pull this off. I failed to do the same myself until I saw the solution and I practically kicked myself!

```
my_number = 19
guess = 0
max_guess = 3
while guess < max_guess:
    number = int(input("Guess the number: "))
    guess += 1
    if number == my_number:
        print("Wow! Look at you, genius!")
        break
    else:
        print("Nope! Not in a million years! Try again!")
    else:
        print("You ran out of chances")
```

“Wait! Why did you use an ‘else’ with the ‘while’ loop? I didn’t know if you can do that!”

Now you do! The ‘else’ is not just limited to ‘if’ statements, you can use it with while as well.

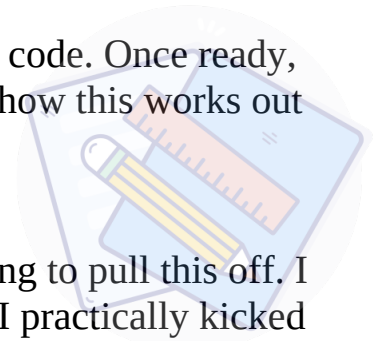
Here’s what the end result looks like:

All incorrect guesses

Guess the number: 1

Nope! Not in a million years! Try again!

Guess the number: 2



Nope! Not in a million years! Try again!

Guess the number: 3

Nope! Not in a million years! Try again!

You ran out of chances

Correct guess

Guess the number: 17

Nope! Not in a million years! Try again!

Guess the number: 18

Nope! Not in a million years! Try again!

Guess the number: 19

Wow! Look at you, genius!

Remember nested conditional statements? This is exactly that. The program begins by first understanding certain variables. See how I have named them to make it a little easier to read.

We gave 'guess' a value of zero to begin with. That is exactly what you need to do as the first attempt has not yet been registered by the system. Always begin such guesses/attempts from zero and then add increments. We then followed by setting an upper limit. We could have just written the same in this way:

```
while guess <= 3:
```

The problem with this would have been that the digit '3' was only recognizable by us. For any other programmer, this would not make any sense. Therefore, we replaced that with a variable so that it literally improves readability. Now, it reads like this:

“While guess is less than or equal to three:”

This is how you should always aim your codes to be. They should be readable and easy to understand by everyone.

“<= ” is yet another operator. Here, the values are either less than or equal to whatever the value of variable is on the other side.

We started by asking the user to guess, and that's what we need as an input. However, since it will be a whole number, we also converted the same into



an integer. After the user guessed the number, whether right or wrong, we immediately need the program to add a value of '1' to the number of guesses. This is where we used an increment. But, unlike what we did earlier, I changed a little and used the '+=' operator. It basically means to increase the value by whatever the digit you choose to write on the other side. If you are more comfortable using the previous method, it would work flawlessly as well.

Now, here's the twist. We used an 'if' statement to let the program know that if the user guesses the exact number, it should print out a message that is appropriate for the occasion. Otherwise, the 'else' condition will take place, as long as this is not the third and final guess.

Should the final guess wrong, the count will increase for the number of guesses and the while statement will no longer be true, in which case, the 'else' part of it will come into play and end the game.

The thing to notice here is the word 'break' that I used within the code. Go ahead and see what happens when you remove this. If you guess your numbers wrong, the code will work fine. And, if you end up inputting the right value, instead of ending the game, it will still go on until the third attempt is made.

To avoid that from happening, we use the 'break' statement to let the program know what to do if the condition above the break statement is met.

Now, there is almost nothing left about the 'while' loop, let us move to the 'for' loops. Slightly different to what you might expect, but interesting nonetheless.

The 'For' Loop

The 'while' loop executes whatever the code block that is written within multiple times, until the condition is no longer met or invalid. The 'for' loop is designed to "iterate over items of collections" and right away, that is causing some confusion.

Do not be intimidated by fancy words or technical language, once you see the loop in action, it will automatically start making sense.

To give it a little clear meaning, let us look at the example below:

```
for char in "Loops":
```

`print(char)`

To create a 'for' loop, we begin by using the keyword `here`. The word 'char' is just a variable we created. Notice how we did not define this variable before. Whenever we use 'for' loops, we create what are called as loop variables. These exist only within the loop itself, to carry out the loop and its operations. Here, I used 'char' to represent 'characters' since Python does not identify letters as letters.

What this means is "for every character in the word 'Loops'", print out the characters. Surely enough, if you execute this code, you will end up with this:

L
o
o
p
s

The system iterates over each of the components and then uses those according to what the program says. Here, we only asked it to print characters. It started with 'L' and then moved on to 'o' and continued until there were no characters left.

It isn't necessary that you use a string, you can use what are termed as lists. These are a collection of values, either strings or numbers, stored within a list. The lists are represented by a square bracket '[' and can hold as many items as you like.

Let's try that and see what happens:

```
for char in ["I", "Love", "Programming"]:  
    print(char)
```

Output:

I
Love
Programming

See how that differed? That is because when we used a single string, every character was a different object. Here, the list holds multiple objects, instead

of printing them separately, it printed out whatever the value was within each component of the list.

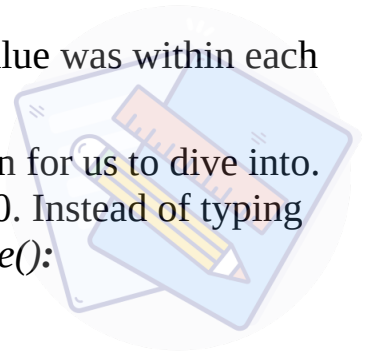
Here's one more example, and this one has a new function for us to dive into. Suppose you wish to print out the numbers from one to 20. Instead of typing the entire numbers, we use a built-in function called *range()*:

```
for number in range(20):
```

```
    print(number)
```

Here, we pass the higher end of the range as a parameter. Now, Python will execute this for us and the results will be exactly how you might imagine:

```
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
```



See how it printed the numbers to 19 and not 20? That is because for Python, the first position is always considered as zero. If you scroll up, you will see that the count started from zero. For now, do not get bogged down there. We will discuss that when we discuss about index numbers.

If you wish to set a specific starting point, you can do so by adding a value, followed by a comma, just before the 20:

```
for number in range(10, 20):
```

Now the count will begin from 10 and end at 19. Let's take that up a notch. Suppose I want to print out numbers from 10 to 20, and I want 20 to be printed, but I do not want all the numbers. I want the program to print out every second number, like 12, 14, 16 and so on. You can actually do that as this range function comes with what is termed as a 'step' for this function.

```
for number in range(10, 21, 2):
```

```
    print(number)
```

Output:

10

12

14

16

18

20

Now, the program executes, starts with the first number and knows that it needs to jump two steps and print that number. This will carry on until the final number, or the last possible number of iteration, is printed. Notice, in order to print 20, I had to change the value to 21 within the range.

E-commerce and e-shops use these quite a lot to iterate over the cart items and deliver you a total price of your potential purchase. In case you wish to see how that happens, here's one more example for a 'for' loop.

Scenario: I have five items in my imaginary cart. They are \$5, \$10, \$15, \$20, and \$25 in prices, respectively. I want the program to let me know what my total is. While I can use the calculator myself, or pause for a few seconds and calculate the price myself, I want a quicker solution. You, as a programmer,

will need to create something like this:

```
prices = [5, 10, 15, 20, 25]
```

```
total = 0
```

```
for item in prices:
```

```
    total += item
```

```
print(f"Your total price is: ${total}")
```

Output:

Your total price is: \$75

Let's be honest. This was much more fun to do than using a simple calculator, wasn't it? Programming can be tough at times, frustrating too. Sometimes, you might arrive at a point where you would spend the rest of your day, wondering what could possibly be causing you to have such a nightmarish time with a program that seemed too simple to execute.

Relax! Every one of us faces that. It comes with the kind of work we do. Programming can be quite deceptive and will take quite a lot of time for you to master. What's important is that you never give up. Should you feel frustrated, grab a drink, have some fresh air, and calm your mind. The solution would be more obvious than you might think.

Now that we have calmed down a little. Let's get back to learning Python. It is time to put an end to the loops by learning one more type of loop called 'nested' loop. If you recall, we have already seen a nested conditional statement; an 'if' statement within an 'if' statement. Similarly, we use a 'for' loop within a 'for' loop to get things that we wish to acquire.

The 'Nested' Loop

Let us start this one off by trying to type in some values for 'a,' 'b,' and 'c' We wish to have values from zero to two for each one in somewhat a similar fashion like we type coordinates:

(a, b, c)

(0, 0, 0)

(0, 0, 1)

And so it goes on until 'c' is two, after which the counter is (1, 0, 0) and starts again. That would be quite a lot of work if we were to write these on



our own. Fortunately, we have Python to help us out by using nested loops. How? Let's take a look.

```
for a in range(3):  
    for b in range(3):  
        for c in range(3):  
            print(f'({a}, {b}, {c})')
```



Wow! Look at that! A 'for' loop within a 'for' loop within another 'for' loop. That is a lot of loops right there. But, that is exactly how this will work. What happens now is that the program initiates with the first position of our loop variable 'c' while the remaining variables hold a value of zero. Then, the loop starts again; this time, only 'c' jumps to a value of one while others remain the same. This will continue right until 'c' reaches the end of the range, after which 'b' will gain value of one. Hopefully, you see how this is going. The result is as under:

```
(0, 0, 0)  
(0, 0, 1)  
(0, 0, 2)  
(0, 1, 0)  
(0, 1, 1)  
(0, 1, 2)  
(0, 2, 0)  
(0, 2, 1)  
(0, 2, 2)  
(1, 0, 0)  
(1, 0, 1)  
(1, 0, 2)  
(1, 1, 0)  
(1, 1, 1)  
(1, 1, 2)  
(1, 2, 0)
```

(1, 2, 1)

(1, 2, 2)

(2, 0, 0)

(2, 0, 1)

(2, 0, 2)

(2, 1, 0)

(2, 1, 1)

(2, 1, 2)

(2, 2, 0)

(2, 2, 1)

(2, 2, 2)



Phew! That would have taken us quite some time to write. However, some clever trickery of nested loops and just a few keystrokes later, we have it right now we want it. That is how effective nested loops are. When you are to deal with big chunks of data, you will want to rely quite a bit on nested loops. These get the job done and are mighty effective too.

Now, since that is out of the way, let us focus on operators. Those pesky little signs that keep on changing every now and then remember? We will be looking into these to see how they work for us.

Chapter 9 File handling

The Python programming language allows us to work on two different levels when we refer to file systems and directories. One of them is through the module `os`, which facilitates us to work with the whole system of files and directories, at the level of the operating system itself.

The second level is the one that allows us to work with files, this is done by manipulating their reading and writing at the application level, and treating each file as an object.

In python as well as in any other language, the files are manipulated in three steps, first they are opened, then they are operated on or edited and finally they are closed.

What is a file?

A python file is a set of bytes, which are composed of a structure, and within this we find in the header, where all the data of the file is handled such as, for example, the name, size and type of file we are working with; the data is part of the body of the file, where the written content is handled by the editor and finally the end of the file, where we notify the code through this sentence that we reach the end of the file. In this way, we can describe the structure of a file.

The structure of the files is composed in the following way:

- File header: These are the data that the file will contain (name, size, type)
- File Data: This will be the body of the file and will have some content written by the programmer.
- End of file: This sentence is the one that will indicate that the file has reached its end.

Our file will look like this:

Header of file
(name, size, type)

Body of file (data)

End of file



How can I access a file?

There are two very basic ways to access a file, one is to use it as a text file, where you proceed line by line, the other is to treat it as a binary file, where you proceed byte by byte.

Now, to assign a variable a file type value, we will need to use the function `open ()`, which will allow us to open a file.

Open() function

To open a file in Python, we have to use the `open()` function, since this will receive the name of the file and the way in which the file will be opened as parameters. If the file opening mode is not entered, it will open in the default way in a read-only file.

We must keep in mind that the operations to open the files are limited because it is not possible to read a file that was opened only for writing, you cannot write to a file which has been opened only for reading.

The `open ()` function consists of two parameters:

- It is the path to the file we want to open.
- It is the mode in which we can open it.

Its syntax is as follows:


```
)
1  function = open("file.txt", "w")
2  function.write()
3  function.close()
```



Of which the parameters:

File: This is an argument that provides the name of the file we want to access with the open() function, this is what will be the path of our file.

The argument file is considered a fundamental argument, since it is the main one (allowing us to open the file), unlike the rest of the arguments which can be optional and have values that are already predetermined.

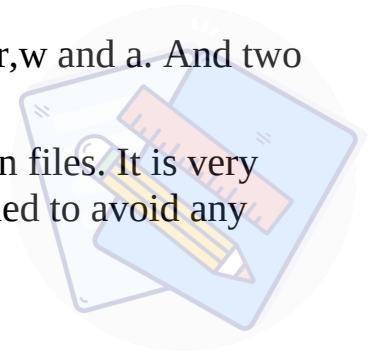
Mode: The access modes are those that are in charge of defining the way in which the file is going to be opened (it could be for reading, writing, editing).

There are a variety of access modes, these are:

r	This is the default open mode. Opens the file for reading only
r+	This mode opens the file for its reading and writing
rb	This mode opens the file for reading only in a binary format
w	This mode opens the file for writing only. In case the file does not exist, this mode creates it
w+	This is similar to the w mode, but this allows the file to be read
wb	This mode is similar to the w mode, but this opens the file in a binary format
wb+	This mode is similar to the wb mode, but this allows the file to be read
a	This mode opens a file to be added. The file starts writing from the end
ab	This is similar to mode a, but opens the file in a binary format
a+	This mode is pretty much like the mode a, but allows us to read the file.

In summary, we have three letters, or three main modes: r,w and a. And two submodes, + and b.

In Python, there are two types of files: Text files and plain files. It is very important to specify in which format the file will be opened to avoid any error in our code.



Read a file:

There are three ways to read a file:

1. `read([n])`
2. `readlines()`
3. `readline([n])`

Surely at this point, we have the question of what is meant by the letter n enclosed in parentheses and square brackets? It's very simple, the letter n is going to notify the bytes that the file is going to read and interpret.

Read method ([])

```
1 myfile = open("D:\\pythonfile\\mypythonfile.txt", "r")
2 myfile.read(9)
```

There we could see that inside the `read()` there is a number 9, which will tell Python that he has to read only the first nine letters of the file

Readline(n) Method

The `readline` method is the one that reads a line from the file, so that the read bytes can be returned in the form of a string. The `readline` method is not able to read more than one line of code, even if the byte n exceeds the line quantity.

Its syntax is very similar to the syntax of the `read()` method.

```
1 myfile = open("D:\\pythonfile\\mypythonfile.txt", "r")
2 myfile.readline()
```

Readlines(n) Method

The `readlines` method is the one that reads all the lines of the file, so that the read bytes can be taken up again in the form of a string. Unlike the `readline` method, this one is able to read all the lines.

Like the `read()` method and `readline()` its syntax are very similar:

```
1 myfile = open("D:\\pythonfile\\mypythonfile.txt","r")
2 myfile.readlines()
```

Once we have opened a file, there are many types of information (attributes) we could get to know more about our files. These attributes are:

`File.name`: This is an attribute that will return the name of the file.

`File.mode`: This is an attribute that will return the accesses with which we have opened a file.

`file.closed`: This is an attribute that will return a "True" if the file we were working with is closed and if the file we were working with is still open, it will return a "False".

Close() function

The close function is the method by which any type of information that has been written in the memory of our program is eliminated, in order to proceed to close the file. But that is not the only way to close a file; we can also do it when we reassign an object from one file to another file.

The syntax of the close function is as follows:

```
1 myfile.close()
2
```

What's a buffer?

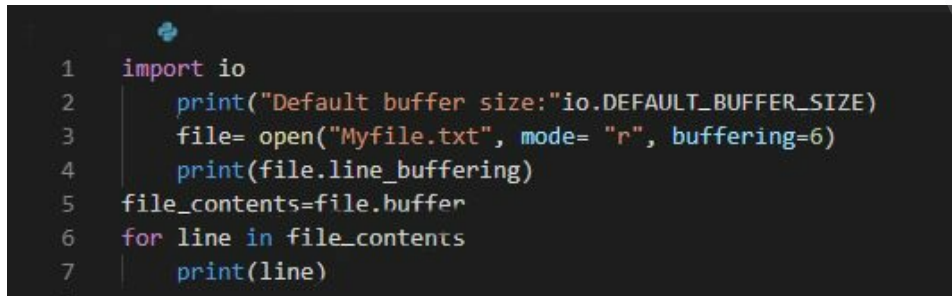
We can define the buffer as a file which is given a temporary use in the ram memory; this will contain a fragment of data that composes the sequence of files in our operating system. We use buffers very often when we work with a file which we do not know the storage size.

It is important to keep in mind that, if the size of the file were to exceed the ram memory that our equipment has, its processing unit will not be able to execute the program and work correctly.

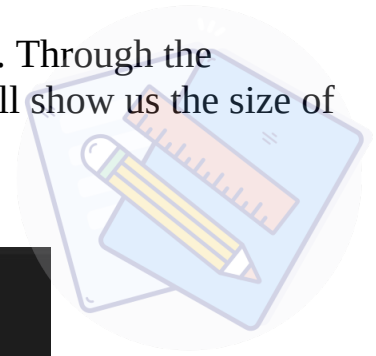
What is the size of a buffer for? The size of a buffer is the one that will

indicate the available storage space while we use the file. Through the function: `io.DEFAULT_BUFFER_SIZE` the program will show us the size of our file in the platform in a predetermined way.

We can observe this in a clearer way:



```
1 import io
2 print("Default buffer size:"io.DEFAULT_BUFFER_SIZE)
3 file= open("Myfile.txt", mode= "r", buffering=6)
4 print(file.line_buffering)
5 file_contents=file.buffer
6 for line in file_contents:
7     print(line)
```



Errors

In our files, we are going to find a string (of the optional type) which is going to specify the way in which we could handle the coding errors in our program.

Errors can only be used in txt mode files.

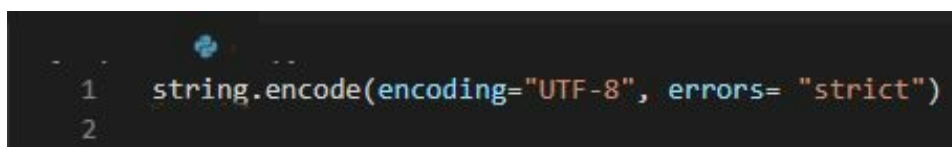
These are the following:

<code>Ignore_errors()</code>	This will avoid the comments with a wrong or unknown format
<code>Strict_errors()</code>	This is going to generate a subclass or <code>UnicodeError</code> in case that any mistake or fail comes out in our code file

Encoding

The string encoding is frequently used when we work with data storage and this is nothing more than the representation of the encoding of characters, whose system is based on bits and bytes as a representation of the same character.

This is expressed as follows:



```
1 string.encode(encoding="UTF-8", errors= "strict")
2
```

Newline

The Newline mode is the one that is going to control the functionalities of the new lines, which can be '\r', " ", none, '\n', and '\r\n'.

The newlines are universal and can be seen as a way of interpreting the text sequences of our code.

1.The end-of-line sentence in Windows: "\r\n".

2.The end-of-line sentence in Max Os: "\r".

3.The end-of-line sentence in UNIX: "\n"

On input: If the newline is of the None type, the universal newline mode is automatically activated.

Input lines can end in "\r", "\n" or "\r\n" and are automatically translated to "\n" before being returned by our program. If their respective legal parameters when coding are met, the entry of the lines will end only by the same given string and their final line will not be translated at the time of return.

On output: If the newline is of the None type, any type of character "\n" that has been written, will be translated to a line separator which we call "os.linesep".

If the newline is of the type " " no type of translator is going to be made, and in case the newline meets any value of which are considered the legal for the code, they will be automatically translated to the string.

Example of newline reading for " ".

```
1 string.encode(mode="r", newline= " ")
2
```

Example of newline reading for none:

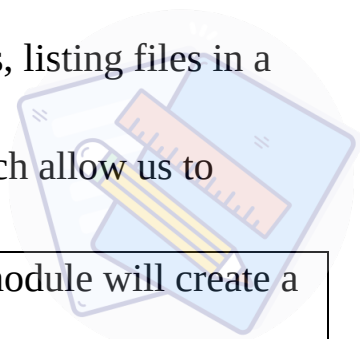
```
1 string.encode(mode="w", newline= "none")
2
```

Manage files through the "os" module

The "os" module allows us to perform certain operations, these will depend

on an operating system (actions such as starting a process, listing files in a folder, end process and others).

There are a variety of methods with the "os" module which allow us to manage files, these are:



os.makedirs()	This method of the “os” module will create a new file
os.path.getsize()	This method of the “os” module will show the size of a file in bytes.
os.remove(file_name)	This method of the “os” module will delete a file or the program
os.getcwd ()	This method of the “os” module will show us the actual directory from where we will be working
os.listdir()	This method of the “os” module will list all the content of any folder of our file
os.rename (current_new)	This method of the “os” module will rename a file
os.path.isdir()	This method of the “os” module will transfer the parameters of the program to a folder
os.chdir()	This method of the “os” module will change or update the direction of any folder or directory
os.path.isfile()	This method of the “os” module will transform a parameter into a file.

Xlsx files: xlsx files are those files in which you work with spreadsheets, how is this? Well, this is nothing more than working with programs like Excel. For example, if we have the windows operating system on our computer, we have the advantage that when working with this type of files, the weight of it will be much lighter than other types of files.

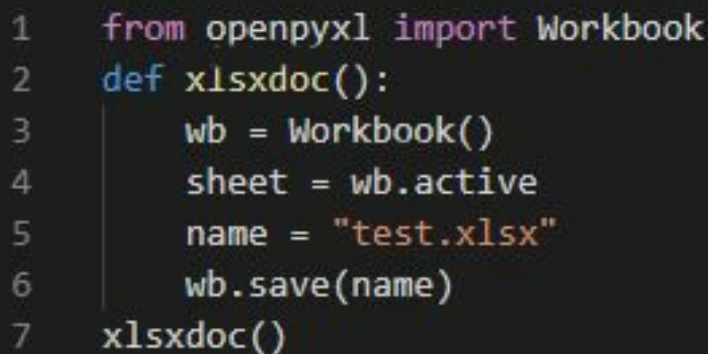
The xlsx type files are very useful when working with databases, statistics, calculations, numerical type data, graphics and even certain types of basic automation.

In this chapter we are going to learn to work the basic functionalities of this type of files, this includes creating files, opening files and modifying files.

To start this, first we will have to install the necessary library; we do this by executing the command "pip3 install openpyxl" in our Python terminal.

Once executed this command it is going to download and install the openpyxl module in our Python files, we can also look for documentation to get the necessary information about this module.


Create an xlsx file: To create a file with this module, let's use the openpyxl() Workbook() function.



```
1  from openpyxl import Workbook
2  def xlsxdoc():
3      wb = Workbook()
4      sheet = wb.active
5      name = "test.xlsx"
6      wb.save(name)
7  xlsxdoc()
```

This is the first step that we will do to manage the files of the type xlsx, we can see that first we have created the file importing the function Workbook of the module openpyxl; followed by this to the variable wb we have assigned the function Workbook() with this we declare that this will be the document with which we are going to work (we create the object in the form of a worksheet in this format). Once this is done, we activate the object whose name is wb in order to assign it a name and finally save the file.

Add information to the file with this module: In order to add information to our file, we will need to use another type of functions that come included with the object, one of them is the append function.



```

1  from openpyxl import Workbook
2  def xlsxdoc():
3      wb = Workbook()
4      sheet = wb.active
5      sheet["B4"] = "Goodnight"
6      name = "test.xlsx"
7      wb.save(name)
8  xlsxdoc()

```

We can observe that this is similar to the last example that we needed to create a document, for it we did the usual steps: we created in the function `xlsxdoc()` the object `wb`, we activated the object and there we added the information. In this new space we will need to know the specific position in which we are going to write, in this case, we will write in the fourth box of the second row "B4" and these will be matched with a string that says "goodnight". The final steps are exactly the same as the last example, therefore, we will place the name and save it with the save command.

There is a simpler way to write and enter data, we can do this through the function `append()`

```

1  from openpyxl import Workbook
2  def xlsxdoc():
3      wb = Workbook()
4      sheet = wb.active
5      messages = ("Hello" , "good morning", "goodnight" )
6      sheet.append(messages)
7      name = "test.xlsx"
8      wb.save(name)
9  xlsxdoc()

```

We can observe that we have created the document "test.xlsx" with the steps that we explained previously, we can observe that we have created a tuple called `messages`, this tuple has three items that are:


"Hello", "goodmorning", "goodnight".

Once the tuple is created, we use the `append` function, which will allow us to attach all the information contained in the tuple `messages` and finally save the document with the `save` function.

The `append()` function only admits iterable data, what does this mean? This

refers to the data of type arrangements, tuples since, if they are not entered in this way, our program will return an error.

Read documents in xlsx



```
1  from openpyxl import Workbook
2  name = "test.xlsx"
3  def xlsdoc():
4      wb = load_workbook(name)
5      sheet = wb.active
6      file1 = sheet["C1"].value
7      file2 = sheet["C2"].value
8      file3 = sheet["C3"].value
9      print(file1)
10     print(file2)
11     print(file3)
12     xlsdoc()
```

Let's go back to our first example to get information from xlsx files, we could see that, for this, we imported the load_workbook class. The first thing we need to know is the name of the file we want to open and for this, we created the variable with the name.

It is important that the files are located in the same folder in which the program is stored, because otherwise the program will throw us an error. Inside the function xlsdoc() we will create the object wb that will be with which we are going to work, followed by this the object "sheet" is created which is going to represent the sheet that we are going to use.

Once all this is done, we are going to request the information of the specific boxes "C1", "C2", "C3" next to the function value, to validate that the information that we acquire is real, we print all the information requested.

Handling PDF files

It is known that the initials of this type of file are: "Portable Document Format", which have grown significantly over the years, are mostly used in

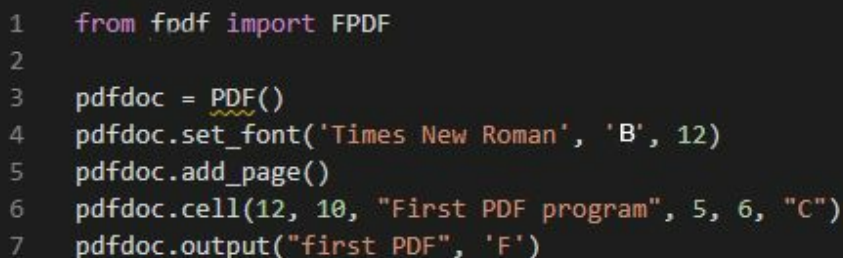
business and education. This is due to the fact that they provide a great amount of benefits in which its security is highlighted, allowing to add access keys to control who can edit the document and even add a watermark to it to avoid plagiarism of information.

Other outstanding data is that these documents can be seen from any device since it is not necessary to have a specific program; in addition, the weight of the files is much lower since these texts are compressed, unlike Word documents.

A disadvantage of PDF files could be that they are not easy to edit once they have been created.

In this chapter, we will only learn how to create PDF files.

To create a PDF file the first thing we will have to do is to download the library through the command "Pip3 install fpdf", followed by this we can proceed to create our document:



```
1 from fpdf import FPDF
2
3 pdfdoc = PDF()
4 pdfdoc.set_font('Times New Roman', 'B', 12)
5 pdfdoc.add_page()
6 pdfdoc.cell(12, 10, "First PDF program", 5, 6, "C")
7 pdfdoc.output("first PDF", 'F')
```

This is a simple level example, but at the same time, it is much more difficult than other types of files. To start a document you need a lot of commands, for it we will import the FPDF class from the fpdf library, followed by this we create the pdfdoc object and this will be the pdf document. Once created this document, we will have to customize the formats, size, and style of the letters we are going to use. To do this we use the command set_font.

In this case, the type of Font that we are going to use is going to be Times New Roman, with bold style and a size of 12.

Followed by this we will add a page through the command add_page(), since we will need a page on which to write and the function fpdf does not create a blank page by default. Then, we're going to insert information with the cell() function which contains a set of very important arguments.

The cell function will contain the width and height that the cell will occupy, it

must include the message that will be written in string format, in case it is required that the edges to come with some detail included we must add 1 since the 0 is by default and does not allow anything to be inserted.

If you want to add a cell below or located to the right, you place a 0 and otherwise is placed 1, if you want the text to be centered to the right, left, up or down a string will be placed and if you want in it to be centered you write C

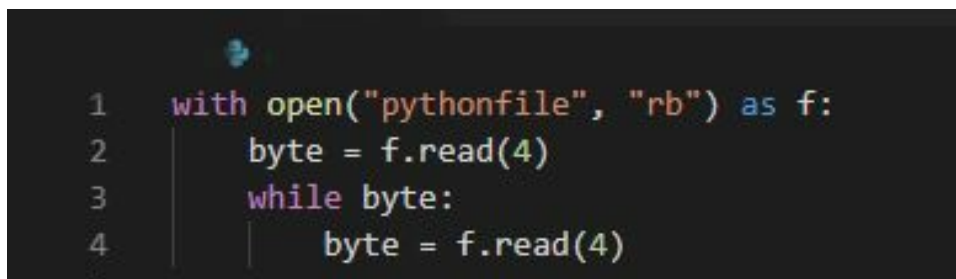
Finally, we will have to save the document through the command `output()`, and the arguments that will go with them will be the name of the file (with the ".pdf" included since we want a file in pdf) and then a string "F".

Managing BIN files

As we saw earlier, not all files are necessarily text files. These same ones can be processed by lines and even there exist certain files that when being processed, each byte contains a particular meaning for the program; for that reason, they need to be manipulated in their specific format.

A clear example of this are the files in Binary, to work with this type of files is no more than adding a b in the space of the parameter mode.

For example:



```
1  with open("pythonfile", "rb") as f:
2      byte = f.read(4)
3      while byte:
4          byte = f.read(4)
```

When we handle a binary file, it is very important to know the current position of the data we need in order to modify it. If you don't know the current position, the `file.tell()` function will indicate the number of bytes that have elapsed since we started the file.

In case you want to modify the current position in the file, we use the function `file.seek(star, from)` which will allow us to move a certain amount of bytes from start to finish.

Chapter 10 Exception Handling

What Is Exception Handling

Exception handling is error management. It has three purposes.

1. It allows you to debug your program.
2. It allows your program to continue running despite encountering an error or exception.
3. It allows you to create your customized errors that can help you debug, remove and control some of Python's nuances, and make your program function as you want it to.



Handling the Zero Division Error Exception

Exception handling can be an easy or difficult task depending on how you want your program to flow and your creativity. You might have scratched your head because of the word creativity. Programming is all about logic, right? No.

The core purpose of programming is to solve problems. A solution to a problem does not only require logic. It also requires creativity. Have you ever heard of the phrase, "Think outside of the box?"

Program breaking exceptions can be a pain and they are often called bugs. The solution to such problems is often elusive. And you need to find a workaround or risk rewriting your program from scratch.

For example, you have a calculator program with this snippet of code when you divide:

```
>>> def div(dividend, divisor):  
    print(dividend / divisor)  
>>> div(5, 0)
```

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

File "<stdin>", line 2, in div

ZeroDivisionError: division by zero

```
>>> _
```

Of course, division by zero is an impossible operation. Because of that, Python stops the program since it does not know what you want to do when this is encountered. It does not know any valid answer or response.

That being said, the problem here is that the error stops your program entirely. To manage this exception, you have two options. First, you can make sure to prevent such operation from happening in your program. Second, you can let the operation and errors happen, but tell Python to continue your program.

Here is what the first solution looks like:

```
>>> def div(dividend, divisor):
    if (divisor != 0):
        print(dividend / divisor)
    else:
        print("Cannot Divide by Zero.")
```

```
>>> div(5, 0)
```

Cannot Divide by Zero.

```
>>> _
```

Here is what the second solution looks like:

```
>>> def div(dividend, divisor):
    try:
        print(dividend / divisor)
    except:
        print("Cannot Divide by Zero.")
```

```
>>> div(5, 0)
```

Cannot Divide by Zero.

```
>>> _
```

Remember the two core solutions to errors and exceptions. One, prevent the error from happening. Two, manage the aftermath of the error.

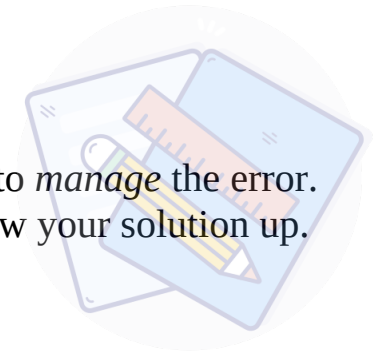
Congratulations!

The eight character of the password required to unlock the answer booklet is

letter t.

Using Try-Except Blocks

In the previous example, the try except blocks was used to *manage* the error. However, you or your user can still do something to screw your solution up. For example:



```
>>> def div(dividend, divisor):  
    try:  
        print(dividend / divisor)  
    except:  
        print("Cannot Divide by Zero.")
```

```
>>> div(5, "a")
```

Cannot Divide by Zero.

```
>>> _
```

The statement prepared for the “except” block is not enough to justify the error that was created by the input. Dividing a number by a string does not actually warrant a “Cannot Divide by Zero.” message.

For this to work, you need to know more about how to use except block properly. First of all, you can specify the error that it will capture and respond to by indicating the exact exception. For example:

```
>>> def div(dividend, divisor):  
    try:  
        print(dividend / divisor)  
    except ZeroDivisionError:  
        print("Cannot Divide by Zero.")
```

```
>>> div(5, 0)
```

Cannot Divide by Zero.

```
>>> div(5, "a")
```

Traceback (most recent call last):

File "<stdin>", line 1, <module>

File "<stdin>", line 3, in div

TypeError: unsupported operand type(s) for /: 'int' and 'str'

>>> _

Now, the error that will be handled has been specified. When the program encounters the specified error, it will execute the statements written on the “except” block that captured it. If no except block is set to capture other errors, Python will then step in, stop the program, and give you an exception.

But why did that happen? When the example did not specify the error, it handled everything. That is correct. When the “except” block does not have any specified error to look out for, it will capture any error instead. For example:

```
>>> def div(dividend, divisor):
```

```
    try:
```

```
        print(dividend / divisor)
```

```
    except:
```

```
        print("An error happened.")
```

```
>>> div(5, 0)
```

An error happened.

```
>>> div(5, "a")
```

An error happened.

```
>>> _
```

That is a better way of using the “except” block if you do not know exactly the error that you might encounter.

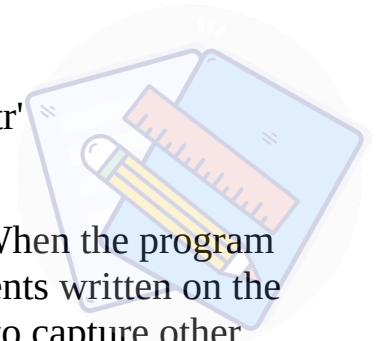
Reading an Exception Error Trace Back

The most important part in error handling is to know how to read the trace back message. It is fairly easy to do. The trace back message is structured like this:

<Traceback Stack Header>

<File Name>, <Line Number>, <Function/Module>

<Exception>: <Exception Description>



Here are things you need to remember:

- The trace back stack header informs you that an error occurred.
 - The file name tells you the name of the file where the fault is located. Since the examples in the book are coded using the interpreter, it always indicated that the file name is "<stdin>" or standard input.
 - The line number tells the exact line number in the file that caused the error. Since the examples are tested in the interpreter, it will always say line 1. However, if the error is found in a code block or module it will return the line number of the statement relative to the code block or module.
 - The function/module part tells what function or module owns the statement. If the code block does not have an identifier or the statement is declared outside code blocks, it will default to <module>.
 - The exception tells you what kind of error happened. Some of them are built-in classes (e.g., ZeroDivisionError, TypeError, and etcetera) while some are just errors (e.g., SyntaxError). You can use them on your except blocks.
 - The exception description gives you more details with regards to how the error occurred. The description format may vary from error to error.
- **Using exceptions to prevent crashes**

Anyway, to know the exceptions that you can use, all you need to do is to generate the error. For example, using the TypeError found in the previous example, you can capture that error too and provide the correct statements in response.

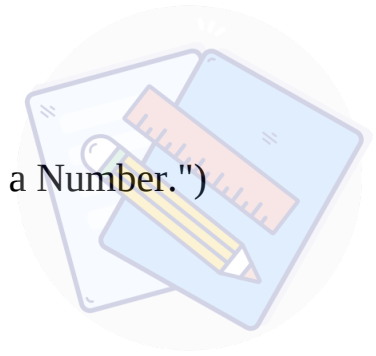
```
>>> def div(dividend, divisor):  
    try:  
        print(dividend / divisor)  
    except ZeroDivisionError:
```

```
        print("Cannot Divide by Zero.")
    except TypeError:
        print("Cannot Divide by Anything Other Than a Number.")
    except:
        print("An unknown error has been detected.")
>>> div(5, 0)
Cannot Divide by Zero.
>>> div(5, "a")
Cannot Divide by Anything Other Than a Number.
>>> div(undeclaredVariable / 20)
An unknown error has been detected.
>>> _
```

However, catching errors this way can still be problematic. It does allow you to prevent a crash or stop, but you have no idea about what exactly happened. To know the unknown error, you can use the **as** keyword to pass the Exception details to a variable. Convention wise, the variable detail is often used for this purpose.

For example:

```
>>> def div(dividend, divisor):
    try:
        print(dividend / divisor)
    except Exception as detail:
        print("An error has been detected.")
        print(detail)
        print("Continuing with the program.")
>>> div(5, 0)
An error has been detected.
division by zero
Continuing with the program.
```



```
>>> div(5, "a")
```

An error has been detected.

unsupported operand type(s) for /: 'int' and 'str'

Continuing with the program.

```
>>> _
```



The Else Block

There are times that an error happens in the middle of your code block. You can catch that error with try and except. However, you might not want to execute any statement in that code block if an error happens. For example:

```
>>> def div(dividend, divisor):
```

```
    try:
```

```
        quotient = dividend / divisor
```

```
    except Exception as detail:
```

```
        print("An error has been detected.")
```

```
        print(detail)
```

```
        print("Continuing with the program.")
```

```
        print(str(dividend) + " divided by " + str(divisor) + " is:")
```

```
        print(quotient)
```

```
>>> div(4, 2)
```

4 divided by 2 is:

2.0

```
>>> div(5, 0)
```

An error has been detected.

division by zero

Continuing with the program.

5 divided by 0 is:

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

File "<stdin>", line 8, in div

Print(quotient)

UnboundLocalError: local variable 'quotient' referenced before assignment

>>> _

As you can see, the next statements after the initial fault are dependent on it thus they are also affected. In this example, the variable quotient returned an error when used after the try and except block since its supposed value was not assigned because the expression assigned to it was impossible to evaluate.

In this case, you would want to drop the remaining statements that are dependent on the contents of the try clause. To do that, you must use the else block. For example:

```
>>> def div(dividend, divisor):
```

```
    try:
```

```
        quotient = dividend / divisor
```

```
    except Exception as detail:
```

```
        print("An error has been detected.")
```

```
        print(detail)
```

```
        print("Continuing with the program.")
```

```
    else:
```

```
        print(str(dividend) + " divided by " + str(divisor) + " is:")
```

```
        print(quotient)
```

```
>>> div(4, 2)
```

```
4 divided by 2 is:
```

```
2
```

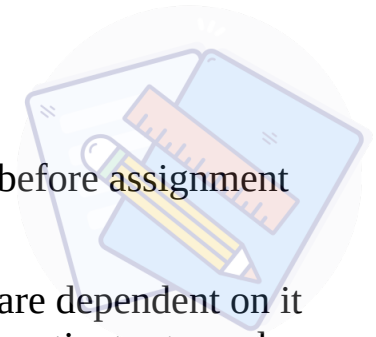
```
>>> div(5, 0)
```

```
An error has been detected.
```

```
division by zero
```

```
Continuing with the program.
```

```
>>> _
```



The first attempt on using the function with proper arguments went well. On the second attempt, the program did not execute the last two statements under the else block because it returned an error.

The else block always follows except blocks. The function of the else block is to let Python execute the statements under it when the try block did not return and let Python ignore them if an exception happens.

Failing Silently

Silent fails or failing silently is a programming term often used during error and exception handling.

In a user's perspective, silent failure is a state wherein a program fails at a certain point but never informs a user.

In a programmer's perspective, silent failure is a state wherein the parser, runtime development environment, or compiler fails to produce an error or exception and proceed with the program. This often leads to unintended results.

A programmer can also induce silent failures when he either ignores exceptions or bypasses them. Alternatively, he blatantly hides them and creates workarounds to make the program operate as expected even if an error happened. He might do that because of multiple reasons such as the error is not program breaking or the user does not need to know about the error.

Handling the File Not Found Exception Error

There will be times when you will encounter the FileNotFoundError. Handling such error depends on your intent or purpose with regards to opening the file. Here are common reasons you will encounter this error:

- You did not pass the directory and filename as a string.
- You misspelled the directory and filename.
- You did not specify the directory.
- You did not include the correct file extension.
- The file does not exist.

The first method to handle the FileNotFoundError exception is to make sure

that all the common reasons do not cause it. Once you do, then you will need to choose the best way to handle the error, which is completely dependent on the reason you are opening a file in the first place.

Checking If File Exists

Again, there are always two ways to handle an exception: preventive and reactive. The preventive method is to check if the file exists in the first place.

To do that, you will need to use the `os` (`os.py`) module that comes with your Python installation. Then, you can use its `path` module's `isfile()` function. The `path` module's file name depends on the operating system (`posixpath` for UNIX, `ntpath` for Windows, `macpath` for old MacOS). For example:

```
>>> from os import path
>>> path.isfile("random.txt")
False
>>> path.isfile("sampleFile.txt")
True
>>> _
```

Try and Except

You can also do it the hard way by using `try`, `except`, and `else` blocks.

```
>>> def openFile(filename):
    try:
        x = open(filename, "r")
    except FileNotFoundError:
        print("The file " + filename + " does not exist.")
    except FileNotFoundError:
        print("The file " + filename + " does exist.")
>>> openFile("random.txt")
The file 'random.txt' does not exist.
>>> openFile("sampleFile.txt")
The file 'sampleFile.txt' does exist.
```

```
>>> _
```

Creating a New File

If the file does not exist, and your goal is to overwrite any existing file anyway, then it will be best for you to use the "w" or "w+" access mode. The access mode creates a new file for you if it does not exist. For example:

```
>>> x = open("new.txt", "w")
```

```
>>> x.tell()
```

```
0
```

```
>>> _
```

If you are going to read and write, use "w+" access mode instead.

Practice Exercise

Try to break your Python by discovering at least ten different exceptions.

After that, create a loop.

In the loop, create ten statements that will create each of the ten different exceptions that you find inside one try block.

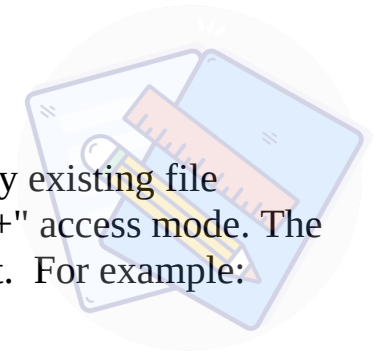
Each time the loop loops, the next statement after the one that triggered an exception should trigger another and so on.

Provide a specific except block for each one of the errors.

Summary

Exception handling skills are a must learn for beginners. It teaches you how to predict, prevent, and manage exceptions. It allows you to test and debug your program and generate compromises with your skill, program, and system's limits.

That being said, you have made great strides to get this far and I commend you!



Chapter 11 Tips and Tricks for Success

We've reached the end of this book. You've been introduced to Python's main topics and written some programs. What's next? How do you keep improving and mastering the Python language? Here are some tips for success that keep you on the right path:



Code everyday

Practice makes perfect. When learning a new language like Chinese or Spanish, experts recommend you use it every day in the form of speaking and going through an exercise or so. It's no different with a programming language. The more you practice, the more the basic syntax will become second nature to you, and you'll instinctively know when to use concepts like conditionals and loops. There are lots of resources (which you find in the appendix) that provide exercises and sample programs you can work on right away.

Write by hand

When you're taking notes (and you should take notes), write them out by hand. Studies show that the process of taking physical pen to physical paper facilitates the best long-term memory retention. Writing by hand includes writing code by hand, and then checking it on your computer, so you know for sure if it's accurate. Outlining code and ideas this way can help you stay organized, as well, before starting to actually build a program.

Find other beginners

Learning to code by yourself can get boring and frustrating. One of the best ways to learn and improve is to find others who are in the same phase as you. You can bounce ideas off each other, help out on projects, and more. If you don't know anyone in your immediate circle, you can check out groups online and/or find local events through Meetups and Facebook. Always exercise caution and employ safe practices when first meeting people you only know online. Stick to public places during daylight hours, and don't go anywhere alone with someone you don't know well until you feel comfortable.

Try explaining Python out loud

Sometimes explaining something you just learned to someone is the best way to really cement it into your mind. It allows you to reframe concepts into your

own words. You don't even have to talk to a real person; it can be an inanimate object. This is such a common practice among programmers that it's known as "rubber duck debugging," which references talking to a rubber duck about bugs in a program. Pick a topic in Python like conditionals or variables, and see if you can explain it. If you have trouble or realize there's a gap in your knowledge, it's time to go back and keep learning.

Check out other languages

This book is about Python, so obviously we believe that should be your priority, but getting to know a little bit about other languages can be very helpful, too. It will definitely make you a better programmer in the future. Checking out other languages can help you discover what common architecture is built into every language as well as the differences between them and Python. Even if you just read about other languages and never write much code in anything besides Python, you'll be improving your knowledge and skill.

Have a plan for when you get stuck

When you get stuck while coding, take a step back. What are you trying to get the code to do? What have you tried? And what's happening? Write the answers down and be as specific and detailed as possible. Then, you can go to someone else for help, and you won't have to spend a ton of time trying to explain the problem. The answers are also really useful just for your own thought process. Take a close look at any error messages you're getting. Work your way backward to try and spot any mistakes.

Another response to getting stuck is to just start over. If your code is really long, it can be discouraging to start from scratch, but that means you don't have to go through the whole thing, picking it apart and wearing out your eyes. Starting over may actually be easier.

Take a break

Whether you choose to begin again or go through the code with a fine-toothed comb, you should take breaks. When you work on a problem for too long, your brain gets stuck in a groove, and it's difficult to come up with new solutions. Go do something that doesn't use the exact same muscles as coding. Exercise your body instead, take a long shower, lie down for a nap, or bake some cookies. Einstein would often come up with solutions to his problems while he played the violin, and who doesn't want to think a little bit

like Einstein?



Conclusion

Learning how to get started with computer programming may seem like a large challenge. You can go with many distinct programming alternatives, but many of them are difficult to learn, will take some time to figure out, and will not always do all the things you need. Many people are afraid they need to be smart or have a lot of education and coding experience before they can make it to the level of coding they want. But, even a beginner may get into programming with Python.

Whether you're a beginner or have been in this field for some time, Python has made it so easy to code. The language is English-based, so it's simple to read, and it's got rid of many of the other symbols that make coding difficult for others to read. And since it's a user domain, to make things easier, as anyone can make changes and see other codes.

Keep in mind that, if you have any questions that may not have been answered in this book, you can always visit the Python website! The Python website contains a lot of material that will help you work with Python and ensure that you are entering your code properly. You can also find any updates that you may need for your Python program in the event that your program is not updating properly or you need another version of it.

You can work with the program and teach it what you want it to do, and you may even be able to help someone else out if they are not able to get the program to do what they want it to do!

Just remember that you do not need to worry if your Python code doesn't work the first time because using Python takes a lot of practice. The more you practice, the better your code will look, and the better it will be executed. Not only that, but you will get to see Machine Learning in action each time you enter your code!