

PYTHON PROGRAMMING FOR BEGINNERS

2023

MASTERING IN 1 WEEK

MARK GOWEN

Python Programming for Beginners

The Simplified Beginner's Guide to Mastering Python Programming in One Week. Learn Python Quickly with No Prior Experience.

Mark Gowen

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.

Copyright @ 2023 by Mark Gowen

Table of Contents

INTRODUCTION

Chapter 1- STARTING WITH PYTHON

Setting up

Installing Python in Windows

Python on different operating systems

Installing Python on MacOS

Installing Python in Ubuntu Linux

Installing Python on Android

VARIABLES AND SIMPLE DATA TYPES

Variables

Strings

Numbers

Chapter 3- INTRODUCING LISTS

What is a list

How to create a list

Organizing a list

Changing, adding, and removing elements

Exercise

Chapter 4- WORKING WITH LISTS

Looping

Tuples

Making numerical lists

IF STATEMENT

Conditional tests

If statements

Using IF with lists

Exercise

DICTIONARIES

Working with dictionary

Exercise

INPUT AND WHILE LOOPS

How Input works

Exercise

Introduction While Loops

Exercise

FUNCTIONS

Definition

Exercise

Classes

How to create and use classes

Creation of a class

Exercise

Solution

FILES AND EXCEPTIONS

Reading

[Writing into a file](#)

[Appending a file](#)

[Exercise](#)

[Exception - how to use it](#)

[Try and except the statement](#)

[Raising exception](#)

[Security risks](#)

[Exercise](#)

[TESTING YOUR CODE](#)

[Unit tests](#)

[Integration tests](#)

[MAKE THE FIRST PROJECT](#)

[REFERENCES](#)

INTRODUCTION

In the IT and computer programming sector, there are many programming languages used by professionals every day. Because of the high demand put on the IT sector and the technology sector, professionals have seen a rise in many programming languages in the IT and computer programs sector. Programmers and developers are using many new programming languages and tools that make the process easier and give high-performance code as output.

However, there are some classical languages that are still in trend. One of the classic programming languages is the Python language. The Python language was created in the year 1991 by a programmer named Guido van Rossum. The language is used for many purposes in the IT and computer programming arena. Today, Python is a programming language that is used to create server-side programs, apps, and online applications. The apps created with the help of Python are used on different devices. Python is also used to create mathematical applications and to also do system scripting for many devices. The versatility of the Python language and its adaptability to different devices in the era of multiple smart devices is a reason why the language is very popular among programmers and developers.

Python is a high-utility language that can be used for multiple reasons. The language is a good one to use for app development purposes. Python can also be used in conjunction with other programs to design intricate workflows. Database system connections can also be made using the programming language. The language can also be used to read and modify files for another cycle of development. The programming language can also be used to create and handle big databases so that complex mathematics commands can be operated with the help of a system.

The language is also used to maintain the code for the prototyping of software. The programming language can also be used to create production-ready software that is very high quality and efficient in terms of its performance. Apart from its versatility, there are other reasons why

Python is still a popular language in the IT and computer programming sector. The Python language is known for its readability. It is very easy to use because it is very close to the English language, which is also influenced by the mathematics language and can be used to solve mathematical problems very easily.

The Python language is also well known for its adaptability to different systems and devices. It can be used, for instance, on a variety of operating systems, including Windows, Mac, Linux, Raspberry Pi, etc. The simple syntax of the language makes it easier for programmers to use it for the proper development of apps on many systems.

Developers also use Python because the language uses complete sentences and line breaks for its code. The language uses far fewer lines to code than other programming languages. Moreover, the Python language also runs on an interpreter system. The language, with the help of the interpreter system, becomes very easy to execute. The execution happens as soon as it is written. Because of the fast-paced interpreter system integrated with the Python language, the execution of code and prototyping of software is very easy and fast-paced.

The Python language is free and open source, so it is very easy to use and economical. The language is supported by a number of libraries and tools, so users can easily code with it. You can directly run the commands from the source code, which makes it a very convenient language to use. The process of interpretation is organized in such a manner the code is easily interpreted and given to the computer system in a native language format so that the code is easily executed properly. The source code can be changed into small code packets of bytecodes which are then interpreted into the native language.

The language is a high-level language that is easy to manage for developers. This is a reason that freshers in the development field use this language to code early in their careers. The language of Python can be used in line with C and C++ so that you can create code for scripting and creating embeddable code to make operating systems. The language and code written with it are easy to manage and store because they can be used with good memory management techniques.

In this book, we will learn about the different ways in which we can code with the help of Python. With the use of Python, we can create something that is very useful and efficient as software. Python continues to be popular and is evolving into new, more useful versions. The sign of a great computing language is when the language evolves further and is very relevant in the future. There are many competing languages alongside the Python language. However, this language is a very useful and popular language even today.

Learning Python programming can help you get the best kind of jobs in the industry because companies are looking for professionals who are proficient in this language. Those who know the Python language are given preference over other freshers in technical interviews. Python is a beginner-friendly language that can be built upon with the help of your skills and learning potential.

Chapter 1- STARTING WITH PYTHON

Before we go into the salient and advanced features and use of Python, we have to know about the way in which Python can be set up in a computer so that the professional can start to code. Installing the Python language is very easy with a computer, and some computers even have the language pre-installed. Before you start setting the language up, you need to check if you already have it in the system.

Let us first learn how to check for pre-installed Python in your Windows system. You can search for Python in the search bar. On the other hand, you can also type the command line (cmd.exe) or type Python—version for your Windows system or Mac system. If you don't already have Python installed on your computer, you can do so by visiting <https://www.python.org/>. and looking for a link to a download. When downloading, check for the releases that are made for the Windows system. The language should be downloaded in the most recent version, like the Python 3.9.4 release.

Setting up

Installing Python in Windows

You must additionally download and run the executable installer in order to configure the downloaded language. You must double-click the downloaded executable file to accomplish this. The install window will then appear. Choose the customized installation option, then carry out the steps. The Python path will then be automatically configured when you have selected the Add Path check box.

After this, once you've downloaded Python, start the installation. Make sure the install launcher for all users and add Python 3.9 to path checkboxes are selected before running it. After this, you need to choose the install now checkbox.

The installation process will take some time, and you have to wait for the system to complete it. If you want to disable the path length limit, you will be prompted in the following dialog box. When you choose this option, you are able to choose the allowance limit for Python so that the language can bypass the 260-character limit maximum length for its command line. Effectively, it will help you enable Python to use long path names.

None of the other system settings will be affected by the Disable path length limit option. If you're using the language on the Linux operating system, turning it on will assist you in resolving any name length difficulties.

Once the Python version is installed on the computer, you need to check its functionality and commands. To confirm that the installation process has been smooth, you need to check the language by opening the Cmd prompt in your system. After this, you need to run Python V on the computer system.

After you have installed Python on the computer with Windows, you need to go to Windows search. Then you need to type IDLE and then open it. You will be ready to run the first Python code on your Windows system.

Python on different operating systems

Apart from Windows, there are other operating systems on which Python can run. To run Python on these OS, you need to first install the language. The steps to install the language on different

operating system is different.

Installing Python on MacOS

For example, the steps to installing Python on MacOS are different from the steps on Windows. Here are the steps to install Python on MacOS.

Before you install Python on the MacOS, you need to check your system and see if it is already installed in it. If it is not installed, you can respond to the prompt that the MacOS will give. The prompt will ask you to install the tools needed to use Python on the system. Xcode is an app that you need to install in the MacOS to use Python. When you choose to install Xcode, the programs will also install Python 3 so that it can be used easily. Along with the Xcode tool and Python, this process will also install some additional programs.

If you do not get the prompt to install Xcode into the system, you can choose to install Xcode by choosing to type **\$ Xcode-select --install** in the Terminal.

To install Python, the best place to start is the official website. As you did with the Windows system, you can do the same for the MacOS. For this, you need to go to www.python.org and then click on the link to download the latest version of Python for the MacOS system.

After this, you need to go to the download folder and then double-click on the Python-<version>-macosx.pkg so that the Python installer runs. Go through each of the prompts and then keep the default install location. Close and then move the installer to Trash.

There are many package installers that you can use to install Python on MacOS.

Installing Python in Ubuntu Linux

You can also set up Python on Ubuntu Linux. There are many ways to install Python on Ubuntu Linux.

Before you install Python on Ubuntu Linux, you need to check if the system already has the programming language in it. If the language is already installed on the computer, you do not have to re-install it, but you can update it. Use the Alt + Ctrl + T keyboard shortcut to open up the Terminal. If, after giving the command, you get an output with a version number, then you have the language installed. If you wish to exit Python, then you need to press Ctrl +D. If you get an error message which says command not found then it means that you do not have Python inside. You can also run the **python3 --version** command to see if you have the latest version.

There is another command that helps you check if you have an earlier version inside. For example, you can run **sudo apt --only-upgrade install python3** to check for an older version of the language and to update it to the most recent version.

If you have not installed Python, then you can do so through the official repository of Ubuntu. Since Python is present in the official repository, then you don't have to do much. You only have to run a seamless command to start the installation process. To update all packages and repositories in Ubuntu, open Terminal and execute the command provided below.

sudo apt update && sudo apt upgrade -y

Afterward, install Python on your machine by running the command below, which will perform an automatic installation.

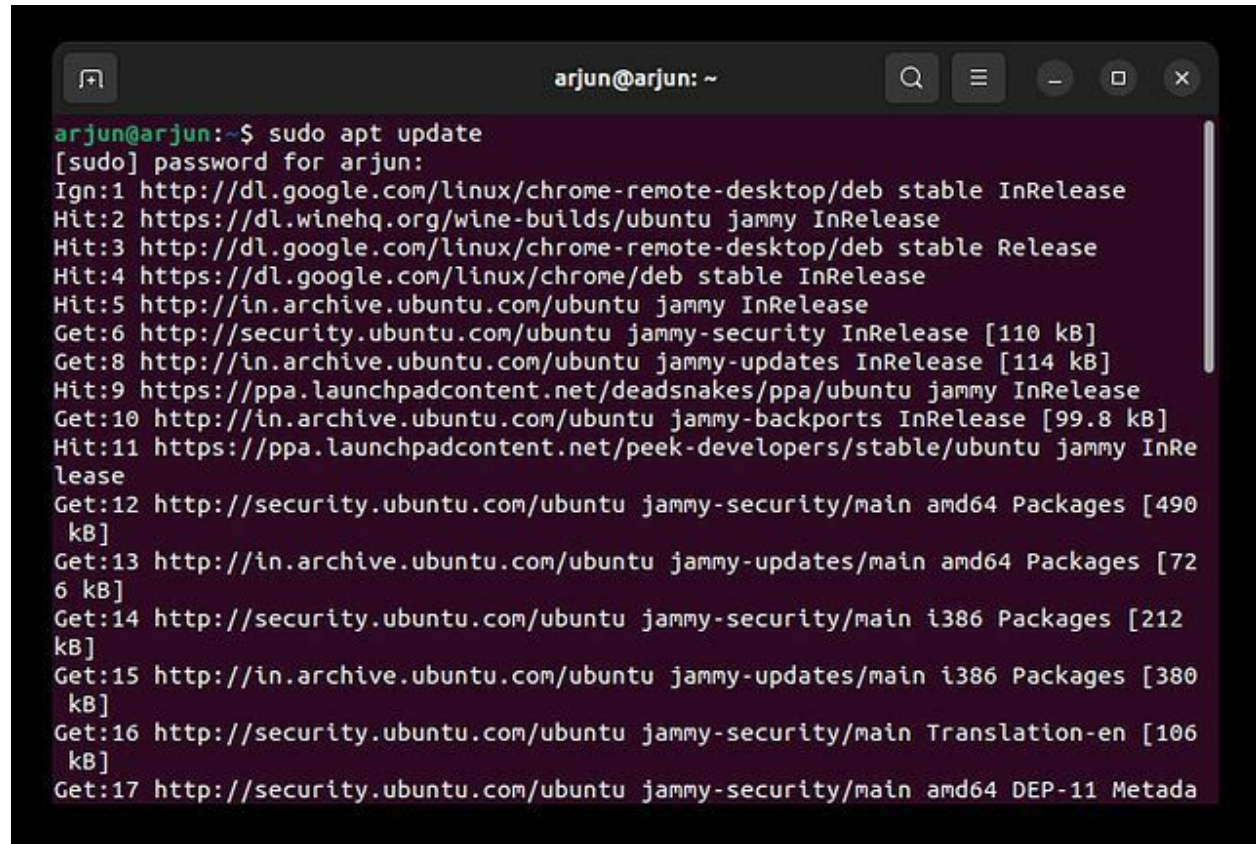
sudo apt install python3

These are the usual ways of installing the language in Ubuntu Linux. However, there is also

another way in which you can build the language from the source code. However, the process of building Ubuntu Linux is a lengthier one and takes more than a few commands. You will spend more time building the language from the source code. You have to also pay attention to your hardware specs if you plan to build the language.

First, you need to open the Terminal and run the command to update all packages.

sudo apt update



```
arjun@arjun:~$ sudo apt update
[sudo] password for arjun:
Ign:1 http://dl.google.com/linux/chrome-remote-desktop/deb stable InRelease
Hit:2 https://dl.winehq.org/wine-builds/ubuntu jammy InRelease
Hit:3 http://dl.google.com/linux/chrome-remote-desktop/deb stable Release
Hit:4 https://dl.google.com/linux/chrome/deb stable InRelease
Hit:5 http://in.archive.ubuntu.com/ubuntu jammy InRelease
Get:6 http://security.ubuntu.com/ubuntu jammy-security InRelease [110 kB]
Get:8 http://in.archive.ubuntu.com/ubuntu jammy-updates InRelease [114 kB]
Hit:9 https://ppa.launchpadcontent.net/deadsnakes/ppa/ubuntu jammy InRelease
Get:10 http://in.archive.ubuntu.com/ubuntu jammy-backports InRelease [99.8 kB]
Hit:11 https://ppa.launchpadcontent.net/peek-developers/stable/ubuntu jammy InRelease
Get:12 http://security.ubuntu.com/ubuntu jammy-security/main amd64 Packages [490 kB]
Get:13 http://in.archive.ubuntu.com/ubuntu jammy-updates/main amd64 Packages [726 kB]
Get:14 http://security.ubuntu.com/ubuntu jammy-security/main i386 Packages [212 kB]
Get:15 http://in.archive.ubuntu.com/ubuntu jammy-updates/main i386 Packages [380 kB]
Get:16 http://security.ubuntu.com/ubuntu jammy-security/main Translation-en [106 kB]
Get:17 http://security.ubuntu.com/ubuntu jammy-security/main amd64 DEP-11 Metada
```

The following command will install the necessary dependencies for building the Python language.

sudo apt install build-essential zlib1g-dev libncurses5-dev libgdbm-dev

The next step is to make a Python folder and move it there. You must use the command line sudo if you get a permission denied message.

sudo mkdir /python && cd /python

Then, you can use the wget command to download the Python language from the official website.

sudo wget <https://www.python.org/ftp/python/3.12.0/Python-3.12.0a1.tgz>

Then, you need to extract the downloaded file using the tar command.

sudo tar -xvf Python-3.12.0a1.tgz

cd Python-3.12.0a1

After this, you need to turn on optimizations by running the `./configure --enable-optimizations` and then go on to build the language. This reduces the time spent on compiling the language.

The last step will be to execute the **`sudo make install`** command. This will help compile the language in 10 to 15 minutes. Once the compilation process is done, you can run the **`python3 --version`** command to check the Python language and its run on the system.

Installing Python on Android

You can also download Python language and related tools on an Android system. To download Python on the Android system and code with the language, you need to download Pydroid 3 – IDE so that you can code with Python 3 on the app. You can download the IDE from the Google Play store.



Pydroid 3 - IDE for Python 3

IIEC

Contains ads • In-app purchases

4.4 ★

4K reviews



44 MB



Rated for 3+ ⓘ

50

De

Install



About this app



Learn Python 3 with the most powerful Python 3 interpreter & IDE on Google Play

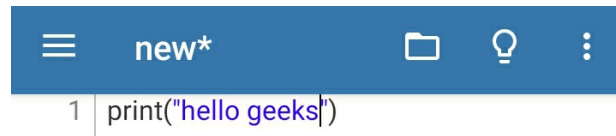
Education

Ratings and reviews



After installation, run this app on the system, and it will show installing Python.

Wait for a few minutes, and it will show you the IDE. Here you can enter the Python code. After this, you can click on the yellow button.

A screenshot of a code editor interface. The top bar is blue and contains a hamburger menu icon, the text 'new*', a folder icon, a lightbulb icon, and a vertical ellipsis icon. Below the bar, the first line of code is '1 | print("hello geeks|')'.

```
1 | print("hello geeks|')
```





TAB



ello geeks

Program finished]

VARIABLES AND SIMPLE DATA TYPES

There are different data types that Python works with. These data types can be variables, strings, and numbers. In the field of computer programming, the data types specify the Type of data that can be stored inside a variable. First, we have to know what a variable is in Python. Python is an object-oriented language. So, everything that we type in Python is an object. Data types are actually classes of this object, and variables are instances of these classes.

Variables

Variables and data types in Python are the values that vary. A variable in a programming language is a place in memory where you can store a value. The standards may require that the value you may have recorded alter in the future. When a value is assigned to a Python variable, it becomes a variable. In Python, a variable can be defined with just one command. When writing a variable, there are a number of guidelines that you must stick to.

Example of a variable - **x = 10**

#variable is declared as the value 10 is assigned to it.

A Python variable's nature is determined by a set of characteristics and rules.

1. The variable name cannot start with a number. It can only start with a character or an underscore.
2. Variables in Python are case sensitive.
3. They can only contain alpha-numeric characters and underscores.
4. No special characters are allowed in variables.

There are many data types in Python that programmers use to write commands and values. The data types are numbers, lists, sets, range, dictionary, strings and tuples.

Variables assignment

The assignment of Variables value is done with the help of an equal sign. For example, **>>> n = 300**

The value 300 is given to this variable, which is read as **n**. A value statement or expression can then be made using **n**. For example, **>>> print(n)**

You can also change the value of **n**.

```
>>> n = 1000
```

```
>>> print(n)
```

```
1000
```

```
>>> n
```

```
1000
```

The variable can also be stated as a chained assignment. This makes it possible for the system to assign the value to several variables again and again.

```
>>> a = b = c = 300
```

```
>>> print(a, b, c)
```

300 300 300

In Python, a variable is a name that represents or points to an object. After assigning an object to a variable, you can use that name to refer to the object. However, the object still contains the data itself.

Variable exercises

- Define variables. Write the characteristics of variables in Python.
- What are the different types of variables that we have in Python?
- Create a variable named **carname** and assign the value **Volvo** to it.

Answer

```
Carname= "Volvo"
```

- Create a variable named x and assign the value 50 to it.

Answer

```
X=50
```

- Show the sum of 2 and 10 using two variables, x, and y, in Python.

Answer

```
X=2
```

```
Y=10
```

```
Print (x+y)
```

- Create a Python application that requests user input for two numbers, a and b, and then displays their sum: a + b

```
X=5
```

```
Y=10
```

```
Z=x+y
```

```
Print (z)
```

- Write a program in Python that displays the first 100 integers: 1 2 3 ... 100

```
for i in range(1, 101):
```

```
    print(i, end=' ')
```

Python has the following data types in it.

Python has the following data types built-in by default in these categories:

Text Type: str

Numeric Types: int, float, complex

Sequence Types: list, tuple, range

Mapping Type: dict

Set Types: set, frozenset

Boolean Type: bool

Binary Types: bytes, bytearray, memoryview

None Type: NoneType

Strings

Strings are a different data type in Python. The idea of Strings is that they represent Unicode character values. A string can also consist of just one character. Within single or double quotations, we denote or declare the string value. We use indexes and square brackets to access the values within the string data type. Once replaced, you cannot change a string value.

Example

```
x = input()
print( 'hello' , x)
```

A Python string is a sequence of characters. There is a built in class str that includes the string values of Python. While you can use both single quotes and double quotes to declare the string value, it should be the same on both sides. You cannot use single quote on one side and double quote on the other side to declare the value.

For example, for the line `>>> a='Dogs are love'` you will get the output of **SyntaxError: EOL while scanning string literal**

You also cannot write the command like `>>> a="Dogs are "love"` as this will show invalid syntax.

The right syntax is

- `>>> a='Dogs are love'`
- `>>> print(a)`

Or

- `>>> a="Dogs are love"`
- `>>> print(a)`

If you have to span the code across multiple lines, then you need to use triple quotes.

- `[php]>>> a="""Hello`
- `Welcome"""`
- `>>> print(a)[/php]`

This gives the output of

- Hello
- Welcome

To access a Python string, you need to write the code like

- >>> a="Dogs are love"
- >>> a

This gives an output of 'Dogs are love'

To display a single character on the string, you need to put the index in square brackets.

```
>>> a[1]
```

The output comes as 'o'

String exercises

What is a string data type and what are its characteristics?

Create a program that takes the first, middle, and last characters from an input string and creates a new string. The input string is **str1 = "James"**

Answer

- Use string indexing to get the character present at the given index.
- Use str1[0] to get the first character of a string and add it to the result variable
- Next, get the middle character's index by dividing the string length by 2. $x = \text{len}(\text{str1}) / 2$. Use str1[x] to get the middle character and add it to the result variable
- Use str1[len(str1)-1] to get the last character of a string and add it to the result variable
- print result variable to display a new string

Given two strings, s1, and s2. Write a program to create a new string s3 by appending s2 in the middle of s1.

- s1 = "Ault"
- s2 = "Kelly"

Answer

- First, get the middle index number of s1 by dividing s1's length by 2
- Use string slicing to get the character from s1 starting from 0 to the middle index number and store it in x
- concatenate x and s2. $x = x + s2$
- concatenate x and remaining character from s1
- print x

Given string contains a combination of the lower and upper case letters. Write a program to arrange the characters of a string so that all lowercase letters should come first.

```
str1 = OraNGes
```

Answer

- Create two lists, lower and upper
- Iterate a string using a for loop
- In each loop iteration, check if the current character is the lower or upper case using the islower() string function.

- If a character is lower case, add it to the lower list; else, add it to the upper list
- to join the lower and upper list using a join() function.
- convert list to string
- print the final string

Numbers

Numbers are variables that hold numerical value. In the numerical data class, there are four different sub-classes too. The sub-types of numerical data are

- Integers

Integers are used to represent whole number values like 100. 124. For example,

x = 100

y = 124

- Float

Float is used to represent decimal point values.

x = 10.25

y = 12.30

- Complex Numbers

Imaginary values are represented by complex numbers, and they are identified by the value j at the end of the number.

x = 10 + 5j

- Boolean

Boolean is used for a categorical output, one that helps you get the output in true or false.

- num = 5 > 4
- #num is the Boolean variable
- type(num)
- #the output will be a bool
- print(num)
- #this will print true.

To verify any type of object in Python, then use the **type()** function.

Example

- print(type(x))
- print(type(y))
- print(type(z))

You can also change the Type of number through conversions with the **int()**, **float()**, and **complex()** functions. While different numbers can be changed, complex numbers cannot be changed.

If you wish to include a random number, there is no `random()` function in Python. However, Python contains a `random` module that can be used to generate random numbers.

- `import random`
- `print(random.randrange(1, 10))`

exercises

what are the types of subclasses in numerical elements of Python?

1. Add the appropriate syntax to turn `x` into a floating point number.

`x = 5`

`x = -----(x)`

2. Insert the correct syntax to convert `x` into an integer.

`x = 5.5`

`x = -----(x)`

3. Insert the correct syntax to convert `x` into a complex number.

`x = 5`

`x = -----(x)`

Chapter 3- INTRODUCING LISTS

What is a list

Lists are an important part of the Python language as they are the simplest containers for coded elements. Lists are used to store multiple data at once. A list is created by placing items inside square brackets. These lists created in Python do not have to be homogenous in nature. Therefore, Python lists are very powerful for organizing multiple items inside. A list can contain different data types like Integers, Strings, as well as Objects. While string element in Python cannot be changed, the list variable can be altered and changed according to the needs. After you create the Python list you can change its composition.

How to create a list

A list in Python, can be created by placing items inside []. The items inside the lists are separated by commas.

```
# A list with 3 integers
numbers = [1, 2, 5]

print(numbers)

# Output: [1, 2, 5]
```

The list name is numbers. A list can have any number of items inside it. You can also make a list on Python with mixed data types.

```
# empty list
my_list = []

# list with mixed data types
my_list = [1, "Hello", 3.4]
```

The items in a list are ordered and by this we mean that items have a definite order inside a list and the order does not change. If we add new items to the list, then the new items will be added at the end of the list.

The list can be altered, though. This means that after the list has been created, the elements can be modified, added, or removed. Now that we know how to create list, let us know about organizing and changing a list.

Organizing a list

There are some ways in which you can organize a list.

You can sort the list in descending and ascending order. For this, you need to use the sort() method so that the list is sorted in an ascending order. You can also use the sorted() function to create a new sorted list without modifying the previous list.

Use the reverse=True command with sort() or sorted() to sort the items in descending order.

Before we understand how to change a list and add or remove items from a list on Python, we

should know about the different functions related with it. For example, `append()` is the function to add item to any list.

There is also another function called the `extend()` which adds items of lists and other iterables to the end of the list.

`Insert()` is a function that inserts an item at a specific index. On the other hand, the `remove()` function is a command that removes items from a list. `Pop()` function is a command that returns and removes item at the given index. To remove all items from the list, you need to use the `clear()` command. `Index()` command is used to return the index of the first matched item. `Sort()` command is also a function that we see in the above section.

`Reverse()` command reverses the order of the list.

Changing, adding, and removing elements

Before we know how to change the items of a list, you should first know about the way to access the items of the list. The way to access the items of a list are by referring to their index number.

Print the second item of the list:

```
thislist = ["apple", "banana", "cherry"]
print(thislist[1])
```

The second item is indexed as 1 because the first item is indexed as 0.

You can also specify a range of indexes when accessing the list. When the range is specified, you will get a new list with the items specified in the list.

Return the third, fourth, and fifth item:

```
thislist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]
print(thislist[2:5])
```

If you want to check the presence of an item in a certain list before you change it, then you have to use the `in` keyword to check for it.

```
thislist = ["apple", "banana", "cherry"]
if "apple" in thislist:
    print("Yes, 'apple' is in the fruits list")
```

If you want to change the value of the item, you need to refer to the index number.

Change the second item:

```
thislist = ["apple", "banana", "cherry"]
thislist[1] = "blackcurrant"
print(thislist)
```

This is done if you are changing the value of one item. However, if you are going to change the value of a range of items, then you need to specify the range. For example,

Change the values "banana" and "cherry" with the values "blackcurrant" and "watermelon":

```
thislist = ["apple", "banana", "cherry", "orange", "kiwi", "mango"]
thislist[1:3] = ["blackcurrant", "watermelon"]
print(thislist)
```

If you want to insert new items without changing the order of the existing items in the list, you will need the use of the insert() command.

Insert "watermelon" as the third item:

```
thislist = ["apple", "banana", "cherry"]
thislist.insert(2, "watermelon")
print(thislist)
```

To add elements to a list, you need use the append () command. When you use this function, you add items to the list one by one. It is possible to add multiple elements by using the append() command. But to add multiple elements, you can use the loop method. Lists can also be used to existing lists with the help of the append() function.

```
# Addition of Elements
# in the List
List.append(1)
List.append(2)
List.append(4)
print("\nList after Addition of Three elements: ")
print(List)
```

List after Addition of Three elements:

[1, 2, 4]

```
# Addition of List to a List
List2 = ['For', 'Geeks']
List.append(List2)
print("\nList after Addition of a List: ")
print(List)
```

List after Addition of a List:

[1, 2, 4, 1, 2, 3, (5, 6), ['For', 'Geeks']]

You can also remove items from a list. You can use the del() function and the remove() function to remove items from the list. For example, you can use the del() command to remove items from the list in a one by one manner. On the other hand, if you use the remove() function, you

can remove a specific list item.

```
# deleting the second item
del languages[1]
print(languages) # ['Python', 'C++', 'C', 'Java', 'Rust', 'R']

# deleting the last item
del languages[-1]
print(languages) # ['Python', 'C++', 'C', 'Java', 'Rust']
```

```
languages = ['Python', 'Swift', 'C++', 'C', 'Java', 'Rust', 'R']

# remove 'Python' from the list
languages.remove('Python')

print(languages) # ['Swift', 'C++', 'C', 'Java', 'Rust', 'R']
```

The pop() function can also be used to remove elements from the list. For example,

```
# Removing element at a
# specific location from the
# Set using the pop() method
List.pop(2)
print("\nList after popping a specific element: ")
print(List)
```

```
List after popping a specific element:
[1, 2, 4]
```

If the pop() function is in a default mode, then it removes the last element of the list. However, if you want to remove an element from a specific position, then you can remove with the help of index of the element when it is passed as an argument through the pop() method.

Exercise

- Define a list. What are the items that can be put into a Python list?
- What are the defining characteristics of a Python list?

Write five functions that can be executed in Python lists. What is the use of the following functions?

- Index()
- Sort()
- Clear()

Write a Python program to create a list and then check if the list is empty or not.

```
# Create an empty list
my_list = []

# Prompt the user for the number of elements in the list
num_elements = int(input("Enter the number of elements in the list: "))

# Prompt the user to enter the elements and add them to the list
for i in range(num_elements):
    element = input(f"Enter element {i+1}: ")
    my_list.append(element)

# Print the final list
print("The list is:", my_list)
```

1	l = []
2	if not l:
3	print("List is empty")
4	

Chapter 4- WORKING WITH LISTS

In Python programs, the coder can create variables to store single data records. For such situations, coders can use the Python list data structure. The Python list option allows you to create and manipulate multiple items in one variable. We have already covered some basics about Python lists in the data types related to Python. These Python lists are internally represented as arrays.

Looping

There are many ways of iterating or looping through a list. In this section, you will know about the different ways in which you can loop through a list.

Running a simple for loop

Using a python for loop is a good way of iterating over a list as it is very simple. Python for loops are very versatile and convenient to use. Coders can use the for loop to run the statements contained within the loop once for each item inside the list.

For example

Code

```
fruits = ["Apple", "Mango", "Banana", "Peach"]
for fruit in fruits:
    print(fruit)
```

Output

Code

```
Apple
Mango
Banana
Peach
```

In the above example, the for loop has printed all the items of the list.

List comprehension

List comprehension is similar to the for loop. The difference between the for loop and comprehension is the list comprehension method lets us iterate through the list in a single time. It is a very simple method that coders prefer using to iterate through a list.

Example

Code

```
fruits = ["Apple", "Mango", "Banana", "Peach"]  
[print(fruit + " juice") for fruit in fruits]
```

The key command in the list comprehension method is the for..in command which is enclosed in the () command in square brackets. This is how the list comprehension method is different from the for loop method.

Output

Code

```
Apple juice  
Mango juice  
Banana juice  
Peach juice
```

The results for the iteration of the list is the same as any for loop. However, you can also see the difference in addition of element to the list.

For loop with a range ()

You can also create a for loop with a range () if you want to iterate the list. The range() command generates a sequence of a number of integers from the initial and final index define in the range().

For example

Code

```
range(start, stop, step)
```

The start and step arguments are optional. To generate a sequence you only need the stop argument in the brackets. If you are specifying only one parameter, the stop index, then the function constructs the range object containing all the elements from 0 to stop-1.

Code

```
fruits = ["Apple", "Mango", "Banana", "Peach"]

# Constructs range object containing elements from 0 to 3
for i in range(len(fruits)):
    print("The list at index", i, "contains a", fruits[i])
```

Code

```
The list at index 0 contains a Apple
The list at index 1 contains a Mango
The list at index 2 contains a Banana
The list at index 3 contains a Peach
```

For loop with an enumerated()

Sometimes, you wish to know about the index of the elements in the list that you are accessing. The enumerate() function in a for loop helps you know the same. The enumerate function adds a counter and returns it as something called an enumerate object. When you have to keep track of the number of elements in the list, you can use the enumerate() function to count the number of elements in a simple repetition of the list.

Code

```
fruits = ["Apple", "Mango", "Banana", "Peach"]

for index, element in enumerate(fruits):
    print(index, ":", element)
```

Code

```
0 : Apple
1 : Mango
2 : Banana
3 : Peach
```

For loop with lambda

Python also has a lambda function that helps in iterating a list type data. Python language's lambda function is used as an anonymous function with a mathematical expression which is evaluated and then returned to the list. To use the lambda function, coders need to make a for

loop so that a list of numbers can be iterated. You can do different mathematical operations on the list. For example, you can find the square of each number and then save the list. You can also append it to the list. You can also print a list of squares through this method.

```
lst1 = [1, 2, 3, 4, 5]
lst2 = []

# Lambda function to square number
temp = lambda i:i**2

for i in lst1:

    # Add to lst2
    lst2.append(temp(i))

print(lst2)
```

The for loop is used to iterate through lst1. Each integer is passed through a single iteration and then the list is appended to lst2 with the append() function.

The output will be a lined-out list like the below one.

```
[1, 4, 9, 16, 25]
```

While loop

A python list can also be iterated with the help of a while loop. This is also a simple loop command that most beginners deal with when learning how to code with Python. The term while in the loop stands for some time or time interval. It is referred to as a loop because a piece of code is executed again and again in loop with the command. The while loop executes a piece of code until a certain condition is met.

For example, in the code given below, the condition for the while loop is the length of the list. At first, the i counter is set to zero. The counter then adds 1 to the count each time an item of the list is printed by the loop.

The while loop is ended when the i count exceeds the number of items in the list.

```
fruits = ["Apple", "Mango", "Banana", "Peach"]

i = 0
while i < len(fruits):
    print(fruits[i])
    i = i + 1
```

The code will be given as an output of

Apple
Mango
Banana
Peach

The NumPy library

The methods that have been discussed above are meant for small lists. However, a Python coding professional will be using large and numerous lists daily in his workspace. When you are working with large amount of data, to help increase the efficiency, instead of going for loop commands, you should use the NumPy library. An external library like the NumPy library helps you loop through large data lists. The NumPy library converts the large lists into NumPy arrays.

```
import numpy as np

nums = np.array([1, 2, 3, 4, 5])

for num in nums:
    print(num)
```

1
2
3
4
5

It is even better to use **for num in np.nditer(nums):** command to efficiently work with large lists.

These are some of the ways in which you can loop small and large lists with Python commands and libraries.

Tuples

There are other elements and data types in Python like the tuples that you should know about. Let us talk about the composition of tuples. Tuples in Python are a collection of objects separated by commas. Tuples is often similar to list because of its indexing, nested objects and the repetition that happens. However, Python tuples are immutable while Python lists are mutable under certain circumstances.

A tuples is created by placing all the elements of the data set in parentheses. The elements are then separated by commas. A tuple is a versatile Python data set because it can have any kind of data sets and these could be integer, float, list or string. Placing the elements inside a parantheses is an option but it is done as a good practice.

```
# Different types of tuples

# Empty tuple
my_tuple = ()
print(my_tuple)

# Tuple having integers
my_tuple = (1, 2, 3)
print(my_tuple)

# tuple with mixed datatypes
my_tuple = (1, "Hello", 3.4)
print(my_tuple)

# nested tuple
my_tuple = ("mouse", [8, 4, 6], (1, 2, 3))
print(my_tuple)
```

```
()
(1, 2, 3)
(1, 'Hello', 3.4)
('mouse', [8, 4, 6], (1, 2, 3))
```

When creating a tuple, only including one element inside the parantheses is not enough. Consequently, it is a little challenging to create a tuple with just one element. For the final tuple to be recognized as such, it must have a trailing comma.

```
var1 = ("Hello") # string
var2 = ("Hello",) # tuple
```

You can also use the type () command to know which class of variables an element belongs to.

How to access Python tuple elements

Each element of the Python tuple is represented by index numbers (0,1,). The first element of tuple is 0. You can do indexing in the tuple using the index number. You can use the index operator command to access the elements via their index numbers. If a tuple has eight elements, those will be indexed by the numbers of 0,7. If you try to access an indexed element outside the tuple, it will come back with the IndexError message. We cannot use floats or other kinds as elements of the tuple set to index; the index must be an integer.

```
# accessing tuple elements using indexing
letters = ("p", "r", "o", "g", "r", "a", "m", "i", "z")

print(letters[0])    # prints "p"
print(letters[5])    # prints "a"
```

The (letters[0]) command accesses the first element and the (letters[5]) access the sixth element. You can also do negative indexing with Python tuple.

```
# accessing tuple elements using negative indexing
letters = ('p', 'r', 'o', 'g', 'r', 'a', 'm', 'i', 'z')

print(letters[-1])   # prints 'z'
print(letters[-3])   # prints 'm'
```

The -1 element accesses the last element and the -3 element accesses the third last element. You can even do slicing with the tuple set by using the slicing operator colon.

```
# accessing tuple elements using slicing
my_tuple = ('p', 'r', 'o', 'g', 'r', 'a', 'm', 'i', 'z')

# elements 2nd to 4th index
print(my_tuple[1:4]) # prints ('r', 'o', 'g')

# elements beginning to 2nd
print(my_tuple[:-7]) # prints ('p', 'r')

# elements 8th to end
print(my_tuple[7:]) # prints ('i', 'z')

# elements beginning to end
print(my_tuple[:]) # Prints ('p', 'r', 'o', 'g', 'r', 'a', 'm', 'i', 'z')
```

```
('r', 'o', 'g')
('p', 'r')
('i', 'z')
('p', 'r', 'o', 'g', 'r', 'a', 'm', 'i', 'z')
```

Here, when the slicing action starts, the start index is inclusive and the end index is exclusive.

In the case of tuples, there is no command that can help you add or remove items from the tuple. However, you can count and index the elements as you feel necessary. The my_tuple.count('p') counts the total number of p in the tuple. my_tuple.index('l') returns to the first occurrence of 'l'

in the specified tuple.

```
my_tuple = ('a', 'p', 'p', 'l', 'e',)  
  
print(my_tuple.count('p')) # prints 2  
print(my_tuple.index('l')) # prints 3
```

You can also iterate through a tuple just like you do for a list. Similar to a list, you can also iterate through a tuple with the help of a for loop.

```
languages = ('Python', 'Swift', 'C++')  
  
# iterating through the tuple  
for language in languages:  
    print(language)
```

```
Python  
Swift  
C++
```

The in keyword can also be used to determine whether an element is present in the tuple or not.

```
languages = ('Python', 'Swift', 'C++')  
  
print('C' in languages)    # False  
print('Python' in languages) # True
```

C is not present in the Languages tuple, so the command returns as false. However, the language Python is present, so the output returns as true.

The tuple data set resembles the list data set in Python. However, there are certain benefits of using tuple which makes it a better choice than list. Lists and tuples are used for different purposes. Tuples are mostly used for heterogenous distribution of data and the list type is used for homogenous distribution of data. Iteration is something that happens both with list and tuples. Since tuples are strictly immutable, tuples are easily iterated in a fast manner. So, if you want to boost the performance of a code with the use of Python, you should choose to organize elements with tuples.

Tuples are also used as key for dictionaries in Python which does not happen with Lists. The type of data set that you use depends on the type of data you are using. If the data that you have

is immutable, you would need to use Tuples than lists. The decision ultimately rests on the coder. If you want to protect a data set from overwriting, you should use the tuples as data sets as they do not change.

Making numerical lists

To make numerical lists on Python, you can use the range() command. The range function requires a group of numbers and returns them as a sequence. The numerical list starts from 0 by default and then ends at any specified number with a 1 by 1 increment. When making numerical lists you can change the starting number and the increment that goes on step by step.

```
def getnums(s, e,i):  
    return list(range(s, e,i))  
  
# Driver Code  
start, end, intval = -3, 6,2  
print(getnums(start, end,intval))
```

```
[-3, -1, 1, 3, 5]
```

A list of numbers can be generated using the random module as well. A random number is generated from the list using the random module. The randrange method is used for this process and the coder gives the start, end and interval values to determine the final values of the numerical list.

```
import random  
def getnums(s, e,i):  
    return (random.randrange(s, e,i))  
  
# Driver Code  
start, end, intval = 3, 16,2  
print(getnums(start, end,intval))
```

```
7
```

You can also use the numpy.arrange function to create a numerical list.

```
import numpy as np
def getnums(s, e,i):
    return (np.arange(s, e,i))

# Driver Code
start, end, intval = 3, 16,2
print(getnums(start, end,intval))
```

```
[ 3 5 7 9 11 13 15]
```

IF STATEMENT

While we write programs in Python, there could be a number of decisions that the program has to take. When we need Python written codes to make some choices, we put forth certain conditions. Depending on the conditions, different statements are executed. The IF statement is a conditional statement that is used for conditional execution of code. An if statement is the control structure that controls the flow and output of the program. The IF statement can be combined with different operators for different outputs. These conditional statements can be combined using or, and keywords. An IF statement is a simple statement as it evaluates the data given to it and then makes a choice on it. In the most basic form, an if statement looks like

```
#!/usr/bin/env python3
if <condition>:
    <statement>
```

Examples of if statements

```
#!/usr/bin/env python3
>>> x = 3
>>> if x < 10:
...     print('x below ten')
...
x below ten
>>> if x > 10:
...     print('x is greater than ten')
...

```

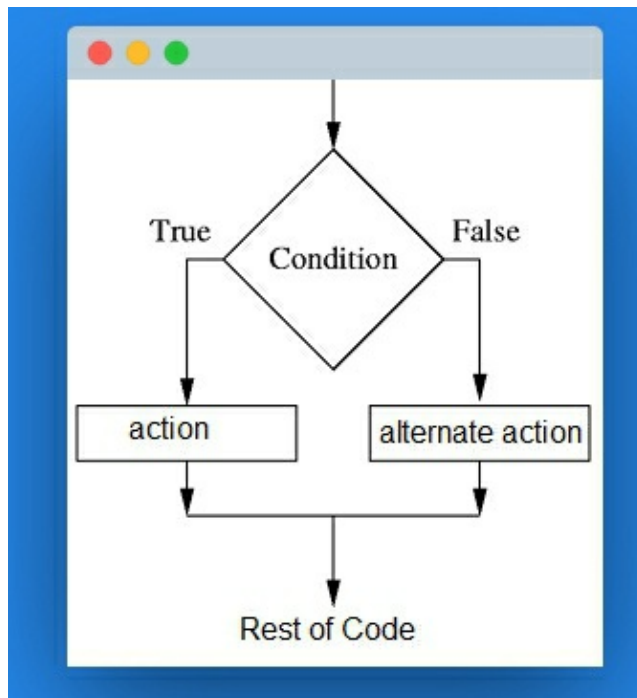
More than one condition of an if statement can be made with the help of the and keyword.

An if statement does not always come up as a single statement. At times, the if statement can be

block. A block is created when there are more than one statements for the condition. A block of statements are however seen as one complete element in Python. So, when the conditions are met, the entire block code is executed. Example of an if statement block is as follows-

```
if <condition>:  
    <statement>  
    <statement>  
    <statement>  
  
<statement> # not in block
```

The flowchart of the if statement is as follows-



First the test expression is checked by the program. Once the test expression is fulfilled, the body of the if statement is executed.

Conditional tests

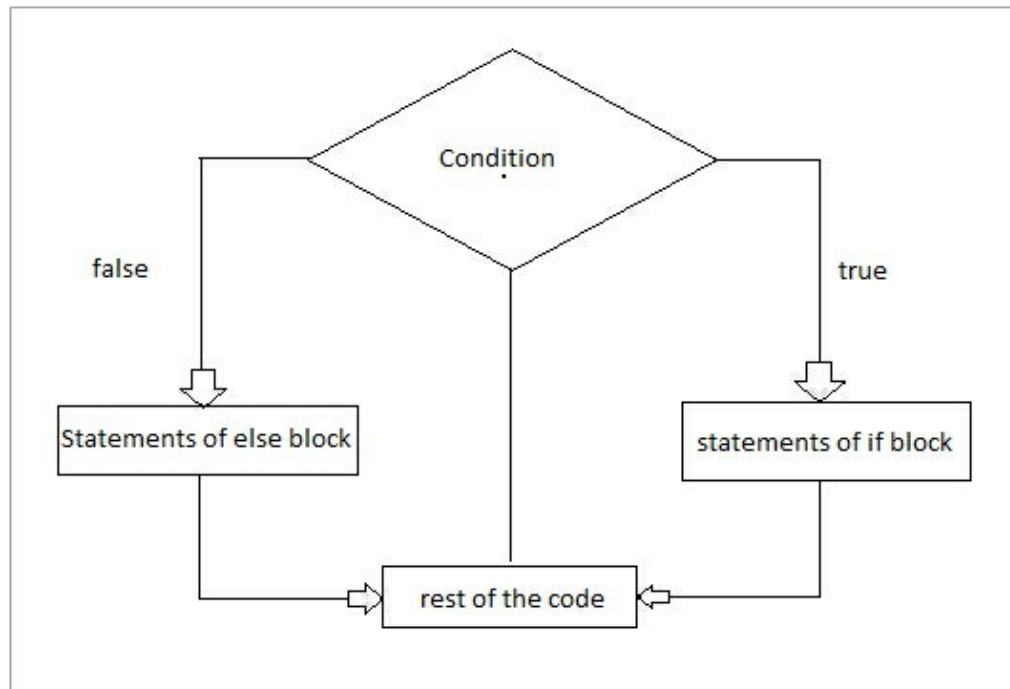
Conditional statements are made to handle conditions and make decisions in a programmed code. There are different types of conditional statements that will help you get the code executed in a certain situation. There are three key conditional statements in Python. The If statement is already discussed. Another conditional statement is the If else statement. Elif statement is also a type of conditional test done in Python. In this section, we will learn about the different conditional statement apart from If statement.

If else statement

The if else statement itself says if a given condition holds true, then the statement present inside

the if block are executed. On the other hand, if the condition holds false, then the else statement is executed. The if else statement works when it evaluates Boolean expressions.

```
If (EXPRESSION == TRUE):  
    Statement (Body of the block)  
else:  
    Statement (Body of the block)
```



When the system evaluates the condition for the if else statement, the controller will first come to the if condition and then it will move to see if the condition holds true. Then the if block will be executed once the condition is true. However, if the condition is not true, the alternative else block will be evaluated and executed by the controller. Once the if or else statement is executed, then the rest of the code that is excluded from the if else statement will be executed.

```
num = 5  
if(num > 10):  
    print("number is greater than 10")  
else:  
    print("number is less than 10")  
  
print ("This statement will always be executed" )
```

The output for the above code will be number is less than 10.

In the above code, there is a variable defined as num. The value of this num variable is 5 and in the if statement we are checking if the number is greater than 5 or not.

The if block will be executed if the number is larger than 5. The if statement will fail and the else block will be executed if the value is less than 5.

Now, let us look at the elif statement.

Elif statement

Elif statement is a modification of the if statement that is used to check for multiple statements if one given condition is not fulfilled. The elif statement is almost the same as an if else statement. However, there is one distinct difference for the statement. For example, the elif statement checks the alternative conditions while the if else statement only executes the else statement, it does not check the statement. Moreover, if else statement works on a single statement. The elif statement works on multiple conditions and evaluates the validity of multiple statements.

```
num = 10
if (num == 0):
    print("Number is Zero")

elif (num > 5):
    print("Number is greater than 5")

else:
    print("Number is smaller than 5")
```

Output

Number is greater than 5.

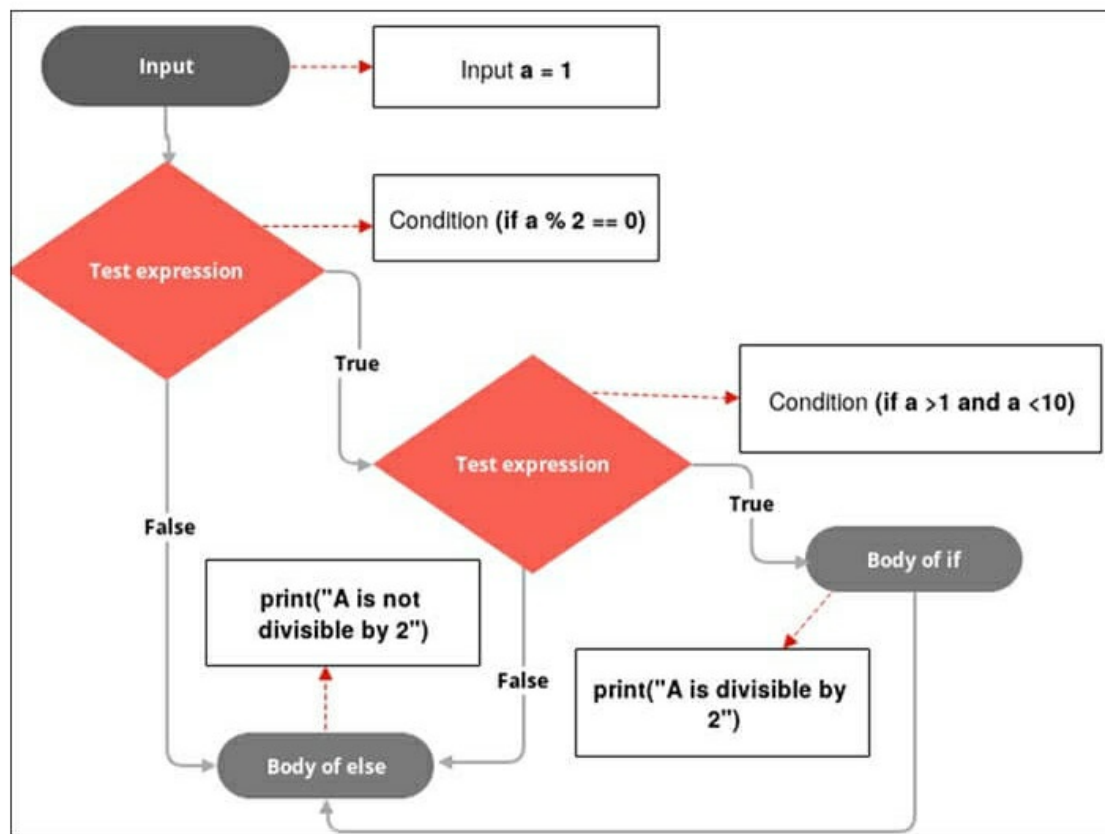
The variable in the above statement is num. The value of this variable is 10 and the if statement checks this statement. If the condition is fulfilled, then the if statement will be executed. The controller will examine the elif condition if the condition of the statement changes to false. The block of code for the elif condition is executed if the elif condition is true.

If the condition for the elif statement is false, then the else statement is executed.

Nested if else statements are a special kind of statement, in which an if statement is present inside another if statement. Python provides provisions to design statements like these and this helps coders in checking for multiple statements at once. An if statement is present inside another if statement and further this statement is present inside another if statement.

```
if(condition):
    #Statements to execute if condition is true
    if(condition):
        #Statements to execute if condition is true
    #end of nested if
#end of if
```

Flowchart of nested if statement



Example of nested if statement

```

num = 5
if(num > 0):
    print("number is positive")

if(num < 10):
    print("number is less than 10")
  
```

Output will be number is positive

Number is less than 10

The declared variable in the above example is num and the value is 5.

First the controller will check the first if statements and then check for the statement condition if it is true. The if block will be executed if the condition is true. On the other hand, if the condition is not true, then the block will not be executed. Once one if statement is executed, the other if statements are also checked and executed accordingly.

If statements

Let us start with the execution of basic if statement. The basic if statement is a basic condition that is being evaluated and executed accordingly. For example,

```
if <condition>:  
    <expression>
```

When the condition is evaluated by Python, it will become either true or false with the Boolean expression. If the condition is true and met then the expression linked with the condition will be executed. However, if the if condition is false, then the expression linked with it will not be executed. Python is a flexible language that you can use to decide which conditions and expressions are coded with the if statement.

Using IF with lists

We can use list comprehension function to create lists in Python. This can also be done with the help of if statements. The if statement is used to provide the condition that can be applied to the list while it is being created.

How does the if statement work in Python?

The if statement helps put logic into the statements coded in Python. The conditions set for Python coded if statements are pre determined by the programmers who use it. If statements are mostly used to check the validity of an event. The if statement can stand alone and get executed when the conditions are fulfilled. However, other conditions that are the modifications of the if statements like if else and elif are also used to back the statement. A combination of if, if else and elif statements set the other rules in place. Moreover, the programmer can use keywords like not, or, and, in to set and streamline the conditions of the statements.

Python also allows the programmer to use the if statement directly with the control flows. The if statement can also be used in conjugation with the for loop.

Using in, and, or keywords with the if statement

To determine whether the specified item or element is present in a list, combine the in keyword with the if statement. For example, below code has the usage of in-

```
myList = ['Python', 'MUO', 'Hello']  
if ('Python') in myList:  
    print("It's in the list")  
Output: It's in the list
```

You can also use the and expression with the if statement to check for more than a single item.

```
myList = ['Python', 'MUO', 'Hello']  
if ('Python' and 'Hello') in myList:  
    print("Hello Python")  
Output: Hello Python
```

If you are checking for more than one items on the list, and see if there is one or the other item, then you need to use the or keyword.

```
myList = ['Python', 'MUO', 'Hello']  
  
if ('Python' or 'Bags') in myList:  
    print("One of them is on the list")  
  
Output: One of them is on the list
```

You can even use the if statement in line with the for loop. To see how it works with the for loop, you can set the conditions while looping through a list.

```
myList = myList = ['Python', 'MUO', 'Hello']  
myList2 = ["Fish", "Gold", "Bag"]  
if len(myList) == 3:  
    for items in myList:  
        print(items)  
else:  
    for items2 in myList2:  
        print(items2)
```

The code checks if the length of the list is exactly three and loops through it if the statement holds true. If the condition is not met, then the else statement is executed.

Lets see some more examples of how the if statement helps in the process of list comprehension.

Using if with list comprehension

In the list comprehension method, conditionals like if statement can be used to modify existing list. for example,

```
number_list = [ x for x in range(20) if x % 2 == 0]  
print(number_list)
```

The output

```
[0, 2, 4, 6, 8, 10, 12, 14, 16, 18]
```

The number_list is made up of the numbers falling in the range of 0-19 and also the ones that are divisible by 2.

You can also run nested if statement with the list comprehension method.

For example,

```
num_list = [y for y in range(100) if y % 2 == 0 if y % 5 == 0]
print(num_list)
```

```
[0, 10, 20, 30, 40, 50, 60, 70, 80, 90]
```

You can also run if else statements with list comprehension methods.

For example

```
obj = ["Even" if i%2==0 else "Odd" for i in range(10)]
print(obj)
```

```
['Even', 'Odd', 'Even', 'Odd', 'Even', 'Odd', 'Even', 'Odd', 'Even', 'Odd']
```

Exercise

- What is the need of conditional statements in any python program? What are the conditions for the execution of an if statement?
- How is an if else and an elif statement different from each other?
- How do you describe a nested if statement?
- How is Python list comprehension useful?
- Filter only even numbers and find their squares with list comprehension of the following list.

List 1- [7,2,6,2,5,4,3]

```
list_1 = [7, 2, 6, 2, 5, 4, 3]
list_2 = [ x * x for x in list_1 if (x % 2 == 0) ]
print(list_2)
```

```
[4, 36, 4, 16]
```

DICTIONARIES

Now that we have known most of the data types on Python, there are some other types of data sets like dictionary. Like a list in Python, the dictionary is also a list of objects. Dictionary is a mutable list of objects and the composition can be changed at any point in time. The dictionary in Python is a very dynamic data set too because it can shrink and grow at any point in time if the programmer wants to do so.

A dictionary also has the ability to be nested. This is an ability that even lists have. This means that a dictionary can have another dictionary inside it. A dictionary can also have another list inside it. However, there is a fundamental difference between dictionary and list. While lists are accessed by the position of the elements in the list, the dictionary is accessed by keys. When the coder creates a dictionary, he also creates a key to the dictionary with which the elements inside the dictionary can be accessed. Python implements data in different ways and the dictionary is the way of creating an associative array for the data set. The dictionary in any form within the Python code consists of key value pairs. Each key value pair of the Python dictionary maps one key to one particular value.

The set of a dictionary is defined by putting a set of values within comma separated list inside curly shaped brackets. A semi colon is used to separate each key from its value. Python dictionaries are useful to the coders because they help us associate a particular, unique value to a key. You can therefore quickly access the key and the value through the dictionary. Lists can also be used for the same reason but lists are shorter and slower for the same operation. If the coder has coded in many objects in the Python code, he would want to lookup for a certain object in a collection of objects. The same thing can be done with the help of the keys and values stored in the Python dictionary. The speed in searching for something with the help of a dictionary is there because the keys of a dictionary are hashable. Every immutable object placed in the dictionary is hashable. So, we can pass the immutable object to the hash function. This returns the hash value to the object. The values hashed in the dictionary are then used to look for a value associated with a certain value key.

Now that we know about the dictionary, let us further know about how to work with a dictionary. Before we go into the working of the dictionary, let us talk about the creation of a dictionary.

There are two methods of creating a dictionary in Python. You can either create a dictionary with the help of curly brackets or with the dict() function.

```
# Create a dictionary
dictionary = {} # Curly braces method
another_dictionary = dict() # Dict method

# Are the above dictionaries equivalent?
print(type(dictionary))
print(type(another_dictionary))
print(dictionary == another_dictionary)
```

Both dictionaries that will be created by two separate methods will have the same data type.

After creating the dictionary, you have to add different elements to the dictionary so that it is no longer empty. You can add values with the help of square brackets. The addition happens in the dictionary[key]= value format. After this, we can access the dictionary to know the different values.

Suppose you create a dictionary with the following values, the code will be

```
capital_city = {"Nepal": "Kathmandu", "Italy": "Rome", "England": "London"}  
print(capital_city)
```

The output for the dictionary set will be

```
{'Nepal': 'Kathmandu', 'Italy': 'Rome', 'England': 'London'}
```

The dictionary so created is named capital_city and there are three keys and three associated values. The three keys are Nepal, Italy, England and the associated values are Kathmandu, Rome, and London. In this example, the keys and the values both are in the form of string, you can also use other data sets for creation of a dictionary.

Working with dictionary

Let us now learn how to operate the dictionary and how to work with it. After you have created a dictionary, you can also add elements into it. This can happen by putting the name of the dictionary within the [] brackets.

```
capital_city = {"Nepal": "Kathmandu", "England": "London"}  
print("Initial Dictionary: ", capital_city)  
  
capital_city["Japan"] = "Tokyo"  
  
print("Updated Dictionary: ", capital_city)
```

```
Initial Dictionary: {'Nepal': 'Kathmandu', 'England': 'London'}  
Updated Dictionary: {'Nepal': 'Kathmandu', 'England': 'London', 'Japan': 'Tokyo'}
```

In the above example, an additional key and value has been added to the dictionary.

You can even take up a function to change a certain value in the dictionary. For example, you can change the value with the help of the following commands.

```
student_id = {111: "Eric", 112: "Kyle", 113: "Butters"}  
print("Initial Dictionary: ", student_id)  
  
student_id[112] = "Stan"  
  
print("Updated Dictionary: ", student_id)
```

The output

```
Initial Dictionary: {111: 'Eric', 112: 'Kyle', 113: 'Butters'}  
Updated Dictionary: {111: 'Eric', 112: 'Stan', 113: 'Butters'}
```

In this dictionary, initially, the value 113 was associated with the name Kyle. Now, it is associated with Stan.

Now let us know about how to access elements from a dictionary.

In the system of Python, we use keys to access the associated values inside the dictionary.

```
student_id = {111: "Eric", 112: "Kyle", 113: "Butters"}  
  
print(student_id[111]) # prints Eric  
print(student_id[113]) # prints Butters
```

If you try to access the value of the key that does not exist inside the dictionary, then you will only get an error message returned to the screen.

You can also remove elements from the dictionary and change the length of the set. To remove an element from the dictionary, you use the del statement.

```
student_id = {111: "Eric", 112: "Kyle", 113: "Butters"}

print("Initial Dictionary: ", student_id)

del student_id[111]

print("Updated Dictionary ", student_id)
```

```
Initial Dictionary: {111: 'Eric', 112: 'Kyle', 113: 'Butters'}
Updated Dictionary {112: 'Kyle', 113: 'Butters'}
```

The code `del student_id[111]` has removed the key associated with the value of 111. You can even delete the whole dictionary with the help of the `del` statement.

```
student_id = {111: "Eric", 112: "Kyle", 113: "Butters"}

# delete student_id dictionary
del student_id

print(student_id)

# Output: NameError: name 'student_id' is not defined
```

The error message as the output of the code run is because the `del` statement was used to delete the entire dictionary and there is nothing of this name that exists in the repositories now. So, when there is a run to check for the same dictionary again, an error message comes through.

You can even iterate through a dictionary like you iterate through a list. you can use a `for` loop to iterate through a dictionary the same way that you do for a list.

```
# Iterating through a Dictionary
squares = {1: 1, 3: 9, 5: 25, 7: 49, 9: 81}
for i in squares:
    print(squares[i])
```

```
1
9
25
49
81
```

The above is a iterated list of the elements inside a dictionary.

Now, let us know about the different methods related to operations in python with dictionary.

When you use `all()` function, you get the output `True` if all the keys of the dictionary are true. The returning output also comes back as `true` if the dictionary set is empty.

The `any()` returns as `true` in the output when any key of the dictionary is true. On the other hand, if the dictionary is empty, it does not return back with `true`, the output is `false`.

When you run the `len()`, the output is the length of the item list in the dictionary or the number of the items in the dictionary. There is also a `sorted()` function which gives a sorted out list of the elements inside the dictionary.

`Clear()` is also a function that you can run with your dictionary. For example, if you run it on a dictionary with elements, you will get an empty dictionary because it removes all the items in the list.

The `keys()` function is also a very useful method that you can run on the dictionary. When this function is run, the output returns with the new object of the dictionary keys.

Then there is the `value()` function which returns with a new object of the dictionary's values.

Moreover, just like the list comprehension, you can also do dictionary comprehension. You can use for loops to do dictionary comprehension.

```
[x for x in grocery_items.values()]
Out:
[[3.99, 4.99, 5.5], 1.49, 4.5, 2.5, 3.99]
```

All these commands and functions are used to manipulate extensive lists of item, sort them, add them, remove the and find a definite output from them.

Exercise

- What are dictionaries? How are dictionaries different from lists?
- What are the different methods in which you can operate in dictionaries.
- Sort the following dictionary in order of values.

```
{'ravi': 10, 'rajnish': 9, 'sanjeev': 15, 'yash': 2, 'suraj': 32}
```

Answer will be

```
{'rajnish': 9, 'ravi': 10, 'sanjeev': 15, 'suraj': 32, 'yash': 2}
```

- Find the sum of all the values present inside the below dictionary

`{'a': 100, 'b':200, 'c':300}`

You can use the sum function to find the sum of all the above values. The output should be 600.

Sum : 600

INPUT AND WHILE LOOPS

How Input works

Input is necessary in case of any program and any language that we are trying to use on the compute. Python also needs to use input data to create code. There are different ways in which programmers can give input to Python. There are two functions that can be used to read data into the Python programming space. One is the raw input() and the input(). The results of the input functions can be stored as variables.

The input() takes input from the user and then registers it.

```
name = input("Enter your name: ")
print(name)

# Output:
# Enter your name: James
# James
```

The input() is executed in the form of input([prompt]).

Developers usually need to interact with the language and the system so that they can give some kind of input or show some kind of result and then manipulate that result. For this different input methods are used. Most of the programs run on computer ask for input through a dialog box where the person can type or click to do certain actions.

The input() is a function that takes data from the user and then converts it into a string. The returned object from the input will always come back as <class 'str'>. The input() does not evaluate what is enclosed, it just returns the entire expression as a string. The input function is an inbuilt function that stops the program when it is activated. It then waits for the user to give a command. The program is again activated when the user presses enter and the program shows what the user has typed. It is an easy function to use and has basic purpose of reading data into the program.

```
inp = input('STATEMENT')
```

Example:

```
1. >>> name = input('What is your name?\n')      # \n ---> newline
---> It causes a line break

    >>> What is your name?
    Ram
    >>> print(name)
    Ram

    # ---> comment in python
```

We have already known that the prompt given during the input function is an optional one. This means that the text message displayed on the output screen to ask for an input from the user is optional.

Raw input()

Now that we know what input() is, let us know about raw input(), a variation of the input(). The raw input() works in older versions of the Python language. This function takes what is typed from the keyboard and then the function converts it into a string output. The string is then returned to the variable in which the data will be stored.

```
# Python program showing
# a use of raw_input()

g = raw_input("Enter your name : ")
print g
```

Typing of data in the raw input() function comes to an end when the user presses the enter key. The function can also be used to give an input of numeric data. To enter numeric data, we can use the method of typecasting. You can use the int(input()), and float(input()) commands to put in the data.

There are different ways of giving input to Python. For example, raw input() is used to feed in an optional prompt argument. It also strips the trailing newline from the character it returns.

On the other hand, print is just a thin wrapper that helps in formatting the inputs. It is used to call the write function of a given object.

```

# basic method of input output
# input N
n = int(input())

# input the array
arr = [int(x) for x in input().split()]

# initialize variable
summation = 0

# calculate sum
for x in arr:
    summation += x

# print answer
print(summation)

```

There is a faster method of input that you can use for giving data in python. In Python 2.7, there are commands of inbuilt stdin and stdout. The sys. Stdin is a file object. It is a file object that can be created to read input from a file. In this case, the file is a standard input buffer. You can also use the stdout. Write ('D\n') for faster input on Python 2.7. However, if you want to complete the input process in the fastest way, you should use the stdout.write("".join(list-comprehension)) command. In this method, the input size decides the level of memory usage.

How does the input function work?

```

# get input from user

inputString = input()

print('The inputted string is:', inputString)

```

```

Python is interesting.
The inputted string is: Python is interesting

```

For different types of data, the input process is different. For example, if you want to input a list of integers into the system, you need to list out the integers in a single line.

1 2 3 4 6 7 8 9 and then arrange it as an array. The programmer creates a function called get list() and then uses it to make the input.

```
import sys
def get_ints(): return list(map(int, sys.stdin.readline().strip().split(' ')))

Arr = get_ints()
```

If you have to make the same input again and again, you do not have to make an input of the same values again and again. You can simply use this function in place of the input command, and it will use the specified numbers as inputs.

On the other hand, if you want to take input from a string, you will need to use a different command. Suppose you are using a string of words as an input and want them to be stored in a single variable; you will need to use the `get_string()` and put the words in the bracket.

```
import sys
def get_string(): return sys.stdin.readline().strip()

string = get_string()
```

After this, you do not have to write the words again and again. You can simply do the input by calling on the `get_string()` for the specified words.

These are the basic ways of making an input and using the input function in Python to add data to the workspace. Taking input with the help of the input function of Python is easy. You need to specify the data very clearly for it to be used as an input, especially when the input repeats. Depending on the version of Python you are using in your system, the input command changes. The input pace is very fast in the Python 3.0 version of the latest release.

Exercise

- Take two numbers as input and find out the result for sum, multiplication, and division.

```
a = int(input("a = "))
b = int(input("b = "))

sum = a+b
mul = a*b
div = a/b

print("Sum is", sum)
print("Multiplication is", mul)
print("Division is", div)
```

- Read the content of one file to another new file through Python.

Suppose it is a text file named `demo.txt`, and the new file will be `new.txt`. You need to read data

from the existing file and write it on the new file and then make sure that the new file displays this information.

```
# reading data from demo file
f = open("demo.txt",'r')
z = f.read()
print("Data in demo.txt file")
print(z)

# writing data to new file
f = open("new.txt",'w')
f.write(z)

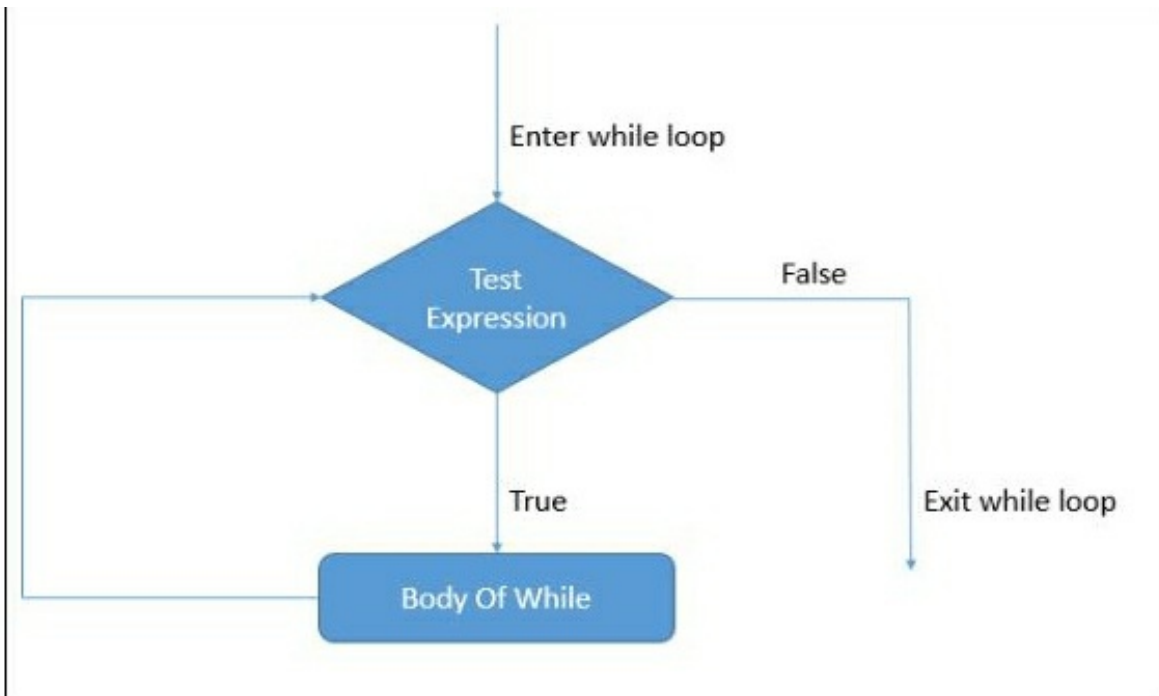
# reading data from new file
f = open("new.txt",'r')
z = f.read()
print("\nData in new.txt file")
print(z)
```

Introduction While Loops

Before knowing what while loops are in python, let us first know what while loops are in general. The while loops are a kind of loop that is used to repeat a block of code a number of times. The number of repetitions is not specified and is unknown. Since this repetition takes some time, it is called a while loop. If the number of times for which the code is to be repeated, then we can repeat it in a for loop. Usually, python commands use for loops. However, in certain situations, you also get to repeat code in while loops.

You can use loops for the repetition, but loops also have their own advantage. For example, the while loop repeats the code as many times as needed until the task is achieved. The while loop can also convert into an infinite loop if the values specified within the loop are not modified by the action specified by the command. After this, the loop will run forever.

According to the rules around python language, the while loop runs until a specific condition is not met for the variables and action.



Now that we know what while loops are and why they are called the name of while let us know about the characteristics of the while loops. For example, the while loop has a break statement.

Break statement for a while loop in Python

The break statement is used to exit from the while loop after a certain condition is met by the program.

```
colors = ["red", "green", "yellow", "black"]
for x in colors:
    print(x)
    if x == "green":
        break
```

```
red
green
```

In this case, the while loop executes until x has the value of three. The condition is that the value should be 3. So, when the value becomes equal to three, the while loop breaks on its own.

While loop with else

The while loop can also be done with the else statement. For example, you can use Python language to add an else statement at the end of the while loop. The else statement is executed when the while loop ends naturally.

```
x = 1
while x<5:
    print(x)
    x = x+ 1
else:
    print("Done")
```

```
1
2
3
4
Done
```

In this loop, the repetition will continue until the value of x is 5. After the value of x is 5, the while loop is broken, and the else statement is executed when the value reaches the threshold of 4. The else block of code is not executed if the while loop gets ended by a break statement.

The continue statement

The continue statement can also be paired with the while loop. By using the continue function, we can stop with the current iteration and start with the next loop of data for iteration.

```
i = 0
while i < 6:
    i += 1
    if i == 3:
        continue
    print(i)
```

Infinite while loop with Python

If the condition of the loop is always true, then the while loop iterates, and it runs for infinite times.

```
age = 32

# the test condition is always True
while age > 18:
    print('You can vote')
```

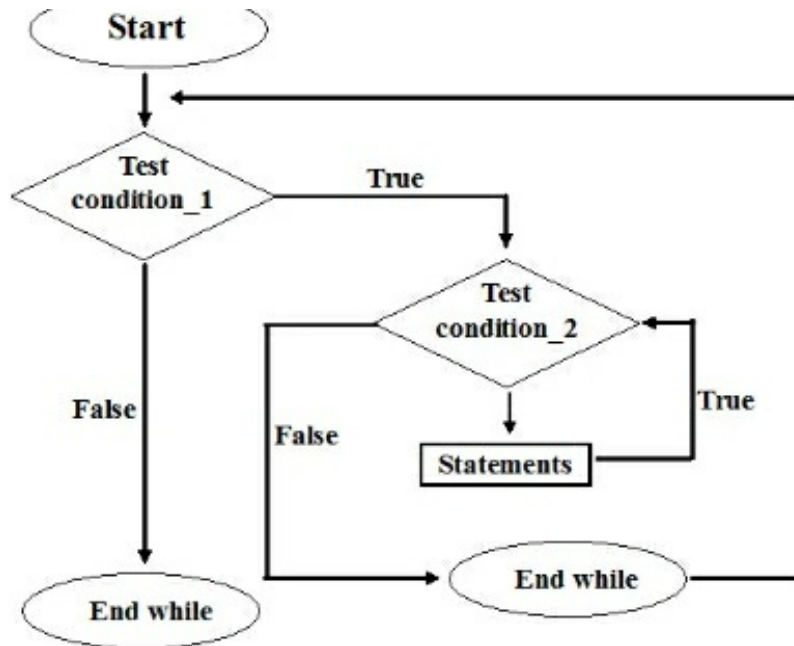
Here the statement always holds true, and hence the while loop is not broken. The while loop here becomes an infinite loop.

Nested while loop

Much like the use of a dictionary within a dictionary, the nested while loop structure allows the use of a while loop inside a while loop. The structure of nested while loops is as follows-

```
while expression:
    while expression:
        statement(s)
statement(s)
```

Flow chart of nested while loops



In a generic nested while loop, there is one outer while loop, and then there is one inner while loop.

At first, the initial outer while loop is evaluated and expressed. When the statement of the outer loop holds true, then the control shifts to the inner loop. The inner while loop then jumps to completion as the outer loop is verified. However, if the test expression for the outer loop is false, then the control shifts from the inner loop to the outer loop, and the outer loop is again executed. This flow and shift of control from the outer loop to the inner loop persist until the result for the outer loop is also false.

There are one-line while loops, too, that depend on single lines of code. To create one line while loops, you need to write the code in a single line. There are many ways of writing a single line while looping. If the loop body consists of one statement, you can write it onto one line. This prints the command and keeps it like a one-line while loop until there is interference. If the loop body consists of multiple statements, you can use the semi-colon to separate the different statements. This separates the statements and also runs one after the other so that one gets executed after another.

On the other hand, if there are nested statements in one while loop, you can use another way of converting it into a one-line while loop. When the inner compound structures are replaced by the ternary operator, you will be able to create a one-line loop out of the compound nested statements.

Examples

Just writing one single line of code for a task in Python is the best way of creating a one-line while loop.

```
while True:
    print('hi')
...
hi
hi
...
...
```

In the multi-statement setup, you need to use the semi colons to separate the commands so that one statement is executed after the other. The use of the semicolon is used to compress multiple statements into a single one-line while loop.

```
c = 0
while c < 10: print(c); c = c + 1
...
0
1
2
3
4
5
6
7
8
9
...
...
```

You can also do the same thing for nested while loops with compound statements. If there are compound statements in one while loop, too, you can change then into a one-line while loop.

```
# YES:
if expression: print('hi')
```

If you want to put multiple statements in one while loop, you can also create a simple one-line while loop.

```
# NO:  
while expression1: if expression2: print('hi')
```

You can replace the if expression2 with a ternary operator and use the expression rather than the compound statement inside it.

```
# Method 3: One-Line While Loop + Ternary Operator  
while True: print('yes') if True else print('no')
```

These are the different ways in which different types of while loops can be created and used. At the same time, loops are used to iterate data sets an unknown number of times until the condition of the statements is fulfilled. These loops can also be modified into different kinds of loops depending on what other statements and conditions are linked with them through Python.

Exercise

- Write a while loop that adds all the numbers to 100 and finds the sum.

```
while counter <= 100:  
    total= total+counter  
    counter= counter+1
```

-
- What are the different types of while loops?
- How is a while loop different from a for loop?
- Explain one method of creating one line while looping.

FUNCTIONS

Definition

In the arena of Python development, you will also come across Python functions. Python functions are codes that are used to perform any given task. If one task has to be defined with the help of Python language, then the user needs to define the Python function related to it. Python functions are used to modularize programs so that the programmer can easily manipulate them according to arising needs. All the variables that are created by the functions can be defined as local variables. These variables are only known to the function under which they are declared. Learning about functions and using functions is very important if you are trying to learn about modular and reusable programming. In this chapter, we will learn about defining functions, calling a function, and how it operates. Functions are used a lot in Python because the programming language has a DRY principle.

The DRY principle means you do not repeat yourself. For this, you need to consider a scenario under which you perform a specific action many times. This will be very repetitive when coded. So, in such cases, to define the actions, you need to use functions. We define an action and the code associated with it using a function and then use it again and again wherever that particular task is supposed to happen. The definition of a function increases efficiency and reduces errors in the code. The coded components are also highly reusable if we look at these functions individually.

Once you have created a function, you can call and use it anywhere in the Python code. Before we know more about the functions present in Python, we should know about the types of functions present.

Types of Functions

The types of functions are as follows-

- Built-in function

The built-in functions are the ones that are predefined and come along with the Python module. The built-in function is used with Python language without any other customization. The built-in Python functions are usually `range()`, `id()`, `type()`, etc.

On the other hand, there are user-defined functions that are more customized.

- User-defined functions

User-defined functions are functions defined by the user according to the requirements and expectations of the programs. User-defined programs are created for specific tasks according to the goal of the program.

In the next section, we will learn about the creation and other operations related to functions. After the creation of the function, code has to be written to execute the function properly.

creating a function

to create a function, you need to follow a certain syntax. The syntax is as follows-

`def name(arguments):`

Statement

return value

There are some steps that you need to follow to create a function. For example, you need to use the `def` keyword to define the function. You also need to pass the number of parameters as per the requirements of the code. The passing of these parameters is optional and not entirely necessary.

After passing the parameters, you also need to define the function body with a block of code that will be associated with the function later. The block of code that you are going to define for the function should be associated with the action that you wish to complete with the help of the function. You do not need to define a function by any kind of brackets; you can simply define them with the help of an indentation that puts different code blocks of the function separately. If you do not separate the code blocks properly, then you might get an error code displayed when the function is executed.

```
def function_name(parameter1, parameter2):  
    # function body  
    # write some action  
    return value
```

Let us know about the terms associated with the function creation process and its syntax. In every function, there is a `function_name` which defines the name of the function created. Similarly, you have the parameter, which is the value passed to the function. There could be any number of parameters defined for a function, and it depends on the requirements made for the function. The parameter is a criterion needed by the function body as it uses the parameter to fulfill an action. The body of the function is defined by the block of code that defines and performs the action associated with the function and data given. Once the function is performed, there is a return value attached to it.

The return value is the final output of the function that the coder gets when the function is executed while fulfilling all the conditions linked with it. While creating and defining a function, you need to use the `def` keyword and also the `return` keyword. However, the `return` keyword is only optional. Functions are created to define common tasks and codes linked with them and then to reuse pieces of code in different projects. The use of functions enhances the modularity and reusability of Python code.

How to create a function with parameters

```
# function  
def course_func(name, course_name):  
    print("Hello", name, "Welcome to PYnative")  
    print("Your course name is", course_name)  
  
# call function  
course_func('John', 'Python')
```

```
Hello John Welcome to PYNative  
Your course name is Python
```

Now, let us learn how to call a function. Once a function has been defined and created, it can be called by its name. The name and calling of a function are very important for its reusability. The easiest and most common way of calling a function is by its name. On the other hand, we can also call a certain function through another program by importing it. To call a function, you need to use parenthesis around the function name. Additionally, if there are other parameters attached to the function, then the function needs to pass them when being called.

The parameters should pass through the parenthesis that is around the function.

```
# function  
def even_odd(n):  
    # check numne ris even or odd  
    if n % 2 == 0:  
        print('Even number')  
    else:  
        print('Odd Number')  
  
# calling function by its name  
even_odd(19)  
# Output Odd Number
```

Calling a function can also happen through a module. You can use the built-in module of the Python language and call functions that are already defined. Python has a random module that is used to generate random data and functions. You can choose from the random data and functions generated by the module. The built-in module has many built-in functions that are used to operate with random data so that there is the generation of valid output.

To import a function from a Python module, you need to define the import statement and use it to import a specific function. After this, you need to call that imported function by its name.

Defining doc strings

In Python, we have to also work with docstrings. The use of docstrings is done right after a certain function is defined so that an action can be done with them. Docstrings are nothing but a well-defined documentation strings. The docstring is a descriptive text that is written by the coder to define what a particular code block does in a function. The docstring is written in source code, and it is written right after a function is defined. The docstring gives further definition to a Python code. The docstring is defined using triple double quotes or triple single quotes. To access an already created doc string, you need to use a doc attribute for accessing any object like

a list, tuple, dict, or any other special user-defined function.

The single-line docstring is used when it is defined in one line to describe a function. Triple double quotes or triple single quotes are used to define the docstring. When you use the help function to define a docstring, it helps you get information and returns with the docstring.

```
# pass function name to help() function
print(help(factorial))
```

The output of the help function is

```
Help on function factorial in module main:
factorial(x)
    This function returns the factorial of a given number.
None
```

You can also use the help function for multi-line docstring.

Getting the return value from a function

There are certain situations where a function returns with a value. To return value from a function, you need to use the return statement. The return statement returns with the value of the expression that follows the return keyword.

```
def fun():
    statement-1
    statement-2
    statement-3
    .
    .
    return [expression]
```

The return value is usually used to define the outcome of a function. The return statement is the end of a function where the outcome is apparent. However, it is not mandatory to use the return statement with a function. If the return sign is used without an expression, then it returns with none as a value. The return statement should be inside the function block.

```
def is_even(list1):
    even_num = []
    for n in list1:
        if n % 2 == 0:
            even_num.append(n)
    # return a list
    return even_num

# Pass list to the function
even_num = is_even([2, 3, 42, 51, 62, 70, 5, 9])
print("Even numbers are:", even_num)
```

The output will be [2,42, 62, 70]

You can also return multiple values with the help of the return statement. To use the return statement to return multiple values from a function, you need to use the return statement while expressing each expression by a comma.

In the function, you can also use the pass statement. Pass is a keyword that is not going to do anything significant, but it is often used in conjunction with functions. If there is an empty block and you need to define it, you can do so with the help of the pass statement.

When the pass statement is interpreted, it returns with no operation as an output.

When a function is called, the function is read by the controller, and the control goes to the function definition. After this, the entire code of the function is executed. All the codes of the function jump to the next statement after the function call.

Now that we know in depth about functions let us know further about the arguments used in the functions. The arguments used in functions are nothing but a value that is accepted by the function's condition. A function can have more than one argument. If you are creating a function with arguments, you have to create the range that passes the corresponding values of the function while calling them.

```
# function with two arguments
def add_numbers(num1, num2):
    sum = num1 + num2
    print('Sum: ',sum)

# function with no argument
def add_numbers():
    # code
```

```
# function call with two values
add_numbers(5, 4)

# function call with no value
add_numbers()
```

These arguments were parameters, the ones that the program has to pass through for the execution of the code in the function. The arguments are mentioned after the name of the function, and they are put inside the parenthesis. On the other hand, you can add as many arguments as you want, but each argument should be spaced out from the others with the help of a comma. Another rule that the Python coder needs to follow is that the function should have the correct number of arguments. If the function has two arguments, it should be called during execution using two arguments.

Let us know about the different types of arguments that can be paired up with the functions.

Default arguments

Default arguments in functions are the ones that are used when the function is called, but the values of parameters are not stated. In other words, if the right values are not used, then the default argument is used in its place to execute a function.

Arbitrary functions

Arbitrary arguments are different kind of argument that is used to pair an unknown number of arguments with a given function. When the number of arguments or parameters is not defined, then the function uses the arbitrary argument to take up an unlimited number of parameters. The coder adds an asterisk to add a tuple of arguments to the function.

Keyword arguments

There are also keyword arguments that can be used to define functions. If you wish to pass the arguments related to a function in a nonpositional manner, then you would need to use the keyword argument.

In the types of functions, you also get nested functions and recursive functions. A nested function is one that is defined inside another function. The nested function is also called the inner function. One of the main reasons why nested functions are used is to prevent the data from leaking out of its brackets to other parts of the function.

Recursive function

A recursive function is a type of function that repeats its behavior until a specific condition is met. When we use a recursive function, we tend to reduce the length of the code run by the function. With the reduction of code length, the readability of the code increases to a great extent. Therefore, when we think of solving complex problems with the help of Python, we prefer to use recursive functions. However, recursive functions take more memory space and more time for execution. The debugging process is not easy for the recursive function, so you spend a lot of time and effort in verifying the whole function.

Python anonymous function

Sometimes, we need to declare a function without any name. The nameless property function is termed the anonymous function. The reason behind using the anonymous function is instant use. That is, the Python anonymous function is used for one-time usage. The anonymous function is used with the help of the lambda keyword. In opposition to a normal function, a Python lambda function is a single expression. However, when we use the lambda keyword, the single expression can be extended toward multiple lines using parentheses.

```
lambda: argument_list:expression
```

When we define a function using the lambda keyword, the code is very concise. When the code is short and effective, there is better readability. The coder can use multiple arguments with the lambda function, but there can be only one value that returns after expression evaluation. We do not have to write extended return value statements in the lambda function because the function itself returns the expression value.

Lambda functions are more useful when we pass a function as an argument to another function. The lambda function can also be used with built-in functions like filter, map, etc. so that it can pair up with another function as an argument.

```
filter(function, sequence)
```

This is the type of inbuilt function that is used in conjugation with the lambda function. There is another function that can be used is the sequence function. The sequence function is used to code for any element like a string, tuple, etc.

```
l = [-10, 5, 12, -78, 6, -1, -7, 9]
positive_nos = list(filter(lambda x: x > 0, l))
print("Positive numbers are: ", positive_nos)
```

```
Positive numbers are: [5, 12, 6, 9]
```

Map() function

The map() function is a very important function in Python and is used to apply some kind of functionality to the different elements present inside a given sequence and then publish the list or the element with a modification.

```
list1 = [2, 3, 4, 8, 9]
list2 = list(map(lambda x: x*x*x, list1))
print("Cube values are:", list2)
```

```
Cube values are: [8, 27, 64, 512, 729]
```

Reduce() function

The reduce () function is a type of Python function that can be used to minimize sequence elements. The minimization is done in such a way that it can be consolidated into a single value by applying a specific condition. The reduce() function needs to be imported into the Python module with an imported statement before it is used.

```
reduce(function, sequence)
```

```
from functools import reduce
list1 = [20, 13, 4, 8, 9]
add = reduce(lambda x, y: x+y, list1)
print("Addition of all list elements is : ", add)>/code>
```

```
Addition of all list elements is : 54
```

Python library function

There is also a Python library function that you can use when coding with the use of different functions. The library function is a built-in function and can be used directly. There are certain commands that can be used through the Python library function. For example, you can use the `print()` command to print the strings that fall under the brackets.

There is also an `sqrt()` function that you can use to return the square root of a number. Then there is another function called the `pow()` function. This function helps in returning the power of a number. These functions are very useful, and they are found within the module. To use them, we need to import and use the module too.

```
import math

# sqrt computes the square root
square_root = math.sqrt(4)

print("Square Root of 4 is", square_root)

# pow() computes the power
power = pow(2, 3)

print("2 to the power 3 is", power)
```

Output

```
Square Root of 4 is 2.0
2 to the power 3 is 8
```

In the above example, the square root function is used. When the square root function is used, we import it from the `math` module. You need to include the `math` module in your program to

actually use `sqrt ()`.

Why do we use functions?

There are many benefits of using functions. Functions are used to a great extent in Python programming. Let us look at the most common benefits of using the functions defined and created within Python.

Reusability

The use of Python functions makes the code reusable. These days, coders and programmers are working on multiple projects at the same time. Creating programs with Python is a very demanding and competitive job role. To simplify and enhance the pace of development through Python, the code can be created with functions, and then these functions can be reused instead of repeating the same code for multiple projects. The code reusability of Python functions helps in increasing the productivity of coders and also meets the deadline for multiple large-scale projects right on time.

Readability

Python is a high-level language, and it can be difficult to understand for the general public. However, the defining and calling of a programmed function make the code simple and readable. For lines of code, one function can be defined, and that can replace lengthy codes that might not make any sense. This way, the code is broken down into small chunks of readable text, which is easier to read, interpret and manipulate.

Reduction of duplicity

When we are creating a program, we are anyway writing a lot of coded lines and text. However, suppose a task is repetitive in the code, and you have to code for it again and again. In such a situation, it is best to focus on reducing the length of sentences and also reduce the number of times a code is repeated. When a code is repeated again and again, there is high duplicity of code which can make the program difficult to use and redundant. So, when we define these codes with simple one-worded functions, the duplicity of coded repeated lines is reduced. Without the duplicity of the code lines, the code becomes more original and useful.

Bigger problems are broken down into smaller problems.

The aim of programming is to define and solve problems in an easy, computational method. However, if we use lengthy codes and repeat code lines, then we are complicating the process of problem-solving. To make an efficient machine backed by an efficient coded program, the language should be simple and easy to read. When it is simple and easy to read, the code written in Python can be broken down into smaller chunks, and bigger problems are also converted into smaller problems. When the code breaks down the bigger problems into smaller ones, they are easier to solve, and the output is also more accurate. Python is known for its ease of use, and this ease of use comes from the ability to break down code into smaller chunks and convert a big problem into small tasks that can be executed and solved easily.

Functions are an easy way of defining a task and then solving the task so that the code is simpler and usable. Python functions are a crucial part of the language and are used by programmers every day in the professional arena. Whether it is an in-built function or a user-defined function, there are many pieces of code that can be turned into a function and used to define a task and its needed values.

You can shorten and simplify Python code with the help of functions defined by the modules and customized by you.

Exercise

Write a program that uses a function to find the squares of all the numbers falling in the range of 0-11.

```
In [6]: #function definition
def square( y ):
    return y * y

for x in range(1,11):
    print (square(x))

1
4
9
16
25
36
49
64
81
100
```

Write code to multiply all the numbers present in a list.

```
In [12]: def multiply(numbers):
        total = 1
        for x in numbers:
            total *= x
        return total
print(multiply((3, 2, 4, 1, 5)))

120
```

Select the correct option.

A function is a code block that only executes when called and always returns a value.

A function only executes when it is called, and we can reuse it in a program.

Python doesn't support nested function

Select the correct option

A function can take an unlimited number of arguments.

A Python function can return multiple values.

Python function doesn't return anything unless and until you add a return statement.

Classes

Python is an object-oriented language, and objects are created to define data and tasks through this high-level language. Classes are associated with the creation and definition of objects through Python. Classes are a range of objects that are used to define and bundle data so that they can be functionally linked with each other.

Creating a new class means creating a new group of objects and creating certain instances of the type of objects. Each class instance can have attributes attached to it so that the classes can maintain their state. Class instances can also come with methods to modify their state.

When we compare Python's classes with other programming languages, classes are more well-defined and useful because they are defined using a level of syntax and semantics. The classes you find in Python are a mixture of the classes found in C++ and Modula-3. Python classes are specially designed for object-oriented programming and can be used to support the code that programmers develop with the help of Python.

The classes come with a class inheritance mechanism, a process through which there are multiple base classes. Moreover, a derived modified class can also override base classes if the conditions allow it to. Objects like classes hold arbitrary amounts of data and are dynamic, just like the language they are coded into. Classes are created during runtime and are further modified with time.

If we look at the theoretical definition of classes in Python, the classes are defined as a user-defined data type that can be used to keep data and methods that can be used to modify and manipulate the class itself. In a way, Python uses classes as templates to create objects in the object-oriented programming method. Classes define the characteristics and details of the operations that are the basis of a certain object that is created with the help of the language. To understand this better, take the class as a building.

There is a particular plan on which the building is based. The number of windows, doors, and other details of the building are known. Hence the building can be created based on its plan and details. In the same plan, there can be many buildings that can be created if we stay true to the details and methods. The same way, many objects can be created with the help of Python when their classes are defined.

Defining classes can be good for the programming project because classes help in reusable pieces of code that have encapsulated data and behavior together in a single entity. Just like functions, the use of classes in Python helps increase the reusability of code. With the use of coded classes, you can code for real life models and also solve real life problems in an organized way. If you want to categorise and solve complex problems with codes of Python, you need to create Python classes.

There are instances within the Python classes and each instance has some defining characteristics and properties. There are number of operations that are also linked with these class instances. When we do object oriented programming, we define different instances by their attributes and the attributes we need for dynamic programming. In terms of the language, attributes are variables inside a class that are defined in such a way that they can store necessary data and details that are necessary for the functioning of the class. The class also stores the methods by which the data can be used and the class can be operated.

Methods are functions that are defined within the class and are linked with the behavior of objects stored inside the class. These methods usually work with the attributes that define the instances stored in the class. The attributes of the objects and the methods used with them, all of these are the members of a class. Anything in the real life can be defined by a class in Python. Moreover, classes can be used to define class hierarchies so that code blocks can be reused and the code can be simplified into organized pieces of execution ready program.

Before we go towards creating and defining a class, let us first know the benefits of using classes in object oriented programming.

Benefits of using classes

Solve complex real life problems

There are many benefits of using classes defined by Python. One of the primary benefits is the use of code to solve complex problems. With the help of classes and the objects that are inside them, the coder can map out real-life object models. This method can be used to simplify complex problems and solve real-life problems. For better solutions through object-oriented programming, coders use classes defined by Python.

Reusability of code

To reduce the hassle related to coding, it is important to create reusable pieces of code. The reusability of coded components allows the coders to recycle pieces of code and use them in different projects. It also reduces the complexity of the code for a certain task. Apart from using functions, classes are also used to enhance the reusability of code. With the help of classes, class hierarchies are also defined, and that can help in organizing data and code.

You can also define related class hierarchies for better organization of code. The base classes lie at the top of the hierarchies and help define functionality. This common functionality is used to define the classes and their subclasses down the hierarchy. This reduces the code duplication related to the class and also promotes the reusability of code in a much simpler way.

Encapsulate large data sets in single entities

The use of classes is done to store and organize data sets. Classes are, therefore, very beneficial, and classes can be used to bundle attributes and methods together. This bundling and organization process helps you organize code blocks in a much better way. The organization of data and attributes with the help of classes increases the modularity of the code so that its internal architecture is supported.

These modular, independent entities can be used to numerous projects and lines of code.

Simplifies the implementation of concepts and objects

You can use classes to abstract away and organize the data so that the implementation of core concepts and details can be done very simply. The use of classes in the back end provides the scope of implementation of APIs so that the front-end users have a more interactive experience from the user interface.

Explore polymorphism

The use of Classes in Python can also help in unlocking the potential of polymorphism. The polymorphism of code is seen in the way a particular interface can be implemented across different types of classes. These classes can be used interchangeably with the code. This

simplicity and ease of use help in securing the polymorphism of code. The use of a common interface helps explore more flexibility with the code as it can be used in an adaptable manner.

From the above-listed benefits, we can conclude that the use of classes is a crucial part of coding and organizing programs. Classes in Python can be used to structure, organize and maintain code in a flexible, reusable, and user-friendly manner. While the use of classes is really beneficial, these elements should not always be used in Python. There are certain situations in which it is better to stay away from using classes. The classes can sometimes overcomplicate code if not used in the proper instance. Sometimes, it is better to use certain functions instead of class or class hierarchies. You should not use classes if the only goal is to store data and no other attribute. If the goal is to only store data, there are other objects that you can use. For example, you can store data using lists or named tuples.

Similarly, if the goal is to only store a method, then instead of choosing to use a class, you can use a function. The use of function can be done to store simple data and methods that do not need further organization with classes. If more methods arise with the progress of coding, then you can opt to create a class, define it and store data and methods in it.

On the other hand, you should also know when to create what type of class for data and functionalities. If there are certain in-built classes that you can use to store data and functionality, you should not create customized classes for the same. You need to know if there is an in-built or third-party class that you can use to store the same data and functionality easily. If it is possible with the use of in-built or third-party classes, you do not need to create user-defined classes.

If you have created a small and simple program that does not rely on complex data structures or logic, then you do not need to use classes to store data and functionality. The use of simple functions is enough for simple and small coded programs. Classes are mostly used to reduce the complexity of large data programs.

Moreover, if you are creating a performance-critical program where you need super high performance and no lags, then refrain from using too many classes. The use of classes also adds overhead to your program. In fact, if you add too many objects to a created class, the overhead increases to a great extent. Suppose the overhead increases far beyond the code base data; then the performance of the code is affected. Moreover, if you are using a legacy codebase and doing some modifications to it, using classes that are not aligned with the legacy codebase is not a good thing. If the legacy codebase does not use classes, then do not use them in an editing and modification cycle. If you unnecessarily use classes, it can disrupt the structure of the code. To keep the code consistent, you should avoid using classes when the codebase does not use classes. In the same way, if the code base does not align with the coding style that the team follows, then do not use classes that you can create for the code base. Consistency across the codebase is very important. If the code is being written in a functional approach and style, then putting all of it into classes is not the right decision. The use of classes will break the consistency of functional programming that is being done for the codebase.

How to create and use classes

Creation of a class

In the language of Python, a class can be created by using the keyword `class` in the code. The keyword `class` is followed by the name of the class to define it. Each class that is created and defined is also associated with a document string or docstring, just like a function is. The docstring describes the nature of the class. The docstring is defined and accessed by `<class-`

name>. `__doc__`. There are also statements associated with classes which are defined in the statement suite, and the classes are defined with the details inside them. The statement suite includes fields, constructors, etc., that can be used for the manipulation of the class and its data.

The syntax of the class is as follows.

```
class ClassName:  
    #statement_suite
```

`class_name` stands for the name of the class. The docstring describes the nature and function of the class. The docstring is not mandatory to associate with the class, but it should be associated to describe the purpose of the class in a better way. The statements or the statement suite holds the attributes and methods associated with the class and its objects.

```
class class_name:  
    '''This is a docstring. I have created a new class'''  
    <statement 1>  
    <statement 2>  
    .  
    .  
    <statement N>
```

The class can be created and defined with the use of certain details and attributes. For example, you need to use the class name and then a semicolon after the class keyword. The class body definition comes next, which can be done at the next indentation level. The class body ideally holds the methods and attributes related to the class. The class body works as a namespace where the attributes and methods of the class are stored. These class attributes and methods can only be accessed with the help of the class or its objects.

Self is typically used as the initial argument in methods in Python classes. To make it simple to utilize it inside the class, the self-argument keeps a reference to the current object.

Once the class is created, then you need to create objects that are inside the class.

Creation of objects inside the class

The class should not be empty; rather, it should be used to store new data in the form of objects and data sets. The process of creating a concrete object for an existing class is called the process of instantiation. In this process, with each cycle, you create a new object for the class. An object is a particular instance inside a class. You create objects from the class using the class constructor. The object's attributes are initialized in the constructor.

```
# Declare an object of a class
object_name = Class_Name(arguments)
```

```
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age
    def greet(self):
        print("Hello, my name is " + self.name)

# Create a new instance of the Person class and assign it to the variable person1
person1 = Person("Ayan", 25)
person1.greet()
```

```
"Hello, my name is Ayan"
```

An object is a very important thing to create in the class because, with its help of it, you can interact with the attributes associated with the class. The creation is done with the help of a special tool called the constructor. The class constructor is specific to the class type and is defined by it. There are two parts to the process of object creation. One way is the creation of objects and then the initialization of the objects.

When the object is created internally, the `__new__` command is used. On the other hand, the object is initialized with the help of the `__init__()`, which initializes the object into the class.

The syntax for objects inside the class.

```
<object-name> = <class-name>(<arguments>)
```

Now, let us learn about attributes present in the class. Attribute references use the standard syntax that is usually used for all other references in Python. Valid attribute names are all the names that were present in the namespace when the class and objects were created. The attributes of a class can be defined as types of variables that are associated with the class. When coders design a certain class, they also choose the variables that are associated with them. The types of variables linked with the class are either instance variables or class variables.

Instance variables are attributes that are attached to an instance or object in the class. The

instance variables are defined in the constructor method of the class. The class variable, on the other hand, is defined inside a class, but it is outside of any object or instance that might otherwise be inside the class. Instance variables are not associated and shared by different objects. Every object has its own copy of the instance variable. The class variables are, however, different and are shared by all the objects of a class.

In Python, you can create attributes with the help of ‘.’ Operator. The operator is used to define the attribute after an instance in the class. The definition can happen with whatever arbitrary name you want to give it and then follow it with its value.

```
class Example:
    def __init__(self):
        attr1 = 10 #Add an attribute upon initialization

example = Example()
example.attr2 = 12 #Add an attribute after initialization
```

In Python, you can access the attributes of a class with the help of dot notation and dot operator. The same can be done with the methods of the class too.

```
obj.attribute_name

obj.method_name()
```

Give me the following attribute or method from this object is stated in this using the dot notation. The second line accesses the method so that the programmer can call it, while the first line returns the value corresponding to the attribute. Like calling a function, calling a method requires a pair of parentheses. If necessary, you can also make a sequence of arguments.

Class methods

Now that we have talked about the classes and attributes stored inside the class let us talk about class methods. The class methods are usually of three types. The three types of methods are instance methods, class methods, and static methods.

The instance method is used to access and modify any object present inside the class. The instance method is so-called because it uses instance variables in its operations. The class method is different from the instance method because it is used to access or modify the class state in an overall manner. The class methods use class variables, and these variables are used to define and modify the entire class and not the objects present in the class.

In comparison to the other methods, the static method is different. The static method is used to move forward with the general utility method. The static method operates with the class as it works with the class in isolation to complete a task in isolation. In the operation of the static method, we do not use class or other variables present inside the class because the static method does not access the variables for the completion of the task. The different methods operate in different sections of the class. At the same time, class methods are operated with the class

variables, while the static method does not need any variable. On the other hand, the instance variables need the instance methods.

Pass statement in class

The use of a pass statement also happens with class. The pass statement is usually a null statement. So, the pass statement is executed, and this leads to no action or modification to the class. The pass statement leads to no operation. The pass statement is used as a placeholder in place of code when the coder is not sure what code statement to put in place of it. A passing statement is also used as a placeholder when we are planning to add code to the existing code at certain places in later releases.

Suppose we have a class, but there is no implementation yet; then we use the pass statement so that there is no empty body. This is done when we know that we will be using the same code in the future and will be making some modifications. You cannot have an empty body in the class because then the interpreter will read it as an error. To not generate an error message, you need to use the pass statement. In this way, we construct a class body that does nothing.

```
class Demo:  
    pass
```

Now that we know about the creation of objects and attributes, let us now know about object properties in the class. Object properties are an association of the name and the value of an object that defines the object. Object properties inside a class are nothing but instance variables. Every object has some properties associated with it. The coder can set and modify the object properties once the object has been initialized into the class. For this, the class property is directly called using the dot operator.

```
Obj.PROPERTY = value
```

You can even delete object properties to modify the object. The deleting process is done with the use of the del keyword. Once an object property is deleted, you cannot access it again. If the coder then tries to access it, he will get an error message shown on the screen. Any type of object inside a class can also be deleted with the help of the del keyword.

```
del object_name
```

Let us now talk about class definitions. The class definitions are almost similar to function definitions. These statements are defined by the use of the def keyword. These def statements must be executed before they have any effect. You can place a class definition in a branch of an if statement or inside a function. If we see the real definition of class definition statements, there is not much difference between the class definitions and class definitions.

When a class definition is entered into the Python workspace of the code, then there is a new namespace created. This is also used as the local scope. All the assignments that are done to the variables then go toward the new namespace. When the class definition is stated normally, a class object is created.

Naming conventions of class objects

Now that we know about the characteristics and operations aligned with class objects let us know about the naming conventions of class objects. These naming conventions are followed by the different members of the class. The naming conventions for members in a Python class are easy because the language itself is very flexible and easy to use. The Python class members are not distinguished on the basis of private, public, and protected attributes as are done in other languages. In Python, all of a class's attributes can be accessed using statements in one way or another. The naming conventions of Python classes are well-established and should be followed properly.

The first rule of the naming convention that we should follow is the use of an underscore below the name of the class member so that the name is highlighted. In the public classification, the name is written with normal naming conventions. When the class member is classified as a nonpublic member, then the name is classified with a leading underscore. Public members of the class are officially included in the API of your classes. This means that you should not use nonpublic members outside of their defining class.

When we use the leading underscore before the member name, that shows that we should not be using the member attribute directly from outside the defining class; the difference between nonpublic and public members of the class is that nonpublic members exist to support the public members of the class that are present internally. The nonpublic members can be removed at any moment, and the class should not fully rely on them for its operations. Depending on how the class is ultimately implemented, these members may or may not exist. You shouldn't utilize them directly in the client-side code in such circumstances. If you do, it can have a significant impact on the code's consistency.

When we are writing classes with the help of Python code, it is sometimes difficult to decide if the declared members are public or nonpublic. The decision to classify and name the members as public or nonpublic depends on the way the users would use the classes. Mostly, the attributes should be made nonpublic to ensure that the classes are safely used. You can also start with coding for the members in a nonpublic manner, and then as the real use cases appear, you can go towards making these members public in a gradual manner.

Name mangling

There is another naming convention related to the classes defined in Python. It is the adding of two leading underscores to attribute and method names, as per the naming rules. This is a step in the name mangling process. Name mangling is a process by which the member's name transforms automatically, and then the class name is attached to the attribute or the method name. This whole process results in name hiding. By name hiding, we mean that mangled names are not directly accessed. Direct access is stopped for the mangled names because they are not available in the class's API. This happens in the form of `_classname_attribute`. The same thing happens for the method too. There is the internal renaming of the mangled names, which is why they are not available for direct access.

The internal renaming also creates an illusion of a private attribute, and you can only access them through their mangled names. However, accessing attributes through their mangled names is not something that should be done that often. Name mangling is a good practice because it ensures that the attribute that you have specified does not get overwritten in code. In a situation where you are defining a lot of classes and subclasses, you can simply avoid naming conflicts by using

the name mangling convention of Python. The name mangling process also protects the methods of the class from being overridden by subclasses so that you can conserve their performance.

In the previous sections, we have talked about classes, objects, and attributes. Let us now know how to add and access attributes as well as modify them according to the requirements of the code. The data that is stored in the class comes as attributes, and the behavior comes as methods.

When modifying the class attributes, you cannot do so with the help of instances of the class. As the class attributes are associated with the entire class, any modification in the class attribute shows up on all the instances present inside the class. The class attributes are used to add data and share data between the different instances inside the class. On the other hand, the instance attribute is linked with a certain instance within the class. Therefore, if you are trying to access the instance attribute, you cannot do so with the help of the class object. If the same is tried, then you will get a message of `AttributeError`.

There is another attribute related to classes that should be discussed. This special attribute is the `__dict__` attribute. The dict attribute is related to the class and the instances inside it. Each key in the dict attribute denotes an attribute name. The value associated with a single key is also associated with an instance attribute or method. The dict attribute is short for the dictionary attribute, which is helpful in denoting the writable members of the class or instances. When associated with the class, the dict attribute will contain the class attributes and methods.

When the same attribute is applied to an instance or object, then it denotes the instance attributes or methods. The language looks for the class object in the class dict attribute when you access a class member through the class object. The same thing happens for the instance attribute if you search for an instance member through the instance object. If the name does not appear in the instance object, then the name is searched for in the outer class. If there is no reference to the name, the dict attribute returns a `NameError` message.

```
# sample_dict.py

class SampleClass:
    class_attr = 100

    def __init__(self, instance_attr):
        self.instance_attr = instance_attr

    def method(self):
        print(f"Class attribute: {self.class_attr}")
        print(f"Instance attribute: {self.instance_attr}")
```

You specify a class attribute with a value of 100 in the code above. You can create an instance attribute that accepts user input by using the `__init__()` function. Then the code defines the method to print both attributes.

```

>>> from sample_dict import SampleClass

>>> SampleClass.class_attr
100

>>> SampleClass.__dict__
mappingproxy({
    '__module__': '__main__',
    'class_attr': 100,
    '__init__': <function SampleClass.__init__ at 0x1036c62a0>,
    'method': <function SampleClass.method at 0x1036c56c0>,
    '__dict__': <attribute '__dict__' of 'SampleClass' objects>,
    '__weakref__': <attribute '__weakref__' of 'SampleClass' objects>,
    '__doc__': None
})

>>> SampleClass.__dict__["class_attr"]

```

The class `_dict_` dictionary contains both the methods and the class attribute. The dict attribute allows the programmer to look up and access the value of the class attributes. This can be done by specifying the attribute's name in square brackets. You can achieve this by putting the name of the attribute in square brackets.

In the case of instance attributes, you can also modify the dictionary attribute. When we modify the dictionary attribute, the instance attributes linked with the dictionary attribute also go through changes. With the dictionary attribute, the programmer can even add new attributes to an instance. You can prevent RecursionError occurrences when the dictionary attribute is run and descriptors are written if you use it to modify the value of the instance attributes.

Property and descriptor attribute

In addition to a class's already-existing instance attributes, the Python language also lets you add function-like behavior to a class. This action turns the existing instance attributes into managed attributes. Introducing these types of attributes in the class allows the coder to maintain the consistency of the code and does not break the API coded with the class. Managed attributes are one step ahead of the general attributes of a class because they have dual nature. You get to see function-like behavior and attribute-like access in the same managed entity. The use of a well-managed attribute can maintain the concurrency of the APIs and the integrity of the code.

To create a managed attribute for the class, you need to use a descriptor or a property. This depends on the class and code you are creating or the requirements of the project.

The `@property` decorator must be used while writing the code method to convert an attribute to a property. Idealistically, the method should return the attribute's value. The addition of descriptors is also a good way of ensuring the attribute changes into a managed attribute. The use of a descriptor can reduce repetition in the code and streamline the class for better output and

performance. The code with a descriptor is easily maintained and reusable.

Examples of methods

We have already known that there are different kinds of methods associated with an ideal class. There could be instance methods, class methods, or static methods. Let us delve into instance methods with self.

Instance methods with self

There are instance methods in classes that accept the current instance as their first argument. In Python, the first argument is known as self.

The self-argument relates to the current instance, and it enables you to use instance methods to access the same instance. You can access and change a class' instance attributes with the self-argument, as well as call other methods.

Create a normal function with the requirement that the first argument be self. You can access the attribute on the current instance using the dot notation inside the methods by using the self-argument. After that, you can also modify the attribute's present value. It is not necessary to specify a value for the self-argument when using the dot notation to invoke an instance method on a concrete attribute. Python handles the initial step of the procedure automatically using the self-argument. The other arguments related to the attribute must be provided afterwards.

While this action happens automatically, you can also provide an argument to the instance manually. You explicitly call instance methods on the class during this process. You must give the self-argument a suitable and accurate value in order for this strategy to succeed. If you fail to provide a suitable instance, then you also fail to call the instance, and the program shows TypeError. The error message shows that there is a missing positional argument.

Special methods

Since we are talking about methods related to class, let us also talk about the special methods that can be used with the class. Python also has what are called special methods. Dunder or magic methods are other names for the special methods that Python supports. These specialized methods are typically included in instance methods of class mechanisms and are a part of how Python classes operate internally. Python calls these methods in response to specific operations, and this calling happens automatically. Let us know further about the use and types of special methods of Python.

Special methods are easily recognized from the way they are denoted in the program. For example, you can spot a special method by the way it is written. The names of the special methods start and end with a double underscore. The `__init__()` method is the most common method used with Python classes. The `__init__()` works as the initializer of the code. When the coder uses the class constructor to initialize a class object, Python automatically calls on it.

There are other special methods in Python classes too. For example, you have the string method or the `__str__()`. This particular method is used to represent the object under informal string representation. This method gives back a user-friendly string representation of the object. The `__str__()` and `print()` APIs allow you to access an object's informal string representation.

The `__repr__` method is also similar to the other special methods. However, the repr method is opposite to the string method because it stands for the formal string representation of the object. The `__repr__()` method is used to represent the specific object with a string that will help in its

recreation. This special method is mostly used by Python programmers to create the same object again and again.

When attempting to retrieve an object or evaluate an expression, the Python programming language invokes the `__repr__()` function. The formal string representation of the underlying object is then returned by the procedure. When the programmer is using the script mode, he can use the `__repr__()` method to access the formal string representation of the object.

The above functions are some of the special methods used for Python classes.

Now let us focus on the protocols that Python classes follow. There are many protocols present in Python that are usually used to support a given object or functionality.

These protocols are more than one protocols that are put together to support the modification of class objects.

An iterator is a type of protocol that allows the coder to create an iterator object. The iterable protocol makes your class objects iterable. The descriptor protocol allows the programmer to write managed attributes. Similarly, the context manager protocol helps in writing a code that enables an object to work on with statements.

Apart from these most common protocols of the Python class, there are other protocols that are used to make the Python language feature-rich and easy to manipulate.

Example of protocol usage

```
>>> class ThreeDPoint:
...     def __init__(self, x, y, z):
...         self.x = x
...         self.y = y
...         self.z = z
...     def __iter__(self):
...         yield from (self.x, self.y, self.z)
...

>>> list(ThreeDPoint(4, 8, 16))
[4, 8, 16]
```

This class represents the three coordinates of a space point and uses the self-argument, and for that, it also uses three arguments separately. Here the code uses the iterator method.

Specialized classes

There are some specialized classes in Python that are used for certain reasons in specific conditions. In the Python standard library, you will find these specialized classes. If you want to simplify the code you write for a class, you can use the data class from the standard library. The data classes are found in the data classes module.

Data classes

Python's data classes specialize in storing data, but they are also a bit different from other classes because they act as code generators. So, when you need some boilerplate code for a Python class,

you can simply use the data classes, and they will generate a lot of boilerplate code for you. You do not need to implement specific methods if you create a custom class using the data class infrastructure. The data class can make code writing easy for the coder by writing special methods automatically so that you do not have to write the code manually.

The automatic writing of the codes is done according to the best practices of the Python language in a manner that the code is error-free. With the use of special methods like data classes, you get accurate string representation and comparison capabilities. Moreover, you will also have hash ability and more. Data classes may seem like regular classes, but they have more abilities and are able to offer more methods and data storage.

How to create a data class

You can create a data class by importing the `@dataclass` decorator from the data classes. This imported decorator is used to define the data class. The usual initialization method is not used in this process; rather, you use the data fields to define the data class with class attributes by using type hints.

Example

```
# point.py

from dataclasses import dataclass

@dataclass
class ThreeDPoint:
    x: int | float
    y: int | float
    z: int | float

    @classmethod
    def from_sequence(cls, sequence):
        return cls(*sequence)

    @staticmethod
    def show_intro_message(name):
        print(f"Hey {name}! This is your 3D Point!")
```

The definition of attributes in the data classes is very easy because data classes are very flexible. The coder can easily define data fields for the data class without any major issues. Fields can be declared using the type annotation syntax. The data fields and attributes can be initialized with a suitable and sensible default value. You can also combine the type annotation syntax and the non-type annotation syntax when defining the data fields of the data classes.

For example, you can see the following code.

```
from dataclasses import dataclass

@dataclass
class ThreeDPoint:
    x: int | float
    y = 0.0
    z: int | float = 0.0
```

In this code block, there is the usage of type annotation syntax with the first attribute. Then, if you look at the second attribute, there is a default value, and there is no type annotation syntax. The third attribute has a different representation, and this representation comes with type annotation syntax and default value. Specifying the type hint in this process is very important. Python will not automatically generate the matching for that attribute if the type hints are not specified. After defining the data fields, you can begin including the methods.

Enumeration

Like data classes, there are other special actions that you can take with classes in Python. The enumeration process is one such process. Enumerations are a special kind of data type and is also termed an enum. Enum allows you to create a class of named constants which are named as members and can be accessed through the enum class itself. The enum module is not present in the old Python versions. If you look at the different versions of Python, Python 3.4 has a built-in enumeration module, and the same is not seen in earlier versions. The enum module also has the enum class, which is used for general-purpose enumeration.

In our daily lives, both professional and personal, we deal with a lot of numbers. We have so many things, people, and creatures around us that numbers are used to enumerate them. To deal with these numbers, the constant number group is formed by the enum class. Since the numbers can be unlimited and large, the enum class is used to operate with it. The enum classes can be used to specify variables that can take one of a set of limited values of numerical nature.

The enum class, therefore, is used to create and custom enumerations and groups of constant numbers for different purposes. The process is done very efficiently and easily. The enum class has many special features that let you operate with these numbers without the possibility of manual interference. Dealing with constant numbers manually can be a little difficult. The enum class simplifies and automates this process. To create customized enumerations with the enum class, you need to subclass the enum class.

For example,

```

>>> from enum import Enum

>>> class WeekDay(Enum):
...     MONDAY = 1
...     TUESDAY = 2
...     WEDNESDAY = 3
...     THURSDAY = 4
...     FRIDAY = 5
...     SATURDAY = 6
...     SUNDAY = 7
...

```

In the above example, the enumeration happens by defining the weekday element and subclassing the enum class from the enum module. The constants that are listed are called enum members. Since the members of the enum class are strict numbers, you cannot change the value of these numbers, and these are immutable. These numbers are also iterable by default, so you can use and interpret them accordingly.

```

>>> WeekDay.MONDAY = 0
Traceback (most recent call last):
...
AttributeError: cannot reassign member 'MONDAY'

>>> list(WeekDay)
[
    <WeekDay.MONDAY: 1>,
    <WeekDay.TUESDAY: 2>,
    <WeekDay.WEDNESDAY: 3>,
    <WeekDay.THURSDAY: 4>,
    <WeekDay.FRIDAY: 5>,
    <WeekDay.SATURDAY: 6>,
    <WeekDay.SUNDAY: 7>
]

```

Moreover, if the coder tries to change the value of an enum member, then the command returns back with an `AttributeError` message on the screen. However, the iteration that happens happens directly.

Here is an example of how to access the enum members

```
>>> WeekDay.THURSDAY.name
'THURSDAY'
>>> WeekDay.THURSDAY.value
4

>>> for day in WeekDay:
...     print(day.name, "->", day.value)
...
MONDAY -> 1
TUESDAY -> 2
WEDNESDAY -> 3
THURSDAY -> 4
FRIDAY -> 5
SATURDAY -> 6
SUNDAY -> 7
```

In the above example, you are accessing the `.name` and `.value` attributes of the specific enum members. These attributes can be used to access the components of each member. Moreover, in the enum class, you can also add customized behavior to the enum members to modify their usage. This can be done in the same methods as any other regular classes.

Debugging the Python classes

In the last section of this chapter, let us talk about debugging the Python classes. Debugging often refers to the act of tracking errors and issues in the Python code and rectifying them. There have to be good tools for the debugging process; otherwise, the coder will spend a lot of time and effort in debugging. Let us discuss the general challenges that you will face when debugging the code written through Python classes.

For example, if you try to access an attribute or method in a particular class and that specific value is nonexistent, then you will get the message of `AttributeError`.

```

>>> class Point:
...     def __init__(self, x, y):
...         self.x = x
...         self.y = y
...

>>> point = Point(4, 8)
>>> point.z
Traceback (most recent call last):
...
AttributeError: 'Point' object has no attribute 'z'

```

There are many types of errors that happen in the code, and these error messages show up on the screen. The attribute error arises when the specified object fails to define the attribute or the method that you are stating. If the access point does not coincide with the query that you are sending with the class, then it returns with the attribute error statement.

You might also get a `TypeError` from the class operations. A `TypeError` statement is shown on the screen when you apply an operation or function to an attribute that is not aligned with that particular operation. The object or attribute for which the function is being called should be supportive and accepting of the function. Otherwise, the language will see it as a typing error. Therefore, you need to see which values you are specifying as arguments and for which function.

There is also a possibility that you will come across a `NotImplementedError` statement during the debugging cycle. When you have an abstract method, and it is not implemented in a concrete subclass, the final result will be such an error statement.

Other than these, there are some simple mistakes that Python beginners make when writing code.

For example, the coder might forget to include the self-argument in the case of instance methods. An error can also arise when the coder is unable to instantiate the object instance when he fails to call the class constructor, and there is a lack of suitable arguments. Errors also arise when you are confused between the types of attributes present inside a class. If the coder is confused between the class attributes and instance attributes, then he will be leaving errors in the code.

In the chapter, we have discussed the naming conventions related to the Python code. Python is very particular about the names that are given to objects, classes, and attributes. If the specified naming conventions are not followed for the class members, then there will be a major error in the code execution.

There are other situations where there can be common mistakes in coding. For example, there are strict differences between the public and nonpublic members of a class. When you try to access the nonpublic members of the class from outside the class, you are unable to do so because of access restrictions, and it shows an error message. Moreover, if you overuse and misuse class inheritance, then it can also lead to errors in the code.

Classes are the building blocks of object-oriented programming that happens through Python. Therefore, learning how to use Python classes and their operations is very necessary. For smaller

and simpler codes, functions are enough. However, for large data sets, you need special classes and objects. As you learn and grow your Python skills, you will face challenges in solving complex programming problems. For this purpose, clear and useful knowledge about the classes and objects in Python is very necessary.

Exercise

- What is a class in Python? What are the benefits of classes in Python?
- What are the different types of arguments used in classes?
- What are the common errors that we face when initially coding in Python classes?
- What are class methods, and what are the types of class methods in Python?

Write a program for the creation of a class and also show the namespace for the same.

Solution

```
class py_solution:
    def sub_sets(self, sset):
        return self.subsetsRecur([], sorted(sset))
    def subsetsRecur(self, current, sset):
        if sset:
            return self.subsetsRecur(current, sset[1:]) + self.subsetsRecur(current + [sset[0]],
                                     sset[1:])
        return [current]
for name in py_solution.__dict__:
    print(name)
```

FILES AND EXCEPTIONS

When we operate on a computer, we do it through files and saved objects. When we are using python to code and execute codes, the usage of files will make it very fast and robust. To support the code written through the Python language, you need to use files. Files are specified locations on a disk where information is stored. So, files are used to store associated data in the Python code. Let us learn about file operations related to Python and the go towards Exceptions found in Python code.

Reading

The basic operation that is done with files through Python code is reading files. Often, data is stored inside text files. So, when we are working with such files, we should first read them into the memory of the computer. To read the file, the most basic prerequisite is that the data file exists in the device or any connected device.

The name of the file is passed through the open() function as its only argument. When opening the file, Python looks for the years.txt file in the folder where the file is located. The open() function returns an object that represents the file, and then it is stored in the variable file_object. After opening the file, the file is also closed when it is no longer needed. The file is closed and shut by the use of the keyword with. The read() method is a function used to read all the data that is kept inside the text file so that the code can run with the data. The whole data is read, and then it is stored in the contents.

```
2022  
2021  
2020
```

```
with open('years.txt') as file_object:  
    contents = file_object.read()  
    print(contents)
```

```
2022  
2021  
2020
```

Now that we know how to open and close a file, let us know how to use the contents of the file and work with it when we are using the Python code. Before you start to use the text from a file and you read the text file to work with it, know that any text read by the Python code is treated as a string by Python. If a number is read from the file, then you need to convert the number into float using the float() function and then carry out arithmetic operations with it. You can also convert the number into an integer with the int() function.

Writing into a file

If you are going to write data into a file, you need to use special functions that are meant to commit operations like writing. The output will be even after we close the terminal that holds the program's output. When writing text into a file, you need to use the same function open() with two arguments so that the output will be written. The first argument is the filename so that the right file is open for writing. The second argument that is paired with the command is the mode in which you wish to open and close the file when the writing is done.

There are four modes in which you can open the file. These four modes are read mode, write mode, append mode and read and write mode.

Writing to an empty file

If the file you are writing in is nonexistent or empty, you need to use the open() function, which will auto-generate the file.

```
with open('student.txt', 'w') as file_object:  
    file_object.write("My name is Felix.")
```

```
My name is Felix.
```

Appending a file

You can also append a file with the help of Python commands. When you wish to add data to the file, you have to use the append function. Any data you will add will be added to the end of the file.

```
with open('student.txt', 'a') as file_object: #'a' argument to open the file
    file_object.write("I am 6 years old\n")
    file_object.write("I love playing games\n")
```

```
My name is Felix.
I am 6 years old
I love playing games
```

These are the basic operations that happen with the file when used through the Python functions.

Exercise

- Write a program to read a text file in Python.

```
1 def file_read(fname):
2     txt = open(fname)
3     print(txt.read())
4
5 file_read('test.txt')
```

- Write a program that can append text to a file.

```
def file_read(fname):
    from itertools import islice
    with open(fname, "w") as myfile:
        myfile.write("Python Exercises\n")
        myfile.write("Java Exercises")
    txt = open(fname)
    print(txt.read())
file_read('abc.txt')
```

Exception - how to use it

There are many characteristics of Python code, and one of them is exceptions. The nature of exceptions is a type of error that arises in the code when it is not written properly. Let us talk about exceptions and the handling of exceptions. There are two types of errors in Python code. One type of error is the syntax error that you might come across when the usual syntax of the Python commands is not correct. Exceptions arise when some internal events happen in the code, and then there are some changes that lead to error. Exceptions are responsible for changing the normal flow of the code and its execution.

There are many types of exceptions that are built into the Python language. The common type of exceptions are as follows-

SyntaxError- the SyntaxError happens when the interpreter runs across a code with the wrong syntax arrangement. This can be a misplaced colon, a misplaced bracket, or the wrong word at the wrong place.

TypeError- the TypeError exception happens when you use an operation with the wrong type of object, so there is no concurrency of object and command which shows the TypeError. For example, if you are adding a string to an integer, it will show that there is a TypeError.

NameError- there is also another type of error that comes under the exception. When a variable or a function name is not found in the current scope, then it gives rise to a NameError statement.

IndexError- an IndexError statement comes up as an exception. This happens when an index is out of the range for a list, tuple, or other sequence types.

KeyError- the KeyError statement rises when the code is unable to find a specific key in a dictionary.

ValueError- The ValueError is an exception that is raised when a function or method is invalid because of an invalid argument. When the argument is invalid, the resulting value of the function after execution will be an error.

AttributeError- AttributeError is an exception, and it arises when an attribute is not found in an object. Since you are trying to access an attribute that is not present, it shows an error when the access function is executed.

IOError- an IOError arises when the input and output functions are not properly executed, and the operation that should give a definite result fails.

ZeroDivisionError- this type of error arises when there is an arithmetic operation, and the result comes out not defined because a number is divided by zero. Any number divided by zero does not give a proper result, and so you get ZeroDivisionError.

Import error- the ImportError message comes up when an import statement has been launched, and it fails to find the right module or load the right module. When it fails to import the object that has been stated, the error is of this type.

Let us now discuss built-in exceptions of the Python language. In the Python language, all the exceptions should be derivations of a class that derives from BaseException. When you have a try statement with an except clause, it mentions a particular class, and that except clause handles any kind of exception classes that arise from that class. Two exception classes that are not related by their subclasses are not equivalent to each other.

The use of user code can lead to the rising of built-in exceptions. This can be used to check an exception handler or to see the use of the case to report an error condition. This error is often recognized by the interpreter. However, if the same is raised by the user code, there is a possibility that the error is inappropriate. The built-in exception classes can be used to subclass so that new exceptions are identified. Programmers are often encouraged to find new exceptions from the exception class or one of its subclasses. The exception class is derived from the BaseException class.

Try and except the statement

The try and except statement is used to catch exceptions. For handling exceptions, you need to first try to catch the exceptions. Statements that could be raising the exceptions are kept inside the try clause, and then the statements that handle the exception are put inside the except clause.

```
# Python program to handle simple runtime error
#Python 3

a = [1, 2, 3]
try:
    print ("Second element = %d" %(a[1]))

    # Throws error since there are only 3 elements in array
    print ("Fourth element = %d" %(a[3]))

except:
    print ("An error occurred")
```

According to the above statement, the statements that can cause the errors are placed within the try statement. The exception that arises is then handled by the except statement. It is also possible to catch specific exceptions that might arise when you are going to execute a code. To catch a specific exception, the try statement needs to have more than one except clause. These many clauses are used to catch different exceptions and handling them in a different manner. In such cases, at most, one handler will be executed. For example, if you add `IndexError` to the try statement, then the syntax will be

```
try:
    # statement(s)
except IndexError:
    # statement(s)
except ValueError:
    # statement(s)
```

Then the output will show what type of exception was raised and what was the error type. After this, you can also use the try else clause. The try else clause uses the else clause with the try statement. The else statement is used in conjugation with the try-except statement. The code then executes with the else statement if the try-except clause is unable to find an exception.

For example,

```

# Program to depict else clause with try-exce
# Python 3
# Function which returns a/b
def AbyB(a , b):
    try:
        c = ((a+b) / (a-b))
    except ZeroDivisionError:
        print ("a/b result in 0")
    else:
        print (c)

# Driver program to test above function
AbyB(2.0, 3.0)
AbyB(3.0, 3.0)

```

There is also usage of finally keyword in the exception statements. The finally keyword is executed only when the try and except statements are already executed. The finally keyword is always executed after the execution and termination of the try statement, either due to normal conditions or under some sudden reason of exception.

Raising exception

The raise statement is a very different statement from the try and except statement. The raise statement is used to force a specific exception to happen. The argument that is linked with the raise statement is used to indicate the exception that needs to be raised. The exception to be raised can be an instance or a class.

These are the many ways in which exceptions are handled in Python code.

Benefits of exception handling

There are many benefits of exception handling. The most common benefits of exception handling are as follows-

Improved program reliability- when we handle the exceptions really well and are able to catch them, we also ensure that the program is more stable and reliable. When the program is launched after exception handling, the program runs in an error-free manner, and the performance is high and stable.

Simple error handling- When we do good exception handling, the error handling is also very simple and fast. The error handling code is thus separate from the main programmed code, which makes the error handling process very simple and clean. The main program logic is easily read and maintained without major issues.

Readable, cleaner code- due to exception handling, it is also easier to maintain and read the code, as we have discussed in the earlier point. The code does not have highly complex conditional statements, and this helps simplify the code. The code is a lot cleaner and easy to execute. The coder is able to revisit the code and make sense of it even without an interpreter.

Easier debugging- debugging of the Python code is very necessary. The debugging process is

supported by the raising of an exception. When an exception is raised successfully, there is a traceback to the original source of this exception, so you know where the error has happened and how it can be debugged. The debugging process is done along with the exception-handling process, and you get clean and accurate Python code.

Disadvantages of Exception handling

There are some disadvantages of exception handling too. These disadvantages should also be paid attention to, along with the usefulness and benefits of exception handling.

Performance overhead- exception handling is a process that could add considerable overhead to code and performance in the final phase. The performance overhead comes from exception-handling statements that are slower in runtime than conditional statements. The coder or tester has to first find the error and then also handle the exception. This process is a lengthy one that reduces the productivity of the team and also adds performance overhead to the code.

Increase of complex code- when we are doing exception handling, we are not only checking the code but also writing more new code to catch the exception and resolve the errors. The process is, therefore, a lengthy one that adds more code and increases the complexity of the initial code. If there are fewer types of error and there is only one type that you have to resolve, then the final output is less complex. However, when you have to handle multiple exceptions and also implement complex code logic to resolve the errors, you need to do it with the help of complex code. Therefore, when you are checking and handling the exceptions, make sure you are not overcomplicating the code.

Security risks

There is also a risk of developing security lags and risks during the exception-handling process. Improperly handled risks can often lead to more vulnerabilities in the code, which further jeopardizes the security of the Python code block. Of course, proper exception handling can make the code more robust and easy to use, but when the process is not completed properly, there could be higher risks of the code failing and getting corrupted.

Keeping these advantages and disadvantages in mind, you need to plan and use exception-handling statements. One must know how to handle the errors properly; otherwise, it can lead to more issues and other problems in the code.

Exercise

- What are exceptions and the types of exceptions found in Python?
- What are the benefits of exception handling for Python?
- What is exception handling?

TESTING YOUR CODE

Testing of the code created by the Python language is very important. You need to check for any errors and issues in the Python code before the code is deployed. Today, Python-coded apps and software are used in different applications and industrial purposes. Such high-demand applications need accurate and robust code. The accuracy and high performance of the code are only ensured with extensive tests run on the written code. Many developers often go towards the dual role of tester or take up the testing and quality assurance of Python code as a full-time job role. Earlier, testing was more manual work and used to take a lot of time. Testing of code is about running the code under a number of test situations and simulations so that errors and performance lags are clear before the code is actually deployed. The errors and performance

issues that come up need to be minimized before the code is actually released on the market.

Sometimes, there could be a number of revisions and modifications to the code suggested in the testing phase that improves the code before it is packaged into the back end of an application and launched in the market.

There are many types of tests that you need to run on the Python code before they are launched as an application. The most common type of tests that are run on Python code are unit tests and integration tests.

Unit tests

Unit testing is done when a small unit of code is tested instead of the entire body of code. The unit test often happens for one function or one if statement. The granularity of the tests that are done on the code depends on the complexity of the code and the errors that you expect to be prevalent.

Your unit tests are aimed at testing one unit of the code, and then you can see the performance of that unit as a projection of performance from the whole code framework. For the unit test, you need to run one function and observe the code making one action and how well that coded action happens for the result. This test can be run manually, but it can also be run automatically with the use of testing tools. These days, the testers are using many advanced tools for unit testing. There are Python testing suites that are useful and produce accurate results at a faster pace. Unit tests are preferred by developers and developing companies because the test is very cheap.

Integration tests

After unit testing, there are also integration tests that can run on the existing code. After conducting unit tests, you also need to test how the single units integrate to create integrated results of the whole code. While unit tests are done to check the integrity of code at the lowest level, integration tests are done to look at the bigger picture as a whole. The integration tests are designed in such a way that the tester knows how the single units integrate and interact with each other to gather a collective result. The accuracy of the code as a whole depends on the great length of the integration tests. The issues at the macro level are resolved with the help of integration tests. There are specific tools that are used for integration testing. However, integration tests are also dependent on other resources like databases. Integration tests are slower to run because there are so many dependencies. However, they are written in such a way that they can run in parallel. There are also other types of tests, like contract tests and functional tests, that are run on Python code. All these tests ensure that the code is safe and ready to be launched in the market. If you are planning to write tests for the Python code manually, make sure that you have written testable code.

Testing is very important today because there is continuous and high demand for high-quality and high-performance apps from different industries. To ensure that the applications are high-performance, you need to run a number of tests on the code. The process of writing code in Python is only successful when it goes through different cycles of tests very easily. After testing, the application code is ready to be deployed and used.

MAKE THE FIRST PROJECT

To do the first project, with the help of Python, you can create a random password generator. Below are the requirements and process of making a password generator with Python.

For this challenge, we will use a Python script to generate a random password of 8 characters. Each time the program is run, a new password will be generated randomly. The passwords generated will be 8 characters long and will have to include the following characters in any order:

- 2 uppercase letters from A to Z,
- 2 lowercase letters from A to Z,
- 2 digits from 0 to 9,
- 2 punctuation signs such as !, ?, “, # etc.

To solve this challenge, we will have to generate random characters, and to do so; we will need to use the ASCII code.

ASCII Code

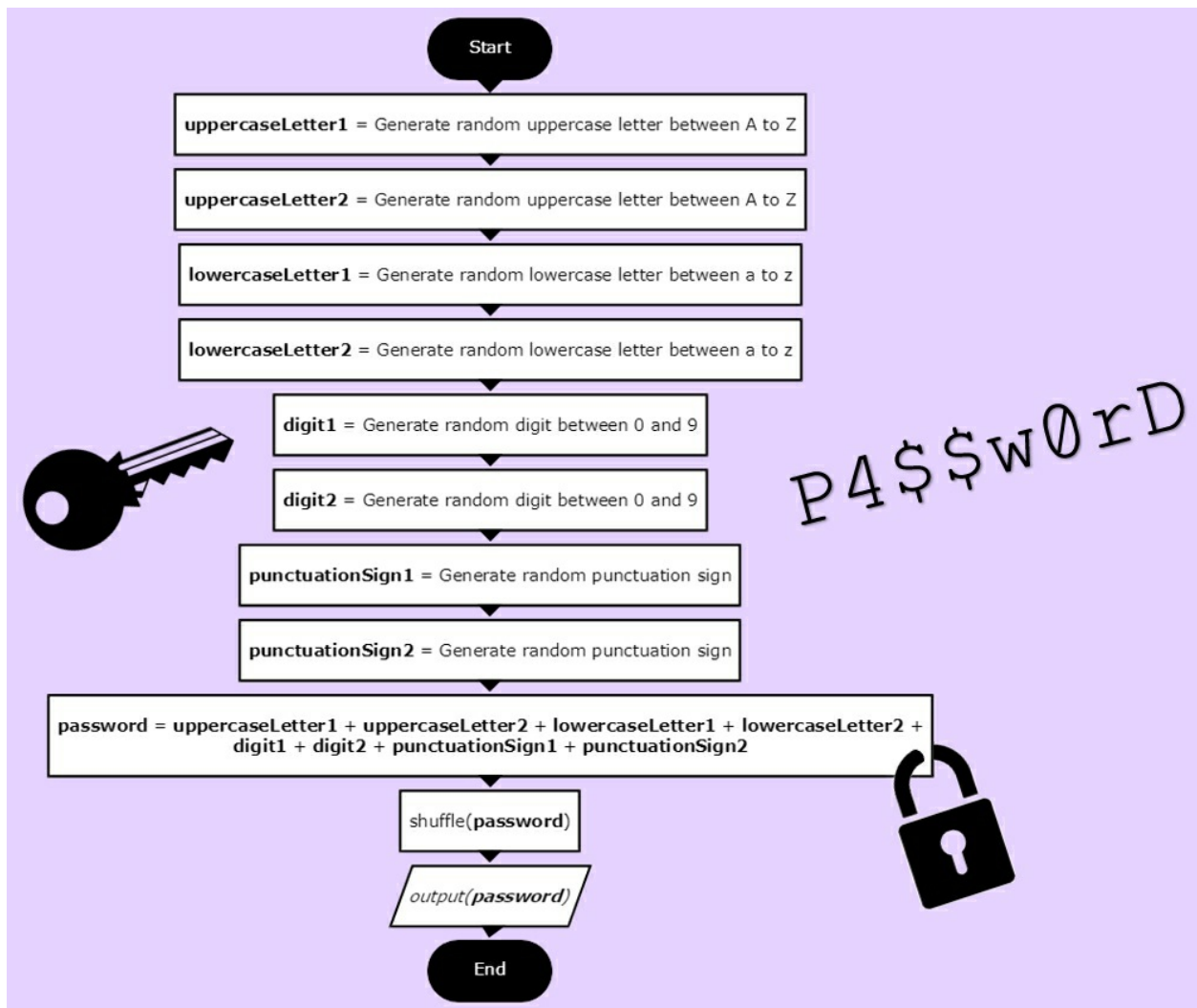
The **ASCII code** is a code for representing English characters as numbers, with each character assigned a number from 0 to 127. For example, the ASCII code for uppercase M is 77. The **extended ASCII code** contains 256 characters (using numbers from 0 to 255).

Using Python, you can easily access the ASCII values of a character using the *ord()* function. For instance *ord("M")* returns 77 and *chr(77)* returns "M"

When looking at the list of the most widely used ASCII codes, you will notice that all uppercase letters from A to Z have an ASCII code between 65 (=A) and 90 (=Z). To generate a random uppercase letter between A and Z, we can hence use the following Python code:

- `import random`
- `uppercaseLetter=chr(random.randint(65,90))` #Generate a random Uppercase letter (based on ASCII code)

Flowchart



REFERENCES

<https://learnpython.com/blog/python-list-loop/>

<https://www.geeksforgeeks.org/iterate-over-a-list-in-python/>

https://www.w3schools.com/python/python_lists_loop.asp#:~:text=a%20While%20Loop-,You%20can%20loop%20through%20the%20list%20items%20by%20using%20a,by%201%20a

https://www.w3schools.com/python/python_tuples_methods.asp

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

<https://www.programiz.com/python-programming/tuple>

<https://www.geeksforgeeks.org/python-create-list-of-numbers-with-given-range/>

<https://www.tutorialspoint.com/create-list-of-numbers-with-given-range-in-python>

<https://www.geeksforgeeks.org/python-program-to-find-the-sum-of-all-items-in-a-dictionary/>

<https://www.educative.io/answers/how-does-the-dictionary-work-in-python>

<https://www.dataquest.io/blog/python-dictionaries/>
<https://www.programiz.com/python-programming/dictionary>
<https://towardsdatascience.com/working-with-python-dictionaries-a-cheat-sheet-706c14d29da5>
<https://realpython.com/python-dicts/>
<https://www.hackerearth.com/practice/python/getting-started/input-and-output/tutorial/#:~:text=Python%20has%20an%20input%20function,to%20key%20in%20the%20>
<https://www.geeksforgeeks.org/python-input-function/>
<https://www.geeksforgeeks.org/python-input-methods-competitive-programming/>
<https://www.geeksforgeeks.org/taking-input-in-python/>
<https://www.programiz.com/python-programming/methods/built-in/input>
<https://www.edureka.co/blog/input-in-python>
<https://www.simplilearn.com/tutorials/python-tutorial/input-in-python>
<https://pythonlobby.com/input-output-python-exercises-with-solution/>
https://users.cs.utah.edu/~germain/PPS/Topics/while_loops.html#:~:text=A%20%22While%22%20
<https://www.simplilearn.com/tutorials/python-tutorial/python-while-loop>
https://www.w3schools.com/python/python_while_loops.asp
<https://www.programiz.com/python-programming/while-loop>
<https://realpython.com/lessons/basic-while-loop-structure/>
<https://code4coding.com/nested-while-loop-in-python/>
<https://holypython.com/intermediate-python-exercises/exercise-9-python-while-loop/>
<https://blog.finxter.com/python-one-line-while-loop-a-simple-tutorial/>
<https://realpython.com/lessons/one-line-while-loops/>
<https://www.freecodecamp.org/news/python-functions-define-and-call-a-function/>
<https://www.simplilearn.com/tutorials/python-tutorial/python-functions>
<https://pynative.com/python-functions-exercise-with-solutions/>
<https://pynative.com/python-functions/#h-creating-a-function>
<https://www.programiz.com/python-programming/function>
https://www.w3schools.com/python/python_functions.asp
https://www.w3schools.com/python/python_functions.asp
<https://www.programiz.com/python-programming/function>
<https://pynative.com/python-functions/#h-creating-a-function>
<https://pynative.com/python-functions-quiz/>
https://www.simplilearn.com/tutorials/python-tutorial/python-functions#what_are_arguments_in_a_python_function

<https://www.freecodecamp.org/news/python-functions-define-and-call-a-function/>
<https://gyanipandit.com/programming/advantages-of-functions-in-python/>
<https://www.w3resource.com/python-exercises/class-exercises/>
<https://medium.com/@nschairer/python-dynamic-class-attributes-24a89df8da7d#:~:text=Adding%20attributes%20to%20a%20Python,called%2C%20followed%20>
<https://pynative.com/python-classes-and-objects/#h-what-is-a-class-and-objects-in-python>
<https://realpython.com/python-classes/#getting-started-with-python-classes>
<https://www.javatpoint.com/python-objects-classes>
<https://docs.python.org/3/tutorial/classes.html#a-first-look-at-classes>
https://www.w3schools.com/python/python_classes.asp
<https://www.geeksforgeeks.org/python-exception-handling/>
https://www.softcover.io/read/e4cd0fd9/conversational-python/ch6_files_excepts
<https://docs.python.org/3/library/exceptions.html>
<https://www.section.io/engineering-education/files-and-exceptions-in-python/#:~:text=Files%20are%20identified%20locations%20on,when%20a%20program%20is%20>
<https://www.programiz.com/python-programming/exception-handling>
[https://coderpad.io/blog/development/testing-in-python-types-of-tests-and-how-to-write-them/#:~:text=The%20two%20most%20common%20types,unit%20tests%20and%20integration%](https://coderpad.io/blog/development/testing-in-python-types-of-tests-and-how-to-write-them/#:~:text=The%20two%20most%20common%20types,unit%20tests%20and%20integration%20)