Core Python for Everyone

A systematic incremental approach to become software engineer for any student or graduate & every enthusiast!

This book's all programs support latest version of Python.

Madhusudan Mothe
M.Tech., IIT Bombay

Foreword by Atul Soneja, Senior Vice President, Infosys.

Guide to Readers:

Q. I have never done programming as of now. Can I read this book?

Yes, you can read this book from chapter 1. If you are familiar with basics related to computer, you can even directly start studying from chapter 2.

Q. I have done programming in C/C++ / Java etc. Is this book useful?

Anyone who already have exposure of little bit programming can directly read 9th chapter.

Q. I am already Python programmer. Is this book useful to me?

Anyone who is already Python programmer can read “Class” chapter & see if there is any value addition in his/her knowledge.

If you like book content, you may click below Amazon link & appreciate this book.

https://www.amazon.in/Everyone-MTech-Bombay-Madhusudan-Mothe/dp/B07XBL4SVL/ref=sr_1_2?qid=1567584411&refinements=p_27%3AMadhusudan+Mothe&s=books&sr=1-2

Or

http://bit.ly/javaforall

Python Youtube Course 👉 http://bit.ly/youtubepython

Python Udemy Course 👉 http://bit.ly/udemypython

Dedicated to my always Smiling & Cheerful nephew Aayush Kulbhushan Mothe

Core Python for Everyone

ISBN: 9481660113996
First Edition: March 2020
Second Edition: January 2021
MRP: ₹ 499

Quality Knowledge Publisher.
qualityknowledgepublisher@gmail.com

Paperback copy of this book can be purchased from Amazon.
http://bit.ly/smartpython

Foreword

Knowledge about technology is no more a privilege but a necessity to survive in this digital world. Technology has become an integral part of our lives and as businesses strive to accelerate on the path to digital transformation, it continues to evolve to help businesses explore new possibilities and take the digital leap. What is helping technology achieve this is the premise it is built on - language. Just like any other language, the language of computers is also the essence of evolution and the backbone of innovative solutions that helps business stay ahead by adopting Artificial Intelligence (AI), Machine Learning (ML) and Automation with agility.

Python has been one of the most sought-after programming languages by data science practitioners and continues to be for experts dealing in Artificial Intelligence, Machine Learning and Data Analytics. This book by Mr. Madhusudan Mothe brings forth the relevance of Python in today's digital world and simplifies the understanding of Python Programming, explaining the concepts through a mix of theoretical paragraphs and FAQs. Paragraphs cover basic introduction to the topic and FAQs explain the crucial components in a very interactive manner of questioning and answering. The Deep knowledge section has brought the deeper insights on the particular topic in a pointwise and easy to catch manner to the readers of the book.

My congratulations to Madhusudan for this great attempt. It's a complete guide for both professionals and aspirants alike. I would recommend this to all technology enthusiasts who want to learn and develop expertise in Python.

Atul Soneja,
Senior Vice President,
Global Head of Edge and Infosys Nia,
EdgeVerve Systems Ltd. (An Infosys Company)

Preface

When any beginner wants to study Python as his/her first programming language, many of us advice them to study C / C++. This book provides smooth path to learn Python directly without studying C/C++ or any other programming language.

Let's imagine that you are sitting in this plane to start the journey of Python programming world ☺

I welcome all of you to a tour of Python programming world! We are in the position to take off. Please, buckle up your seat belts and get ready for takeoff! You are quite known with a fact that the speed of a plane gradually increases and attains higher speed of 1000km/hr or more. What will happen if the plane takes a take off with high speed like 1000 km/hr? It will simply crash. Similarly, if the complicated concepts are explained without building strong foundation of basic concepts, your confidence may crash. Care has been taken to illustrate fundamental concepts of Python programming in a systematic and lucid manner. **Explanation of complicated concepts in easy words is the major strength of this book.**

Many people have done so much for me over the years and it is not possible to mention all these names here due to space constraint. Hence, I am mentioning few of them ☺ ***I do understand that the list is quite lengthy, but, I take this opportunity to acknowledge them.***

I acknowledge Chandrakant Kunjir, Rajodhan Khalate, Balasaheb Deshmukh, Suyog Raut, Sandip Chorage, Sudhir Ghorpade, Bhaskar Pandagale, Nana Nivangune, Shrikant Malegaonkar, Mahadeo Kadam, Santosh Gaikwad, Soumyajit Sengupta , Pratik Gholkar, Satyendra Singh, Chandrakant Gurav, Vikram Patil, Ramesh Pimple, Shyam Chavan, Vinod Pardeshi, Ashok Sahane, Vishnu Katkar, Pratik Dattatray Patil, Vijay Bhagwan Kolekar, Mrutyunjay Rajendra Ranage, Krantisinh Dilip Khandekar, Anjali Ranjeet Gavade, Murtee Shivraj Patil, Priyanka Salgar, Ranjana Harischandra Gholve, Suresh Bhilawade, Navnath Lokare, Sumangala Pawar, Manjusha Phase, Sidgonda Karigar, Bhagyashi Patil, Rekha Prabhakar Nhalade, Satish Kamble, Satish Karande, Shankar Sagare ,Ghanshyam Adsul, Nitin Raval, Shivaji Gavade, Gajanan Pathurde, Rupesh Pawar, Neeraj Thorat, Pramod Shinde, Pandurang Desai, Prakash Baburao Siddh Raosaheb Siddh, Vilas Khubannavar, Appasaheb Pujari, Dadaso Nargatte, Balasaheb Kushappa, Anna Awate, Anant Desai, Bajirao Hupare, Laxman Pujari, Dhondiram Mali, Sanjay Hukire, Jitendra Shirguppe, Namadeo Mane, Balasaheb Kamanna, Ajay Kumbhar, Yuvraj Mhakave, Nikhil More, Ganesh Kumbhar, Siddhu Devaba, Rajaram Birnje, Mahesh Vasudev, Sachin Sontakke, Sachin Kasbe, Hemant Sangaokar, Amit Kumbhojkar, Siddharth Patil, Ajay Jadhav, Shailesh Narvekar, Vishal Mehta, Sagar Mahajan, Rahul Mahajan, Rajendra Jadhav, Shivagonda Awate, Amar Chile, Vijay Pataskar, Vijay Naik, Amit Mahadar, Sejal Shah, Priyanka S, Delekar family. and many more for their support. Medicos Dr. Sanjay Desai, Dr Suhas Kulkarni, Dr. Nilesh Puri, Dr. Sanjeev Dasarao, Dr. Sunil Bandgar, Dr. Parameshwar Kharat, and Ganesh Patil have been always helpful.

I am thankful to Infoscions Pravin Kulkarni (VP), Rajneesh Malviya (SVP) , C.Devarajan, Anand Sinha (Ex-AVP), Lt. Col. Yogesh Joshi (Retd), Prasanna Mulmule, Dakshinamurty Y., Bhagwan Chaudhari, Tushar Bhalerao, Vinod Sankaran Nair, Julie Karmakar, Neeraj Baderia, Suresh Sonawane, Shraddha Dhamangaonkar, Anant Sabane, Anand Pardeshi, Girish Chhatwani, Kalpesh B.Solanki, Ambadas Adam, Lalit Govil, Abhijit M. Naik , Sandeep Rahul, Kailas Tile, Sarabjeet Singh Vig etc. as well as carpool companions Sameer Chaugule, Pushakar Barve, Dhwal Yeolekar, Sachin Shelar, Sachin Gokhale, Kaustubh Paranjape, Saurabh Pandhare, Rajan Phatak, Aditya Dhole, Ajay Deokar, Amogh Pandharipande, Nitin Doiphode, Akshay Marane, Rahul Paranjape, Sandeep Bhavsar, Ganesh More, Gururaj Sagar, Rohit Bhave, Chaitanya Khadilkar, Niranjan Phansalkar, Makarand Shete etc. I also want to acknowledge tea group members Madhav Dabke, Durgesh Ghotagalikar, Suhas Pachore, Tushar Samanerkar, Hrishikesh Mhaiskar. I am also thankful to chairman Sachin Deshmukh, secretary Suhas Hattarki and all the members of Runwal Society for their co-operation.

I take this opportunity to say thanks to academic professionals Dr.B.P. Bandgar (Ex.Vice Chancellor), Dr. Ashok Pise, Dr. H.T. Jadhav, Dr. C. M. Sewatkar, Dr. Shashi. Ghumbare, Dr. Bhakti Raghavendra Umarji, Dr. Dhananjay Talange, Dr. Balasaheb Patre, Dr. Aniket Khandekar, Dr. Ashok Gaikwad, Dr. Sandip Narote, Dr.Yashwant Kolekar, Dr.Rohit Ramteke, Dr.Sudhir Agashe, Dr. Sanjaykumar Patil, Dr. Uttam Chaskar, Dr. Sanjay Shendge, Dr. Sandeep Meshram, Dr.Sanjay Koli, Dr. Sanjay Patil, Shivaji Rote, Nagnath Biradar and Bharati Narute for their encouragement.

I must express my gratitude to the tremendous co-operation given by Ashok Kumar Ratnagiri, Sujatha Yakisari, Santhosh K.B, Ajay Lavhade, Deepali Tannu, Vivek Mhaswade, Kiran Paltankar and all members of Edgeverve Security & Performance Engineering Team. All my colleagues have been very understanding & supportive.

It gives me pleasure whenever I interact with my COEP friends Abhijit Deshpande, Jayant Mhetar, Anil Kulkarni, Amit Lele, Shantanu Joshi, Paresh Deshpande, Shailesh Kamble, Suyog Bhokare , Vivek Joshi and Ranjit Hogade mainly on social media. The same is true with all my Private High School & V. S. Khandekar schoolmates.

I am also thankful to Navya Bhuwaneswaram, Tavishi Pandey, Gargi Vatturkar, Anurag Jain, Aishwarya Ghatage, Deepa Kumari, Sayali Kadam, Mangesh Gurav, Prajakta Samant, Prajkta Chavan, Mayur Raina, Vipul Raj, Sakshi Jain, Ayushi Yadav, Shubhada Jumde, Lohith Poojari, Milind Vaidya, Siddhi Jadhav because of whom I could make this book better.

Thanks to my former colleagues Sachin Daftary, Kiran Laturkar, Atul Kahate, Manish Inamdar, Prashant Khisti, Manoj Madhav Apte, Atul Edlabadkar, Nitin Sapre, Dattaprasad Vaidya, Abhijit Mahale, Amarendra Gokhale, Pundlik Jumbad, Sanjay Phanshilkar, Balgonda Hulyalkar, Ganesh Ramakrishnan, Milind Bhimarao Joshi, Shailendra Kumar, Abhijit Bhalerao, Tushar Surve, Mandar Gore, Ashwinikumar Buche, Sameer Kajale & Bipin Kumar.

This book will remain incomplete if I do not acknowledge to my parents, wife, sister Sharvari , younger brother Kulbhushan, all members of Mothe family, relatives & native of Pattankodoli who have been standing behind me like a wall. I would like to mention my niece Skyla Wade and nephew Ayush Mothe who always makes me happy whenever I play with them ☺

The explanation of complicated concepts is provided in a lucid language which becomes the major strength of the book. I hope that readers will benefit from this book in maximum possible way. Readers can email me on mothemadhusudan@gmail.com.

Madhusudan Mothe
http: //bit.ly/mothemadhusudan

*** Reader can skip this chapter if familiar with basics of computer & directly start studying chapter 2. ***

Chapter 1

WELCOME TO COMPUTER AND PROGRAMMING WORLD!

1.1 Why Computers?

Imagine that you are at Mumbai V.T. station at 2 o'clock in the midnight. Sleeper class tickets for Delhi express are fully booked and only first-class tickets are available. You need Rs. 2,000 more to purchase a first class ticket. Where can you get these 2000 rupees at midnight? Of course! ATM. Thanks to a computer technology which has made human life easier and luxurious!

Computers have become an integral part of the human life now-a-days. **Automated Railway Reservation System** is a well-known practical example of applications of a computer. You can book railway ticket from internet at anytime from anywhere. Computers are used in **hospitals** to monitor and track patients, doctors, other staffs etc. You can play various **games** on a computer. Application software like **Microsoft word** can be used to create various types of documents. In a nut shell, computer technology is a boon to human life and hence study of computers is important for everyone.

Let's add two numbers, say, 5 and 12. Your brain can do this calculation very easily. Now, tell me what is the addition of 13579 and 2468? It's difficult for a normal brain to process it quickly. However, computer can perform this addition in fraction of a second. Further, suppose you have to add at least such 1000 numbers, this repetitive calculation job will be tedious and there can be mistakes. One can definitely rely on a computer to do this job in fraction of seconds and that too without any mistakes.

The great scientist, Blaise Pascal's father was working in a tax department. His father had to spend nights in doing calculations. While helping, Blaise Pascal also faced similar tedious situation of repetitively adding tax values and hence invented the calculator in 1643. It was used for calculation purpose till 1799 in Europe. Then, Charles Babbage, famously known as a **Father of a computer**, developed analytical Engine performing multiplication and division.

1.1.1 What is a Computer?

Computer = compute + er

Computer is a general purpose machine (electronic device) that does accurate (repetitive) computations million or even billion times faster than normal human being. The calculation can be of arithmetic or logical[1] type.

Atanasoff-Berry invented First Electronic Digital Computing Device in 1937 that was capable of solving up to 29 simultaneous linear equations.

We have just read that computer does computations. It can add any 2 numbers like 13579 and 2468 in a fraction of a second. Being a beginner, ***let's try to utilize a computer to add 2 simple numbers and gradually gain the knowledge to become a best programmer***. We need to give set of instructions in a logical order to perform this task of addition.

Program is a set of instructions arranged in a logical sequence to perform a certain task.

Before using a computer, let us quickly understand the manual process[2]* of the same which everyone knows.

Input Data[3] : 2 Numbers
Output Data[4] : Number which is the addition of these 2 numbers

If you want to ask the addition of 2 numbers to your friend, following steps are involved.

- You will tell 2 numbers to your friend which he/she will hear it by an ear.
- Friend's brain will store and add these 2 numbers.
- He/she will communicate the addition of these 2 numbers by speaking to you.

Ear, mouth and brain acts as input, output and processing unit of a human being respectively. As we want to get this execution done from a computer, let's understand input, output and other units of a computer.

1.1.2 Block Diagram of a Computer

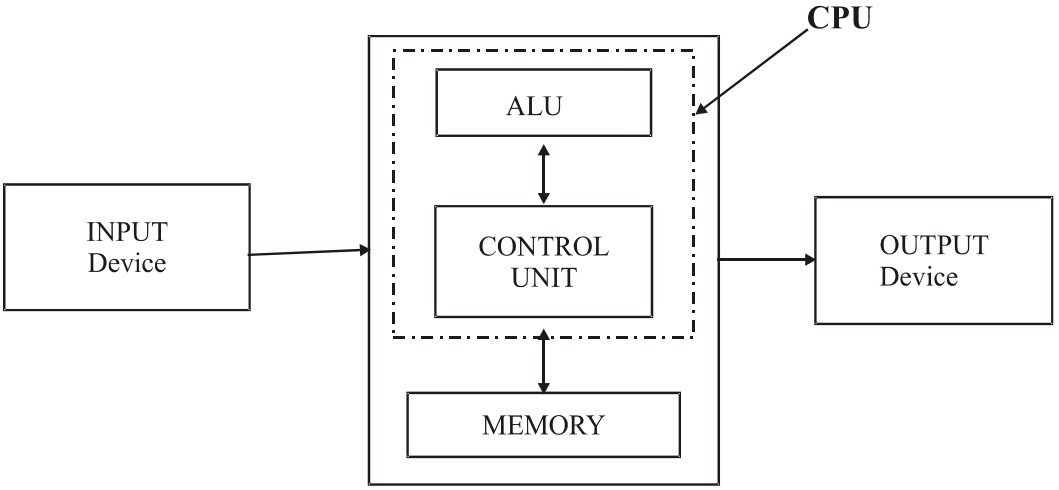


Fig. 1.1

[1] Logical calculations will be discussed in the next chapter.
[2] Manual process is the process executed by a human being.
[3] Data received by a computer from user is referred as an input data.
[4] Data communicated (displayed) by a computer on a monitor, printer etc. is referred as an output data.

Computer can be logically divided into following four parts.

1.1.2.1. Input Unit

We need to input two numbers to a computer to get the sum of these two numbers. What are the parts of a computer that can receive this input data? Computer receives information through input devices like

i. Keyboard ii. Mouse iii. Speaking to a computer iv. Joystick v. Track Ball vi. Scanner
vii. Network (One can access a file on the network located on the other computer.)

For the time being, beginner may consider keyboard, mouse as i/p devices.

1.1.2.2. Memory Unit

Memory is a unit of a computer where data is stored.
In case of the above addition of two numbers program[5], user enters two numbers as input, let us say through keyboard. When this data is successfully received by a computer, it is saved in a memory.

Computer memories can be categorized as
I. Primary Memory
II. Secondary Memory

I) Primary Memory

Primary memory is directly accessed by a CPU. It is volatile in nature means data stored in it is lost if the power is switched off (Exception: ROM). Primary memories can be further classified as

i. RAM (Random Access Memory)

It provides random access to data stored in it. Data stored on RAM can be changed.

The main feature of RAM is that the time required to retrieve data from any memory location is constant irrespective of the memory location address. That's why it is known as RAM.

ii. ROM (Read Only Memory)

It is a read only memory. Once information is stored on ROM, it cannot be easily modified or erased.

II) Secondary Memory

It is not directly accessed by a CPU. It is a permanent storage of the data that is nonvolatile in nature. It is also cheaper and lesser in speed than primary memory.

Secondary memory devices are magnetic tape, diskette, hard disk, CD ROM etc.
When you ask mobile number to someone, can you note it or just store it in your brain forever? It is not possible for a normal brain to store all the numbers. Hence, we note the data in a diary. **Here, human brain acts as a primary memory, while, diary plays a role of secondary memory.** We can retrieve the data from the brain very quickly as compared to the data written in the diary. Diary, notebook etc. acts as secondary memory for the human being.

[5] Program is a set of instructions arranged in a logical sequence to perform certain task. e.g. We need to write set of instructions in a logical sequence to add the numbers 13579 and 2468.

When someone calls you, you hear it by ear and your brain takes certain action based on the message it receives. There must be certain programs existing in human memory by birth which contains instructions to recognize voice, to produce certain voice by mouth etc. These programs are always alive for normal human being and must be residing in read only memory of human being. Similarly, ROM stores the data that cannot be erased. ROM contains the programs that are required for the functioning of a computer. e.g. BIOS program starts a computer.

FAQ[6]

Q. It is true that we cannot forget the image of our best friend even if we try our best to forget it. Can't it be an example of read only memory of human being?

When we revise certain study material, we won't forget it. However, if we do not revise it, we forget it slowly. Similarly, you will slowly forget the image of your friend if you do not remember or memories him/her. Practically, it is unavoidable to remember the close friend and his/her image gets revised in our memory. Hence, we cannot forget it. Therefore, it is not an example of ROM of human being.

1.1.2.3. Central Processing Unit (CPU)

It is the overall administrator (co-coordinator) of a computer.
CPU has two main parts.

I. Control Unit
II. ALU

I. Control Unit

The control unit (CU) reads instructions from memory, decodes[7] and executes.
When the instruction requires to store data into a memory, it receives the data from input device like a keyboard and stores it into a memory. When the instruction requires calculation, it passes the required information to the ALU.
It is also referred as a brain within brain because based on the result of the execution of the instruction by control unit, operations of other parts of a computer takes place.

CPU speed is a feature that tells how quick CPU can execute an instruction.

The speed of the CPU is 2.0 GHz. It means the CPU can execute 2×10^9 instructions per second.
Many today's computers have multiple CPU's.

II. Arithmetic and logic unit (ALU)

It can perform arithmetic and logical calculations on data. e.g. It can add two binary numbers.

1.1.2.4. Output unit

Once this calculation is done by a computer, the calculated sum (output) can be sent to the following output devices.

i. Monitor/VDU ii. Printer iii. Soundcard iv. Robot arm
v. Video vi. Network like internet vii. Film etc.

---

[6] **FAQ** stands for general/frequently asked question/questions. This questions generally pops up in reader's mind while reading or understanding a program. **This approach is used to effectively deliver the knowledge.**

[7] Convert the instruction into binary language.

For the time being, beginner may consider monitor, printer as output devices.

Let's compare Primary and secondary memory.

Primary Memory	Secondary Memory
CPU can access it directly.	CPU cannot access it directly.
It is volatile (Exception: ROM).	It is permanent.
The speed is high.	The speed of accessing the data stored in secondary memory is slow.
The cost is more.	The cost is less.
RAM, ROM is an examples of primary memory.	Secondary memory devices are diskette, hard disk, CD ROM etc.

Let's compare RAM & ROM.

RAM	ROM
The data stored on RAM can be modified.	The data stored on ROM cannot be modified easily.
It stores data temporarily. When the machine is switched off, the data stored on RAM vanishes.	It stores data permanently.

1.2 Machine Language

Hope you have understood the basic concepts explained in previous sections.
Let's go one step ahead and try to use computer practically to add two numbers. It is necessary to give right instructions to a computer to add two numbers. Oh, but, can a computer directly understand the instruction "Add two numbers and show the result?" No. We must have to give this instruction in the language which computer can understand. Then, what is the language of a computer?

Computer can only understand binary language known as a machine language. Binary language has only 2 digits i.e. 0 and 1.

Hence, the following instructions need to be written in a machine language to add two numbers.
Do not try to understand the following instructions; just see that how complicated and messy instructions (code) are written in a machine language to add two numbers.

Binary Instruction	Meaning
010 0 001101	Store the value of the Accumulator in memory location 13.
001 1 000101	Load the value 5 into the Accumulator.
010 0 001110	Store the value of the Accumulator in memory location 14.
001 0 001101	Load the value of memory location 13 into the Accumulator.
011 0 001110	Add the value of memory location 14 to the Accumulator.
010 0 001111	Store the value of the Accumulator in memory location 15.
111 0 000000	Stop execution.

Machine language is the basic language of a computer which is directly understood by it. It consists of binary numbers that tells computer to execute basic operations like printing, adding 2 numbers etc.

Advantages of Machine Language:

1. Though it is tough to write programs in machine language, the execution is fast.
2. It is directly understood by a computer. Other languages are not directly understood by a computer.

Disadvantages of Machine Language:

1. As one has to write the instructions in the number format, chances are always there of doing mistakes in the program.
2. Machine languages are specific to machine. It means the program executed on one machine cannot be executed on the other machine because the binary representation for various instructions are different for different machines.

1.3 Assembly Language

Human being is inventive and innovative by nature. Machine language instructions are in the form of the numbers (that too only 0 and 1) and difficult to read. People started to use English like abbreviations (Mnemonics) to give instructions to a computer to perform a particular task.

Binary instruction or set of instructions are mapped to English abbreviations known as mnemonics.

You can just have a glance on below equivalent assembly language instructions for the above machine language instructions.

```
mov     a1,0Ah
mov     b1,14h
mov     eax, a1
add     eax, b1
mov     c1, eax
```

Assembly language is a machine-orientated language in which mnemonics (symbolic names) are used to represent each machine-language instruction. It is a programming language that is one step above machine language. Each CPU has its own specific assembly language. Hence, assembly language program executed on one machine cannot get executed on the other machine of different processor (CPU).

Advantages:

1. Assembly language is easy to write and also chances of making mistakes in writing instructions are less as compared to machine language programs.

Disadvantages:

1. Assembly language instructions (code) is not directly understood by a computer. It needs to be converted into machine language instructions by another program called as an assembler.
2. Though easier than machine language, writing programs in assembly language is complicated & messy.

Machine and assembly languages are low level languages.

Assembler is a program that converts assembly language instructions into machine language instructions.

As seen above, one has to write so many instructions even to write a simple program. Further, middle and high level languages are developed to speed up the programming.

1.4 Operating System

An operating system (OS) is a set of computer programs that manages the hardware and software resources of a computer. It is heart of the computer.

Operating systems perform basic tasks, such as recognizing input from the keyboard, sending an output to the display screen, keeping track of files and directories on the disk, controlling peripheral devices such as disk drives and printers etc.

Windows, Unix, Linux, Macintosh etc. are the operating systems.

Unix is a multiuser system. Many users can work on the same machine by connecting through terminals. It means all the users share the CPU and memory of one machine at the same time.

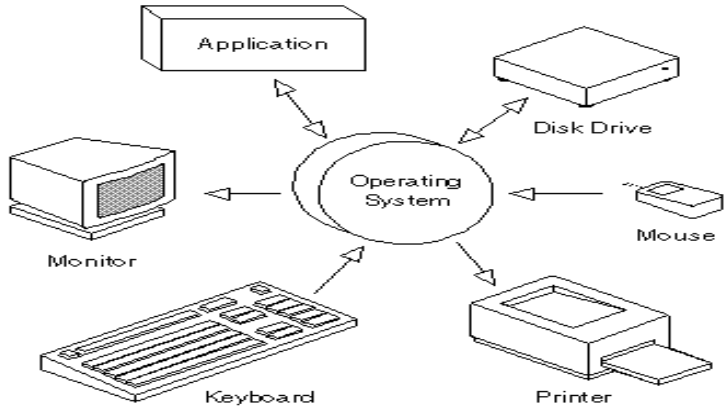


Fig. 1.2

1.5 Points to Remember

1. Computer has mainly four parts. 1. Input Device 2. CPU 3. Memory 4.Output device
2. Primary memory is directly accessed by CPU. It is volatile (Exception:ROM) in nature.
3. Computer performs arithmetic and logical calculations.
4. An operating system (OS) is a set of computer programs that manages the hardware and software resources of a computer.
5. Computer directly only understands machine language.

FAQ:

Q. Does computer directly understand assembly language instructions?

No. Computer can only understand binary language. Hence, the assembly language instruction needs to be converted into a machine language instruction.

Q. Can computer directly understand any Python program?

Basic language of a computer is a binary language and it can directly understand binary instructions. Computer cannot directly understand Python language instruction(s). Hence, we need to convert these instructions in binary language.

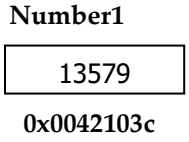
Q. Why computer can directly understand 0 and 1 and not any other numbers or alphabets?

Computer is made up of transistors which are used in on-off mode. When transistor is on, it is treated as 1 and when transistor is off, it is treated as 0. That's why computer can only understand 0 and 1.

Q. Who finally executes binary instructions inside executable file?

CPU reads every instruction from the instruction register and executes it one by one.

Q. Computer understand only 0 and 1. Then, how Number1 is stored in a decimal format as shown in below figure?



The above representation is only for understanding purpose at a beginner level.
Computer understand only 0 and 1. Data is stored in a memory. Memory is divided into units referred as memory locations. Each memory location can store maximum 8 bits. Every memory location is identified by a unique address. e.g. 0x0042103c is an address. Address is a unique integer number.
When any data is stored in a computer, memory should be allocated. In case of an integer, 4 consecutive memory locations are allocated. i.e. the size of an integer data type is 4. The integer value 13579 is converted into binary form. i.e.00110101 00001011. Hence, it is actually stored as

00000000	00000000	00110101	00001011
0x0042103f	0x0042103e	0x0042103d	0x0042103c

Little Endian Address Representation of 13579

Integer value can be positive or negative. Hence, the last bit i.e. 32th bit indicates the sign of the integer value stored. If 32th bit is 0, the integer value is positive, else it is negative.

1.6 Deep Knowledge Section[8]

Q 1. Why RAM is faster than hard disk memory?

RAM is IC based data storage. Hence, it allows to access the stored data in any order- means randomly and **without any physical movement of the storage medium / reading head.** In case of secondary memory like hard disk, data is accessed sequentially. Hence, it takes time to move the read/write head from one location to other memory location from where data needs to be read. Physical movement of head and physical location of data causes more access time for secondary memory data access.

Q 2. How computer starts?

i. When you turn on your computer, the **microprocessor** passes control to the **BIOS** program which is always located at the same place on (Erasable Programmable Read-only Memory) EPROM.
ii. BIOS does power-on self-test (POST) to make sure all computer's components are operational.

[8] **Deep knowledge Section should be read after developing average level in Python. Beginner may skip this section in the first reading.**

iii. BIOS looks for the system files at a specific place on your hard drive and copies information from it into specific locations in RAM. This information is known as the **Boot Record** or Master Boot Record.
iv. The boot sector on a disk is always the first sector on the first track on the first head.
v. It then loads the boot record into a specific place (hexadecimal address 7C00) in RAM.
vi. The boot record contains a program that BIOS now branches to, giving the boot record control of a computer.
vii. The boot record loads the **initial system file** (for example, for DOS systems, IO.SYS) into RAM from the diskette or hard disk.
viii. The initial file (for example, IO.SYS, which includes a program called SYSINIT) then loads the rest of the operating system into RAM.
ix. The initial file (for example, SYSINIT) loads a system file (for example, MSDOS.SYS) that knows how to work with the BIOS.
x. One of the first operating system files that is loaded is a system configuration file (for DOS, it's called CONFIG.SYS).
xi. Another special file that is loaded is one that tells which specific applications or commands the user wants to have included or performed as part of the boot process. In DOS, this file is named AUTOEXEC.BAT. In Windows, it's called WIN.INI.
xii. After all, operating system files have been loaded, the operating system is given control of a computer and performs requested initial commands and then waits for the first interactive user input.

Questions

Q.1 State True or False.

i. Computer cannot think.
ii. Computer can make a mistake.
iii. Data stored on a RAM can be changed.
iv. Information stored on ROM cannot be erased.
v. Data on secondary memory is volatile in nature.
vi. ALU performs arithmetic and logical calculations on data.
vii. Program is a set of logical instructions used to perform a certain task.
viii. Computer can only understand binary language known as a machine language.
ix. Machine and assembly languages are low level languages.
x. Computer can directly understand the assembly language code.
xi. Assembler is a program that converts assembly language code into machine language instructions.
xii. An operating system (OS) is a set of computer programs that manage the hardware and software resources of a computer.

Q.2 What is primary memory? What is RAM and ROM ?
Q.3 What is secondary memory?
Q.4 What is ALU?
Q.5 What is program? What language computer can understand directly?
Q.6 What is an assembler? What it does?
Q.7 What is an operating system?

Answers:

Q1. i. True ii. False iii. True iv. True v. False vi. True
vii. True viii. True ix. True x. False xi. True xii. True

A. Brief History of Computer Technology

A complete history of computing would include a multitude of diverse devices such as the **ancient Chinese abacus**. Abacus was the first device known to carry out the calculations. It works by sliding beads back and forth on a frame with the beads on the top of the frame representing fives and on the bottom ones. **Charles Babbage's analytical engine (1834).** This machine can perform all the four operations that are addition, subtraction, multiplication and division.

In 1960s, analog computers were routinely used to solve systems of finite difference equations arising in oil reservoir modeling. In the end, digital computing devices proved to have the power, economics and scalability necessary to deal with large scale computations. Digital computers now dominate the computing world in all areas ranging from the hand calculator to the supercomputer and are pervasive throughout society.

The advent of **the first electromechanical computers** was an exciting time, because these scamps could actually provide significant amounts of computational power. Development of electromechanical computers continued into the 1960's. The Minivac 601 was a model introduced in 1961 with its name derived from the Sperry Rand Univac computer.

The **evolution of digital computing** is often divided into generations. Each generation is characterized by dramatic improvements over the previous generation in the technology used to build computers, the internal organization of computer systems, and programming languages. Although not usually associated with computer generations, there has been a steady improvement in algorithms, including algorithms used in computational science.

B. History of computers

ABACUS (2400 B.C.E.)	**Anonymous Chinese**	**It is the earliest known tool for use in computation.**
Calculating Clock (1623)	**Wilhelm Schickard**	**First Automatic Calculator. Performing Addition and Subtraction.**
Pascal's Calculator (1643)	**Blaise Pascal**	**Used for Taxes in France Until 1799. Performing Addition, Subtraction.**
Analytical Engine Difference Engine.	**Charles Babbage**	**First to conceptualize and design a fully programmable computer early at 1820.**
Hollrith and tabulating Machine (1890)	**Harman Hollerith**	**Device could automatically read census information which had been punched onto card.**
First Electronic Digital Device (1937)	**Atanasoff-Berry**	**The machine, conceived in 1937, was capable of solving up to 29 simultaneous linear equations and was successfully tested.**

Chapter

2

WELCOME TO PYTHON PROGRAMMING WORLD!

2.1 History

Python is a widely used general-purpose high-level programming language.

Reading and writing instructions using Python language is much like reading and writing regular English statements. Hence, it is high-level programming language. It was initially conceived by **Guido van Rossum**[1] in 1980. Python's implementation began in 1989, while, first version was released in 1989. Python was mainly developed for emphasis on code readability and provides easier way of writing instructions to develop programs.

Python language is nowhere related to Python snake 😊 Guido Van Rossum named it Python as he wanted to keep the name short, sweet & slightly mysterious. Also, at the time of implementation of Python, he was reading a comedy series "Monty Python's Flying Circus". Hence, he named it Python.

Python is a powerful language that you can use to create games, develop graphical user interfaces, (GUI)/ web applications etc. Popular companies Google, Amazon, Facebook, Uber etc. use Python.

Python is currently managed by Python Software Foundation. Most recent major version is 3.9.

Why Python?

- ✓ Python works on different platforms (Windows, Mac, Linux, Raspberry Pi, etc.)
- ✓ Python has a simple syntax similar to English language.
- ✓ Python allows developers to write programs in fewer lines as compared to other languages.
- ✓ Python runs on an interpreter system meaning that code can be executed as soon as it is written.
- ✓ Python can be treated in a procedural way, an object-orientated way or a functional way.
- ✓ It is possible to write Python programs in an Integrated Development Environment, such as Thonny, Pycharm, Netbeans or Eclipse especially managing large number of Python files.
- ✓ Python programming language has a global community with millions of software developers who interact online and offline in thousands of virtual and physical locations

---

[1] Guido van Rossum (31st Jan, 1956) is a Dutch programmer and currently living at Belmont, California.

2.2 Embarking Python Programming: First Python Program

I know you are eager to write first Python program, aren't you? Welcome to the world of Python programming!

Let's write a Python program that performs addition of 2 numbers (i.e. 13579 and 2468) and display it. Though it is a simple program, it will give you an exposure to the various aspects of Python programming.

Before writing any program, it's better to understand how the same task can be done manually.

You will manually do this calculation as below.

1. You will remember value 13579 as number1 in your memory.
2. You will remember value 2468 as number2 in your memory.
3. You will add number1 and number2.
4. You will say "The sum is = 16047".

In a programmer's language, the above mentioned steps can be written as below.

- Store a value 13579 in the computer's memory.
- Store a value 2468 in the computer's memory.
- Add these 2 numbers and store the value in the computer's memory.
- Display the addition of these 2 numbers (sum).

Don't get scared with the word "Algorithm". The above 4 steps are collectively known as an "Algorithm" for a program that adds 13579, 2468 and displays the result.

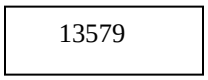
Algorithm is a set of steps arranged in a logical sequence to perform a certain task.

Step 1: Store a value 13579 in the computer's memory.

Following instruction (command) stores numeric value like 13579 in computer's memory.

Number1=13579

For easier understanding, you can see below representation.



0x0042103c

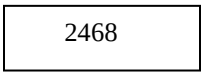
Here, the value 13579 is stored in a memory location 0x0042103c[2].
Number1 is referred as a variable name.

Step 2: Store a value 2468 in the computer's memory.

Similar like step 1, following instruction (command) stores numeric value like 2468 in computer's memory.

Number2=2468

For easier understanding, you can see below representation.



0x00422031

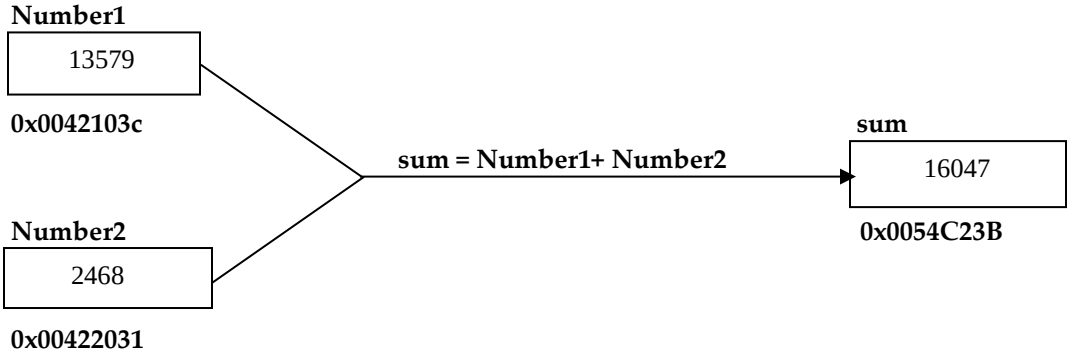
Here, the value 2468 is stored in a memory location 0x00422031[3]. Number2 is referred as a variable name.

[2] This number is presented in the hexadecimal system format.
[3] This number is presented in the hexadecimal system format.

Step 3: Add these 2 numbers and store the sum under a new variable name.

Following instruction adds Number1 & Number2 and stores in a variable sum
sum = Number1 + Number2

For easier understanding, you can see below representation.



Step 4: Display addition of these 2 numbers (sum).

print(variablename) is a standard command /instruction (function) that displays the value stored at the memory location variablename on the output device like a monitor. As we want to display the contents of a variable sum, the instruction is

print (sum)

The complete program of addition of 13579, 2468 and displaying the result can be written as below.

Program 2.1:

```
Number1=13579                # step 1
Number2=2468                 # step 2
sum = Number1 + Number2      # step 3
print(sum)                   # step 4
```

Output:
16047

FAQ:

Q. Every memory address is uniquely identified by a number like 0x0042103c. Then, why are we referring allocated memory by a name?

As it's difficult to refer memory locations by a number, variable names are used to refer memory locations where data is stored. As the program size increases, it becomes difficult to refer memory locations by an address. Hence, names are used. It makes easier to read and write a program.

Q. What are literals?

A constant value in Python is created by using a literal representation of it. e.g. 100, 98.2, 'x'.
Literals in Python are a **sequence of characters** (digits, letters, and other characters) that represent constant values to be stored in variables. 13579 and 2468 are literals used in program 2.1.

Q. Can I use variable names a, b instead of Number1, Number2?

Yes, you can use it. However, as the program size increases, it becomes difficult to understand the meaning of the data the variable is containing. It's better programming practice to use relevant variable names to improve the readability of a program.

Q. What is a function?

Function[4] is a set of instructions arranged in a sequence to perform certain task.

Function is component of a program that provides particular functionality. e.g. print function is used to display the value of the variable sum on the screen in above program. Every function has input and gives output accordingly.

Q. What is the meaning of the "# step 1"?

It is a comment. Text written after # is ignored by the compiler. Helpful when anyone reads a program.

Q. What is a comment? What are the different types of comments?

It is information written in a source code that is ignored by a compiler when the source file is compiled.

Advantages of using comments:

- Comments improve the readability of the program.
- It is easy to read and understand a program with comments.
- Useful when a new programmer has to work on the source code.
- Comments help at the time of maintenance[5]/enhancement of the application.

Python only supports a single line comment. A **Single line comment is** any text written after **"#"**. However, one can comment multiple lines by writing that multiline text between ''' ''' triple inverted commas. This is actually known as documentation strings (docstrings).

We will use these comments in the later part of this chapter/book.

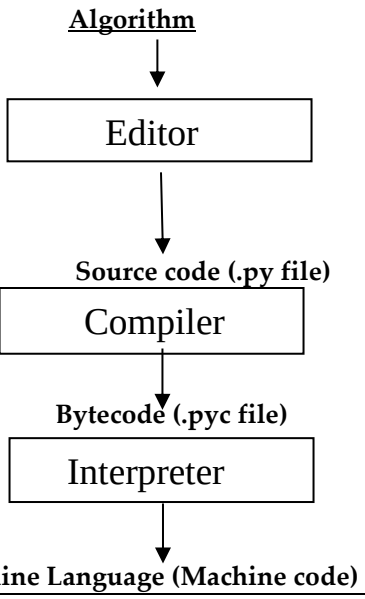
2.3 Typical Python Language Environment

Following are the steps involved in the development of an executable file from writing a Python language program in an editor.

Editor is a program (tool) where one can write instruction/ instructions.

Compiler checks the grammar of every instruction produces corresponding byte code and generate corresponding class file.
e.g. Compiler compiles Test.py into Test.pyc

Interpreter interprets every line in a class file (e.g. Test.pyc in this case), converts it into executable code (machine code) and executes it.



[4] Function will be discussed in chapter 5.

[5] Maintenance is a process in which defects in the program are found and fixed. Concept of maintenance of a vehicle is similar to a maintenance of a program.

2.3.1 Editor

Editor is a standard program that allows you to edit your source program code. Program is written in a file in an editor. There are many editors available, however, we will use notepad as it is good for beginners.

Every Python source program file name ends with .py extension. e.g. Test.py

2.3.2 Compiler

Every language has a grammar. Once the program is written, we need to check whether it follows grammar rules. In case of English, if someone says to you

"program to required is a compiler compile".

You won't understand it. This is actually "Compiler is required to compile a program." in a correct English format as per the grammar rules. Hence, we must give instructions to a computer which follow language grammar rules. e.g. print statement (command) should have brackets.

If you forget to write = in the above program, compiler produces syntax error. Let's remove =, compile the program, you will get syntax error. If program doesn't obey any grammar rule, it gives syntax error.

Compiler is a standard program that checks whether every instruction in the program is written as per the Python language standard grammar rules or not, translates into bytecode.

In a nutshell, compiler is a program that takes translation unit (file) as input and produces .pyc file as output.

2.3.3 Interpreter

As you know, computer only understands machine language. After compilation, Python program gets converted into .pyc file. e.g. Test.py is compiled into Test.pyc. Hence, it is necessary to convert these bytecodes into machine code. Conversion of bytecodes into machine code is done by interpreter.

Component	Input	Output	Functionality
Editor	Algorithm	Source Program	Instructions in Python language are written in a file using an editor. A file where a program is written is known as a **source program**. e.g. Test.py
Compiler	Source Program	Bytecode	Compiler **checks** whether every instruction in a program is written as per the Python language standard **grammar**. It generates compile time error/warnings in case of any discrepancy. Python compiler produces an intermediate code known as bytecode. e.g. Python compiler compiles Test.py into Test.pyc
Interpreter	Bytecode	Executable File	Python interpreter converts the bytecodes generated by Python compiler (.pyc file e.g. Test.pyc) into machine code.

Table No:2.1

2.3.4 Python Error Types

There are 3 types of error that may occur during writing a Python program.

I. Syntax Error/Compile time error

When any instruction in the program is not as per the Python language grammar rule, compile time error occurs. e.g.

- Misspelled variable and function() names e.g. prin ("hi")
- Incorrect Indentation
- Improperly matches parentheses, square brackets, and curly braces
- Incorrect format in selection and loop statements

There are many ways by which this type of error can occur.

II. Runtime Error

Runtime error occurs while executing any instruction in the executable file. e.g.

- Dividing a number by zero. i.e. a=0, k=90/a;
- Trying to open a file that doesn't exist. You will study this in File chapter.

Execution of this instruction will give you runtime error.

III. Logical Error

The aim of the above program 2.1 is to display the sum of the 2 numbers. i.e. 13579 and 2468. If we write the instruction print(Number1) instead of print(sum); we will get incorrect result. If you use – (minus) symbol instead of +(plus) symbol by mistake, it will give you wrong results which is also a logical error.

Common examples where logical error may occur are:

i. Multiplying when you should be dividing
ii. Adding when you should be subtracting
iii. Opening and using data from the wrong file
iv. Displaying the wrong message

Do not get discouraged while compiling and executing your first program. You will learn more if you come across many errors initially.

You must be familiar with the name **Thomas Edison** who invented the bulb. He failed 10,000 times to produce an electric bulb and then successfully invented first electric bulb. When someone asked him about his failure, he said" Now, I know 10,000 different ways by which bulb cannot be produced." **Take motivation from him and do not get nervous even if you initially fail many times to compile & execute a Python program.** You will understand different types of errors that become hurdle to compile & execute a Python program. This experience will help you to develop big sized programs with ease.

2.4 Python Data Type

We have already used integer data type in the programs 2.1. In this program, we have created *Number1* variable of integer data type to store value 13579. Similarly, we have created integer variable *Number2* to store value 2468. Further, we have created integer variable *sum* to store addition of these two numbers. In a nutshell, we are familiar with integer data type.

Now, can we store decimal values like "11.31" or characters like 'x' by creating an integer variable? No, Reason is simple. In practical life, we store books in the shelf. Do we store liquid in the shelf? No, it's not possible. We require container to store liquid. Similarly, Python supports various data types so that programmer can store various type of values depending upon the need by creating variables of those data types.

Following section shows various data types that Python supports. It specifies range which indicates the maximum and minimum value that variables of that data type can store.

2.4.1 Numeric

This data type includes integers (e.g. 11,121 etc.), float (e.g. 23.222,343.2234 etc.) & complex (e.g. 2+2i, 5+2323j etc.). You may learn integer and float data type and skip learning complex number data type if you are not familiar with complex numbers.

(i) integer

What if someone wants to store integer values like 11,45675641, -1234, etc.? Is there any data type in Python whose variables can store numerical values? Yes, Python supports integer data types whose variables are used to store numerical values. These are referred as **int**. They are positive or negative whole numbers with no decimal point. Integers in Python 3 are of unlimited size.

Program 2.2

```
var1=526                #var1 is the variable, whereas, 526 is the value stored.
var2 = 1231564          #var2 is the variable, whereas, 1231564 is the value stored
sum=var1+var2
print("The sum is: ",sum)
```

Output:
The sum is: 1232090

(ii) float

What if someone wants to store decimal values like 11.23? Is there any data type in Python whose variables can store decimal values? Yes, Python supports float data types whose variables are used to store decimal values.

Let us understand below program 2.3 which stores decimal value 11.23 in float variable var2.

Program 2.3

```
var1=22.36              #var1 is the variable, whereas, 22.36 is the value stored.
var2=11.23              #var2 is the variable, whereas, 11.23 is the value stored.
sum=var1+var2
print("The sum is: ",sum)
```

Output:
The sum is: 33.59

Floats may also be in scientific notation, with E or e indicating the power of 10 ($2.5e2 = 2.5 \times 10^2 = 250$). Size of the float data type is 4 bytes.

(iii) complex

If you are familiar with complex number, you can read this, otherwise, you can skip this subsection. What if someone wants to store the complex numbers like 1+2j,etc in Python? Is there any data type in Python whose variables can store complex values? Yes, Python supports complex data types whose variables are used to store complex values and its associated functions using the file "cmath". Complex numbers have their uses in many applications related to mathematics and Python provides useful tools to handle and manipulate them.

A complex number is represented by " **x + yi** ". Python converts the real numbers x and y into complex using the function **complex(x,y)**. The real part can be accessed using the function **real()** and imaginary part can be represented by **imag().**

```
var1 = 1 + 2j          #var1 is the variable, whereas, 1+2j is the value stored.
var2 = 2 + 4j          #var2 is the variable, whereas, 2+4j is the value stored.
print("Addition =", var1 + var2)
```

Output:
Addition = (3+6j)

2.4.2 String

What if the user wants to store the string literals (e.g. "Hello" "etc" or 'Hi', 'Helllo' etc.)? Python provides String data type for that. String literals in Python are surrounded by either single quotation marks, or double quotation marks. e.g. 'hello' is the same as "hello".

Program 2.3

```
var3="Hello"       #var3 is the variable, whereas, Hello is the value stored.
print(var3)
```

Output:
Hello

The above program 2.3 can be written in the following form.

```
var3='Hello'       #var3 is the variable, whereas, Hello is the value stored.
print(var3)
```

Output:
Hello

It is seen from above programs 2.1,2.2 and 2.3 that we have directly saved the values into the variable names. e.g. We saved integer 1231564 into variable var2, float 11.23 into variable var1 and String "hello" into variable var3.

Then, how compiler can understand the data type of the values stored inside variable? Yes, Python provides in-built command (function) that displays type of the data that specified variable holds.

Program 2.4

```
var1=1231564
var2=11.23
var3="Hello"
```

```
print(type(var1))
print(type(var2))
print(type(var3))
```

Output:

```
<class 'int'>
<class 'float'>
<class 'str'>
```

2.4.3 bool (Boolean)

Apart from arithmetic evaluations like addition, subtraction, what if the user wants to evaluate logical operations & store the result in a variable for further using it in the program execution? Python provides a data type known as "bool". It gives the output as either True or False.

e.g. If number a is greater than b or not i.e. a>b, it evaluates it to either True or False.

```
var4=1544>98232
print ("Result is ", var4)
print(type(var4))
```

Output:

```
Result is False
<class 'bool'>
```

You can execute above program by manually changing values of var4 and see the result. You will get output as either True or False.

Q. What is *True* or *False*?

True and *False* are keywords in Python.

Q. What is a keyword? Enlist the keywords in Python.

Keywords are identifiers (names) reserved by the language for special use. Every keyword has a special meaning. Below is the list of keywords in Python.

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

Q. Guess the output of the following programs.

I.

```
 x
print(x)
```

Output:

```
Compilation Error
```

Variable x is not initialized; Hence, it is compilation error. It is necessary to initialize variables in Python before you use it.

II.

```
Number1=13579
Number2=2468
Number1=4517
sum= Number1+ Number2
print(sum)
```

Output:
6985

The output is a 6985 because the variable Number1 is declared twice and hence, the latest initialized value is overwritten. i.e. 13579 is overwritten by 4517. Hence, Number1 is 4517 which is added to 2468 & the sum is 6985.

III.

```
Number1=13579
Number2=2468
sum= Number1+ Number2
sum=156
print(sum)
```

Output:
156

The addition of 13579 and 2468 is first stored in a variable sum.
However, the next instruction sum=156 changes the contents of the variable sum to 156. Hence, the value stored in the variable sum i.e. 156 is displayed as output.

IV.

```
Number1=1.4E-45                #step 1 Variable Declaration
print(type(Number1))
```

Output:
<class 'float'>

Python supports two categories of data types.

- Primitive data types.
- Non-primitive data types.

The primitive or the basic data structures are the building blocks for data manipulation. They contain pure and simple values of a data. In Python there are four types of primitive variable:
Integer, float, String and Boolean. Non-primitive stores not just a value, but rather a collection of values in various formats. The non-primitive data structures are further divided as Arrays, Lists, Files.

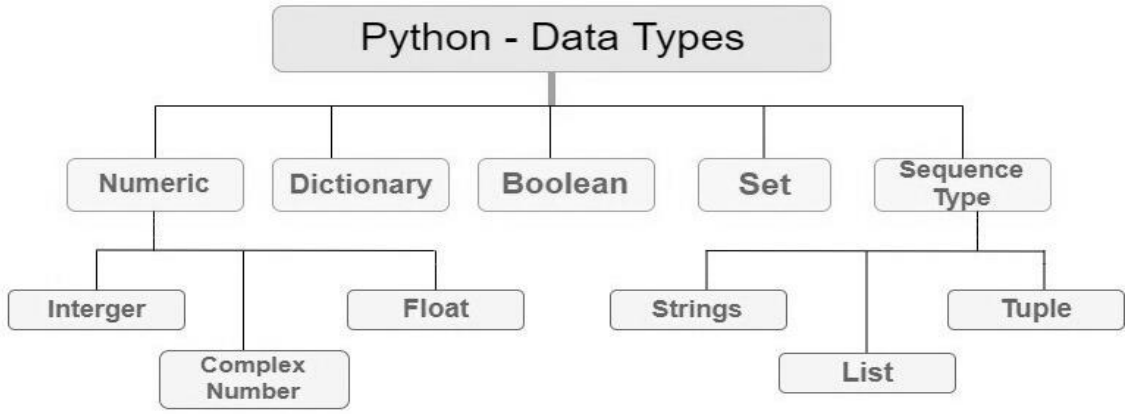
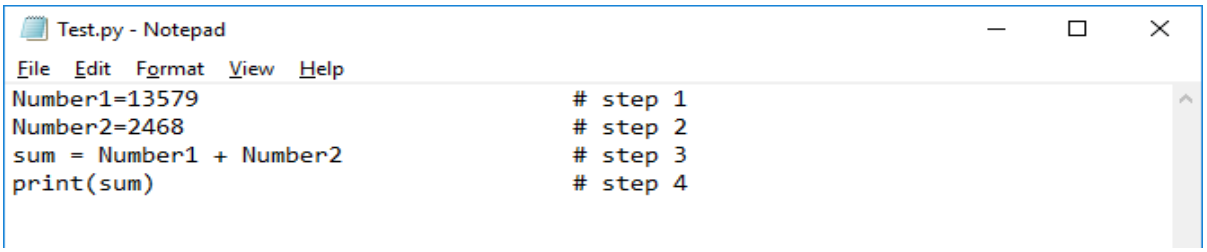


Fig.: 2.1

2.5 Python Program Development & Execution

Reader should install Python latest version by following steps mentioned in the Appendix of this chapter before executing below steps.

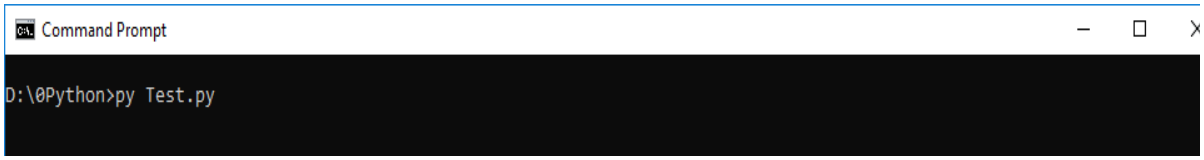
1. Write the simple addition of 13579 and 2468 numbers program using notepad.



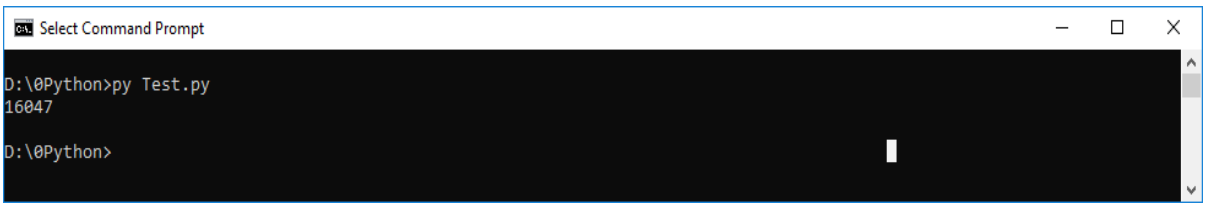
Save this program as **.py** file. Example **Test.py.**

2. Open the command prompt. Change the directory on the command prompt where Test.py is saved. This can be done by writing the command "cd \d" and then your path name where the Python module exists e.g. >cd /d D:\Project\MyPython

3. Execute the program by running the command "py Test.py" on the command prompt as shown.



4. You will get the output of the executed program as below.



FAQ:

Q. Why should we use Python language for writing this program, let us say, addition of 2 numbers as compared to machine and assembly language?

If you use machine language to write this program, you have to write instructions in machine language which is difficult and error prone.

If you use assembly language to write this code, you would have to write instructions in assembly language using mnemonics. Number of the instructions in assembly language are less than machine language. However, still, it is tedious job.

If you see Python language program, it is hardly few lines of code as compared to programs written in machine and assembly language instructions. This reduces the chances of doing mistakes while writing a program because the size of the program is small. **The biggest advantage of Python language is platform independent**. It means it can be compiled on different processors and executed. Machine and assembly languages are specific to the machine.

Q. What are the features of the Python?

Structural & Object Oriented: Python supports sequential, selection & iterative control flow. It is possible to create objects in Python. Hence, it is object oriented as well.

Platform independent: Unlike many other programming languages including C and C++ when Python code is compiled, it is not compiled into platform specific machine code, but it is converted into platform independent byte code. This byte code is interpreted by the Python Virtual Machine (PVM) on whichever platform it is being run.

Simple: Python is a simple and easy to learn language because of its clear syntax and readability. That's why it reduces the cost of program maintenance. It is good for starting out because of its simple syntax. Python's syntax are shorter than most other programming languages (Java, C, C++ etc.) If you understand the basic concept of Object oriented programming (OOP), Python would be easy to master.

Secure: Python is secure because the reverse engineering of the .pyc files (byte code) is a lot harder.

Architectural- neutral: Python compiler generates an architecture-neutral byte code file which makes the compiled code to be executable on many processors with the present Python runtime system.

Robust: Programmer has to write few lines of code as compared to other languages like C, C++ for the same program. So, it is easy to maintain and is prone to fewer issues. Python also has the capability to scale easily to solve complex problems. Also, the memory management techniques are in built in Python.

Multi-threaded: With Python's multi-threaded feature it is possible to write programs that can do many tasks simultaneously. This design feature allows developers to construct smoothly running interactive applications.

Interpreted: Python is an Interpreted Language because Python code is executed line by line at a time. Like other language C, C++, java etc there is no need to compile Python code, this makes it easier to debug our code. The source code of Python is converted into bytecode.

High Performance: With the use of Just-In-Time compilers Python enables high performance.

Dynamic: Python is considered to be more dynamic than C or C++ since it is designed to adapt to an evolving environment. Python programs can carry extensive amount of run-time information that can be used to verify and resolve accesses to objects on run-time.

Q. What is meaning of print function (command)?

The print function prints the specified message to the screen, or other standard output device. The message can be a string, or any other object, the object gets converted into a string before displaying on the screen. You will learn more about print function in the later part of this book.

2.6 Points to Remember

1. Multiline comment can be given by using '''<text>''' triple inverted commas.

2. To make the program add.py more user friendly, you can change the print instruction as below.

print ("The sum is", sum)

Whatever characters we write inside the quotes of the first parameter of the print function gets printed as output. e.g: print ("Enthusiasm is a key to success.") displays

Enthusiasm is a key to success

3. There is no separate data type for short and long numbers in Python. Short & long also belong to integer data type itself.

4. Variable names correspond to locations in the computer's memory. In memory, every variable has assigned a particular address. Every variable has a name, a type, a size and a value. We can declare a variable in Python as shown below.

<variable_name> = <value>

Whenever a new value is placed into a variable, it replaces (and destroys) the previous value.

```
a=10
a=20    #Variable a contains 20 and not 10.
```

5. A Python identifier is a name used to identify a variable, function, class, module or other object. An identifier starts with a letter A to Z or a to z or an underscore (_) followed by zero or more letters, underscores and digits (0 to 9).

6. Python does not allow special characters such as @, $, and % within identifiers. Python is a case sensitive programming language. Thus, Manpower and manpower are two different identifiers in Python.

7. bool(-1) is True in Python.

8. The source code program (e.g. add.py) is stored in a secondary memory. i.e. inside C:\ or any other drive. Hard disk is an example of secondary memory.

9. When compiled file like add.pyc is executed (just click on it), it gets copied into the RAM.

10. The program that copies the executable file from secondary memory to a primary memory (RAM) is called as a loader.

Questions

Q.1 **State true or false.**

i. Python was developed at AT&T Bell Laboratories in 1991.

ii. Algorithm is a set of logical steps to perform a certain task.

iii. It is possible to write a Python program without a main function.

iv. print is a standard command that displays the value stored at that memory location.

v. Function is a group of set of instructions arranged in a sequence to perform certain tasks.

vi. Comment is information written inside the source code that is ignored by a preprocessor when the source file is preprocessed.

vii. Directory is the location on a computer where files and other directories can be stored.

viii. Python language is not a case sensitive language.

ix. Every Python program source file ends with an extension .Python

x. In Python language, it is mandatory to declare variables at the beginning of a block, otherwise, compile time error is generated.

Q.2 **Guess the output of the following programs.**

I.
```
one=1947
 two=191
 subtraction = one- two
 print(subtraction)
```

II.
```
one=1980
two=191
subtraction = one- two
print(subtraction)
```

III.
```
one=1947
two=191.01
subtraction = one- two
print(subtraction)
```

IV.
```
Number1
print(Number1)
```

V.
```
Number1
Number2
Number1=13579
Number2=2468
sum= Number1+ Number2
print(sum)
```

VI.
```
Number1=1.4E-46
print(Number1)
```

Q.3 What is a data type? Give example.

Q.4 Create 3 integer variables that stores 10, 20 and 30 respectively.

Q.5 Explain Python language environment in detail.

Q.6 Write a program that does multiplication of 128 and 3876?
Q.7 What is the basic language of a computer? Why?
Q.8 Who invented Python? Discuss the history of Python.
Q.9 Who is a father of a computer?

Answers:

Q.1 i. False ii. True iii. True iv. True v. True vi. True
vii. True viii. False ix. False x. False

Q2. I. 1756 II. 1789 III. 1755.99
IV. Compilation Error V. Compilation Error VI. 1.4e-46

Appendix

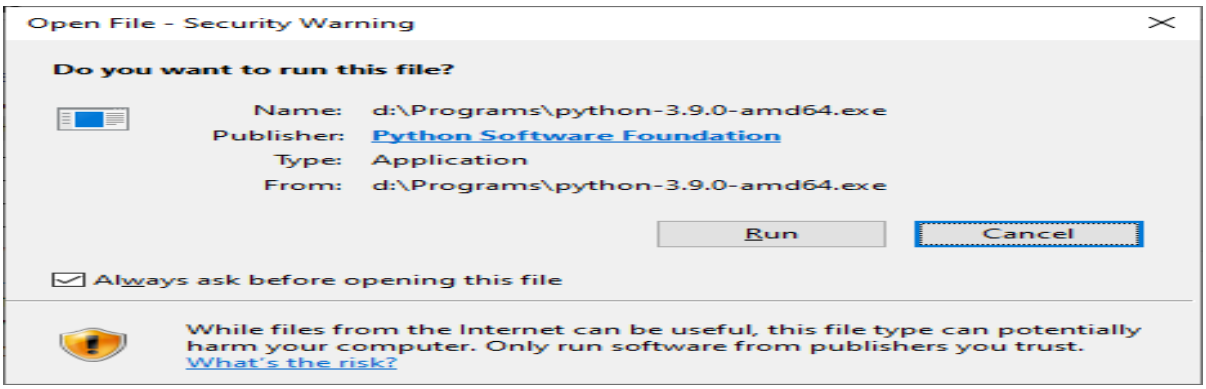
Python is freely available & respective version can be downloaded based on your operating system.

Windows: https://www.python.org/ftp/python/3.9.0/python-3.9.0-amd64.exe
Linux: https://www.python.org/downloads/source/
Mac OSX: https://www.python.org/downloads/mac-osx/

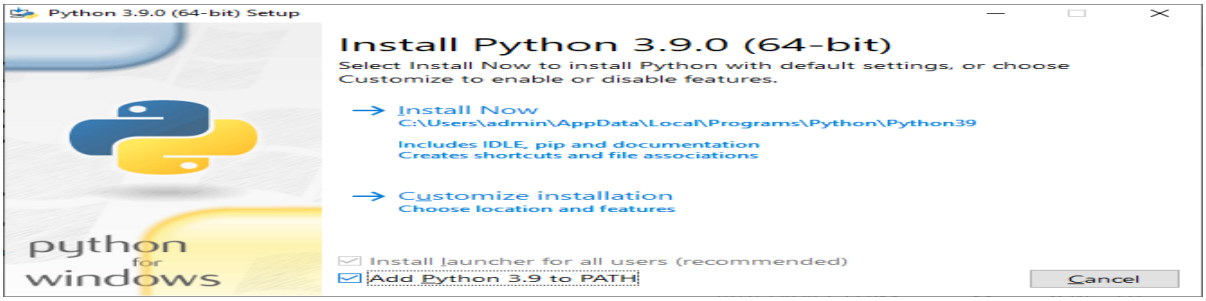
A. Installing Python 3.9 (latest version) on Windows 10

1. Download https://www.python.org/downloads/ latest version as per your

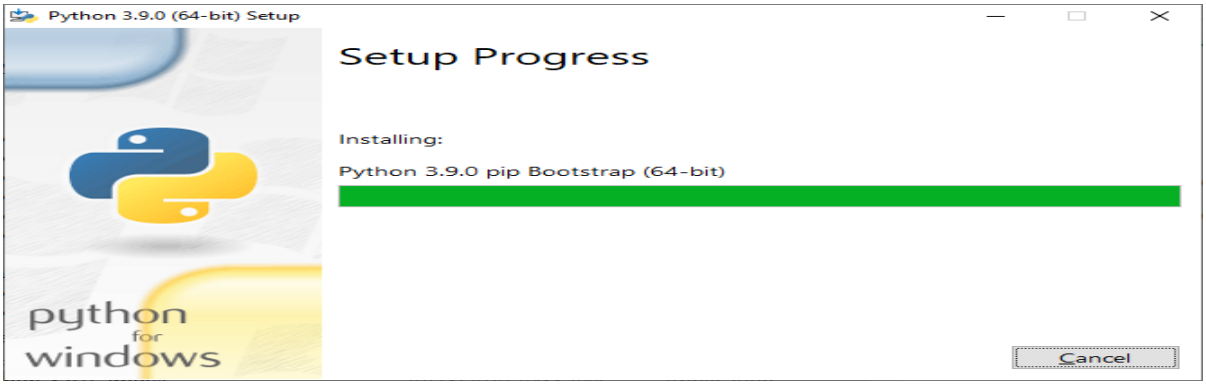
2. You can click on "Run" as shown in the below snapshot.



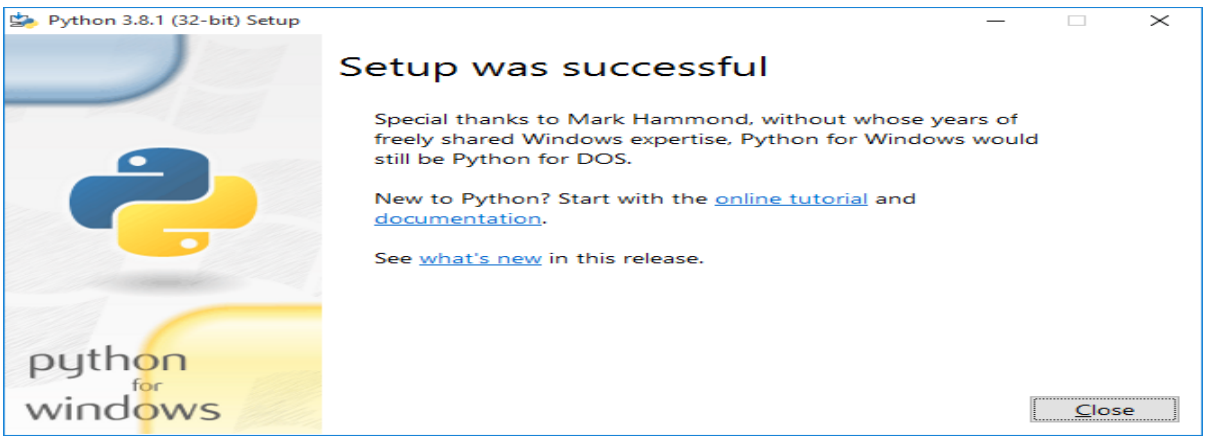
3. Click on “Add Python 3.9 to PATH” & click on “Install Now”



4. “Setup Progress” window would be displayed as shown in the below snapshot.



5. Click on “Close” once you see below window.



You can write Python programs using Notepad, Textpad or sophisticated Integrated Development environments like Eclipse can be used. However, beginner should use notepad.

Chapter

3

STRUCTURED PROGRAMMING: SEQUENTIAL AND SELECTION CONTROL FLOW

Many of you might be wondering why there is a structured programming chapter in this book because we have heard that Python is an object-oriented language[1]. Yes, Python is an object-oriented programming language, but Python also has structured programming features. Without these structured programming features, it is highly difficult to write any Python program. Let us study structured programming features like sequential control flow, selection control flow, iteration control flow etc.

3.1 Sequential Control Flow

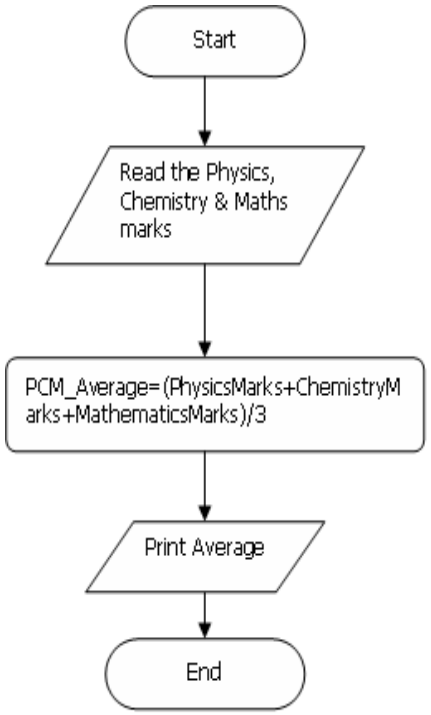
Imagine a hypothetical situation! Your lecturer friend approaches you one fine evening. He can use a computer but cannot do programming. He wants to find average of Physics, Chemistry and Mathematics marks of around 48 students. Further, he must do this task many times and hence requests you to write a Python program. The algorithm and flow chart for this program is given as below.

Let's convert every step in the above algorithm into equivalent Python language instruction to write a complete Python program.

[1] Object oriented programming will be discussed in the later part of this book.

Algorithm:

1. Computer should display the message.
   "Enter Physics Marks?" on the output device.
2. Once the user enters the Physics marks through keyboard, computer should store it in a memory
3. Computer should display the message.
   "Enter Chemistry Marks?" on the output device.
4. Once the user enters the Chemistry marks through keyboard, computer should store it in a memory.
5. Computer should display the message.
   "Enter Mathematics Marks?" on the output device.
6. Once the user enters the Mathematics marks through keyboard, computer should store it in a memory.
7. Averages of stored marks can be calculated as
   PCM Average= (Physics Marks+ Chemistry Marks + Mathematics Marks) /3
8. After calculating average marks, computer should display a message.
   "Average of your marks in Physics, Chemistry and Mathematics is ….."



Step 1: Computer should display the message "Enter Physics Marks?" on the output device.

We have studied in the second chapter that print is a standard command (function) that displays the data on the output device like monitor. The instruction to display "Enter Physics Marks?" is as below.

```
print("Enter Physics Marks?")
```

Step 2: Once user enters Physics marks through keyboard, computer should store it in a memory.

Computer receives input data through input devices like a keyboard. Therefore, you must enter the Physics Marks (input data) through a keyboard. But what is the command/instruction (function) that can read data entered through keyboard and stores it in a computer memory? input() command (function) which is in-built in Python is used

```
PhysicsMarks =input()
```

Step 3-4: Similarly, we can write following instructions for receiving Chemistry marks

```
ChemistryMarks = input()
```

Step 5-6: Similarly, we can write following instructions for receiving Mathematics marks.

```
MathsMarks =input()
```

```
PhysicsMarks = input()
ChemistryMarks = input()
MathsMarks = input()
```

Step 7: Averages of stored marks can be calculated as
PCM Average = (Physics Marks + Chemistry Marks + Mathematics Marks) /3

7th statement in the algorithm can be

PCM_Average = (PhysicsMarks + ChemistryMarks + MathematicsMarks) / 3

When we use computer, multiplication (×) & division (÷) is denoted by * and / symbol respectively.

Step 8: After calculating average marks, computer should display a message "Average of your marks in Physics, Chemistry and Mathematics is …"

print("Average of Physics, Chemistry and Mathematics marks is= ", PCM_Average)

The program can be written as shown below.

```
print("Please enter your Physics marks:")
PhysicsMarks = input()

print("Please enter your Chemistry marks:")
ChemistryMarks = input()

print("Please enter yourMathematics marks:")
MathsMarks = input()

PCM_Average = (PhysicsMarks+ChemistryMarks+MathsMarks)/3
print ("Average of Physics, Chemistry and Mathematics marks is: ", PCM_Average)
```

Oh! The above program gives an error. What can be the reason? The input function (command) accepts input data from user and saves it as a *String*. In the above program, data entered by user as input is stored as a string. **At line 7, attempt has been done to add these strings and divide it by 3. Dividing a string is not valid operation in Python.** To resolve this, we must restrict that the data entered by the user is valid. As marks of a subject can only be an integer, it is necessary to restrict user input values to be only integer in case of this program.
int(input()) instruction receives only integer values. If user enters any value other than integer (e.g. decimal), it does not accept it and gives error.

This program 3.1 resolves the above issues.

Program 3.1

```
print("Please enter your Physics marks:")
PhysicsMarks = int(input())

print("Please enter your Chemistry marks:")
ChemistryMarks = int(input())

print("Please enter yourMathematics marks:")
MathsMarks = int(input())
PCM_Average = (PhysicsMarks+ChemistryMarks+MathsMarks)/3
print ("Average of Physics, Chemistry and Mathematics marks is: ", PCM_Average)
```

When instructions in a program are executed one after another i.e. in a sequential manner, it is known as a sequential control flow.

It is one of the features of a structured programming. The program 3.1 is an example of a sequential control flow as all the instructions in this program are executed sequentially.

Q. What is a control flow? What are the different control flows used in a program?

Control flow is a flow of the execution of the instructions in a program.

OR

It is the order in which the instructions in a program are executed.

In this chapter, you will understand, the instructions are not always executed in the order they are written in a program. By default, the flow of instructions is sequential. A control statement disrupts the sequential flow. It means certain instruction or instructions are skipped during the execution of a program.

Based on the order of execution of instructions, Python programs are mainly written in 4 different structure formats.

1) **Sequential:** Statements in a program are executed one after another i.e. in a sequential manner.
2) **Selection:** Statements in a program are executed based on certain condition(s) defined in a program. e.g. *if; if-else*.
3) **Iteration:** Statements in a program are executed repeatedly. e.g. *while, for*
4) **Jump:** Different set of instructions in the program are executed as per the jump defined in a program. e.g. *break, return, continue*.

Do not get confused after reading these 4 different types of control flows. We are going to study it in this chapter in a simplified and understandable language.

3.2 Selection Control Flow

3.2.1 if-else statement

Assume that one of your friends says that he has got 91 marks in Physics and 99 marks in Mathematics. Instantly, if you think in which subject your friend has got highest marks? Your brain will compare 91 marks to 99 marks and conclude that Mathematics marks are greater than Physics marks. Will it again compare 99 marks to 91? No. It will do only 1 comparison instead of 2 comparisons because based on the result of only one comparison, result can be concluded in this case.

When one statement out of given 2 statements is executed based on the result of a condition, if-else statement is used. If the condition is true, statement associated with if-part is evaluated, otherwise, statement associated with else part is evaluated.

The general form of *if-else* statement

```
if (expression):
    Statement1 #Statement associated with if part
else:
    Statement2 #Statement associated with else part
```

If expression is true, statement1 is executed. Otherwise, statement2 is executed.

Q. What is an expression?

Expression is a combination of an operator and operand that evaluates to a constant value.

OR

Expression can be any combination of variable, operator and / or constant that results into a constant value[1] after its evaluation.

Operand can be a constant value like 6, 17 etc. or a variable x, i etc. **The evaluation of an expression results into a constant value.**

Few examples of expressions are a + b; x*y+3; 8; c etc. where x, y, a, b, c are variables.

Let's understand concept of *if-else* statement by writing a small program that accepts Physics and Mathematics marks as input from a keyboard and checks in which subject the student has obtained more marks as compared to the other subjects. It is assumed that student has not obtained same marks in Physics and Mathematics.

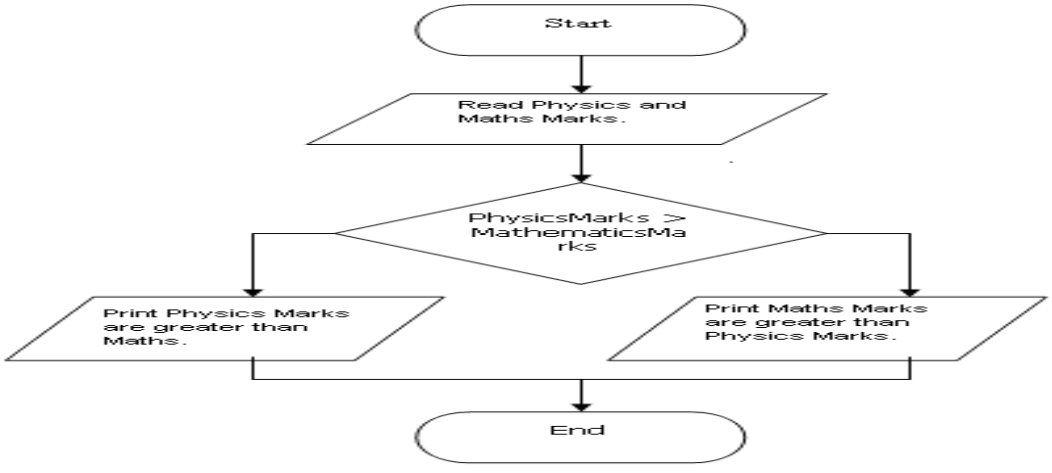
Following are the steps involved (algorithm) to do this task.

1. Computer should display the message "Enter Physics Marks?" on the output device.
2. Once user enters the Physics marks through keyboard, computer should store it in a memory.
3. Computer should display the message "Enter Mathematics Marks?" on the output device.
4. Once user enters the Mathematics marks through keyboard, computer should store it in a memory.
5. *if* Physics marks are greater than Mathematics mark
   Display "Physics marks are greater than Mathematics marks."
   else
   Display "Mathematics marks are greater than Physics marks."

You can easily write instructions for step 1 to step 4. If you require to understand step 1- step 4, you can refer program 3.1. The main logic to write this program is to write step 5. In step 5, only one condition needs to be evaluated out of 2 conditions, hence, we can use if-else statement.

if-else statement for step 5 can be written as below.

```
if ( PhysicsMarks > MathematicsMarks ):
    print ("Physics marks are greater than Mathematics marks.")
else:
    print ("Mathematics marks are greater than Physics marks.")
```



Program 3.2:

```
PhysicsMarks = int(input("Please enter your Physics marks: "))
MathematicsMarks = int(input("Please enter your Mathematics  marks: "))
if PhysicsMarks>MathematicsMarks :
    print ("Physics marks are greater than Mathematics marks")
else:
    print ("Mathematics marks are greater than Physics marks")
```

3.2.2 if statement

What if someone tells you equal marks in both these subjects? Here, we need to check only one condition i.e. whether the marks received in both the subjects are equal.

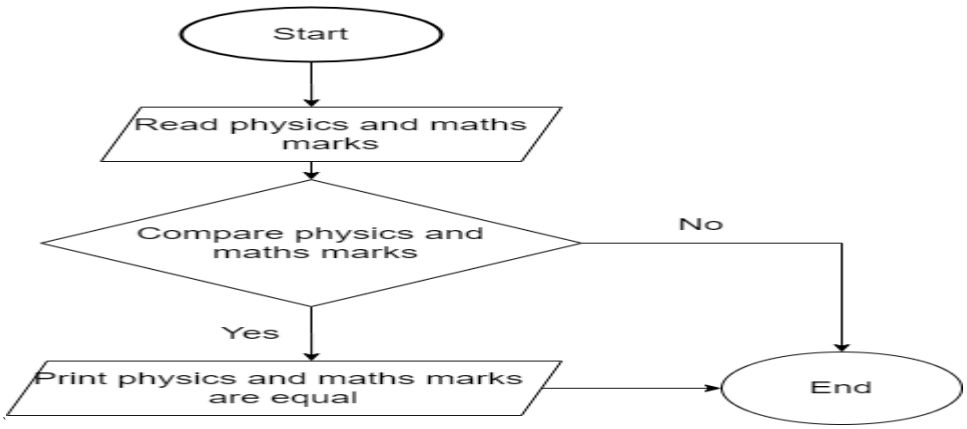
When given statement is executed based on the result of a condition, if statement is used.
If the condition is true, statement associated with if statement gets executed. If the condition is false, statement associated with if is not executed.

The general form of if statement is as below.

if (expression):
    statement

Let's write a program that accepts Physics and Mathematics marks as input, checks whether the Physics marks are equal to Mathematics marks and display the below message if this condition is true.

"Physics marks are exactly equal to Mathematics marks."



If this condition is false, it should not display anything.

Program 3.3:

```
PhysicsMarks = int(input("Please enter your Physics marks: "))
MathsMarks = int(input("Please enter your Mathematics marks: "))
if PhysicsMarks==MathsMarks:
    print ("Physics marks are exactly equal to Mathematics marks")
```

Remember = is an assignment operator, while, == is an equality operator.
Equality operator compares values of its operands. It returns true if the values are equal, otherwise false. If you forget to write one = symbol in equality operator, it becomes assignment operator.

Let's write a program that compares Physics and Mathematics marks and displays following message

If Physics marks are exactly equal to Mathematics marks, display the message
"Physics marks are exactly equal to Mathematics marks."
If Physics marks are greater than Mathematics marks, display the message
"Physics marks are greater than Mathematics marks."
If Mathematics marks are greater than Physics marks, display the message
"Mathematics marks are greater than Physics marks."

Program 3.4:

```
PhysicsMarks = int(input("Please enter your Physics marks: "))
MathsMarks = int(input("Please enter your Mathematics marks: "))
if PhysicsMarks==MathsMarks:
    print ("Physics marks are exactly equal to Mathematics marks")
if PhysicsMarks>MathsMarks:
    print ("Physics marks are greater than Mathematics marks")
else:
    print ("Mathematics marks are greater than Physics marks")
```

Can you guess the output of the program 3.4 if the marks of Physics and Mathematics are 98? The output will be

Physics marks are exactly equal to Mathematics marks.
Mathematics marks are greater than Physics marks.

It means there is a logical error in the above program. What is the Python language construct used when there are multiple conditions need to be evaluated and the result is only 1?

Python language provides ***if - elif - else*** construct when one condition is true from given set of conditions. The general form of *if-elif* ladder is as below.

```
if(expression1):
      Statement1
elif(expression2):
     Statement2
…
else:
    Statements
```

The above program can be written like as shown in Program 3.5.

Program 3.5:

```
PhysicsMarks = int(input("Please enter your Physics marks: "))
MathsMarks = int(input("Please enter your Mathematics marks: "))

if PhysicsMarks==MathsMarks:
    print ("Physics marks are exactly equal to Mathematics marks")
elif PhysicsMarks>MathsMarks:
    print ("Physics marks are greater than Mathematics marks")
else:
  print ("Mathematics marks are greater than Physics marks")
```

if can be used only at the beginning and *else* only at the end. *if* is mandatory, while, *else* is optional.

Q. Guess the output of the following program.

I

```
k=50
if(k>=0):
    print (“Kolhapur")
else:
     print("Pune")
```

Output:
Kolhapur

II

```
k=5
if(k=5):
    print(“A”)
else:
    print(“B”)
```

Output:
Syntax error
Condition must evaluate to Boolean value i.e. *True* or *False*. Hence, syntax error.

III

```
k=5
if(k ==5):
    print (“A”)
else:
    print(“B”)
```

Output:
A
As, k is equal to 5, condition k==5 results into true. Hence, the output.

IV

```
p=10; c=10; m=10 # Multiple variables can be defined separated by semicolon.
if ((p == c) == False)):
    print("M")
else:
    print("N")
```

Output:
N
(p==c) results into True which is compared to False. Hence, the condition results into False.

Pseudo code and Flowchart:

Pseudo code is an artificial, informal, and English like language that helps us to develop a program. It is structured English for describing algorithms. It breaks the bigger tasks into smaller tasks. It is not executed on the computers. Pseudo code helps to write actual program.

Flowchart is a graphical representation of an algorithm. It has single entry and single exit. It is drawn using certain special-purpose symbols connected by arrows called flow lines.

Following table describes the various symbols used in creating a flowchart.

Symbol	Description
Rectangular symbol	Processing block like initialization, assignment and expression evaluation
Arrow symbol	Connects one symbol to others or describe the flow.
Oval symbol	Indicates the beginning or end of a program or a section of code.
Diamond symbol	Used to take decisions (condition evaluation)
Parallelogram symbol	Used to display input, output.

Let's develop a flow chart for a program that accepts Physics, Chemistry and Mathematics marks as input and displays the maximum marks obtained as output. Assume that marks received as input are different in each subject.

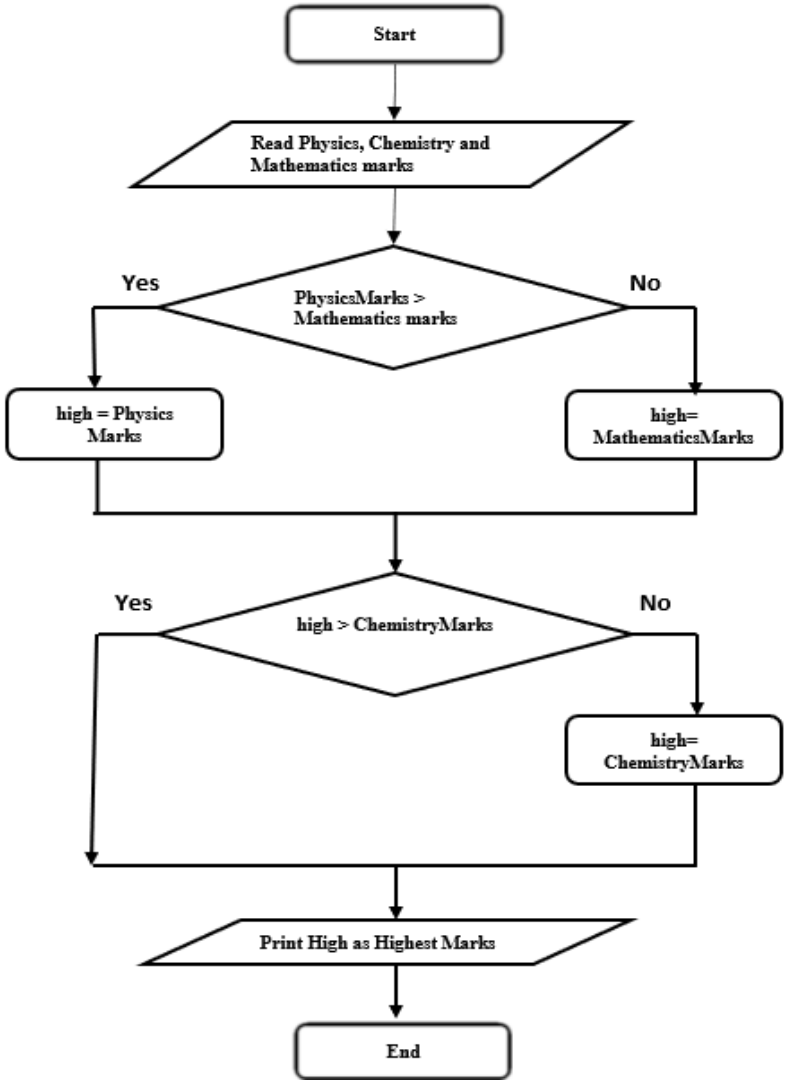


Fig.3.4 Flowchart that finds highest marks among Physics, Chemistry and Mathematics

3.2.3 Conditional Operator

Let's discuss operators before we discuss conditional operator (.

Operator operates on the data. Unary, binary, ternary operators have 1,2 and 3 operands respectively. e.g. & is a unary operator, = is a binary operator. The complete list of operators is given in the appendix.

Note that + is a unary as well as binary operator. k=+10 Here, + acts as a unary operator as it indicates the sign of 10 is positive. int k=10+20 Here, + adds 2 operands 10,20 & acts binary operator.

The general form of ternary (conditional) operator is as below.

[on_true] ***if*** **[expression]** ***else*** **[on_false]**

It is the abbreviated form (shorthand) to write *if-else* conditional statement. If expression is true (non-zero value), then on_true will be executed else on_false will be executed.

Example 1:

Let's write a program 3.2 using conditional operator as below.

Following program accepts Physics and Mathematics marks as input from a keyboard and displays subject name in which the student has obtained more marks as compared to the other subject. It is assumed that student has not obtained same marks in Physics and Mathematics.

Program 3.12:

```
PhysicsMarks=int(input( "Enter Physics marks "))
MathematicsMarks =int(input("Enter Mathematics marks "))

print("Physics") if (PhysicsMarks > MathematicsMarks) else print("Mathematics")
```

Example 2:

Following program accepts integer input and prints whether the given number is even or odd.

Program 3.13:

```
num = int(input("Please enter any integer "))
print("even") if (num%2)==0 else print("odd")
```

Output:

XYZ

The beauty of conditional operator is that we can write the convertible *if-else* statements in a single line instead of 4 lines as shown in the above examples. However, generally conditional operator (ternary operator) is not used because it's difficult to understand and maintain the code.

Q. Guess the output of the following program.

```
i=5
 j=0
 k=6

 i=i+1
 j=i+k
 k=k-1

 k=j
 j=j-1
print("i=" +str(i)+  " j=" +str(j)+  " k=" +str(k))
```

Output:
i=6 j=11 k=12

II.

```
 i=True
 i=i-1
 if ( i ):
     print ("A")
 else:
     print("B")
```

Output:
B
Boolean variable True is treated as 1 in Python. Hence, i-1=1-1=0. Hence, the output is B.

3.3 Operator Precedence

Can you guess the output of the following program?

```
i = 7
j = 3
k = 8
m = i*j+k
print(m)
```

You may feel the output is (7*3 + 8) = 29 or 7 * (3 + 8) = 77. Oh, then, what is the correct output? In the first evaluation 7 and 3 are identified as operands of *, while, in the second evaluation, 7 and (3+8) are identified as operands of operator *. Hence, there should be certain rules to evaluate such expressions to avoid calculation ambiguity.

Certain rules are used while computing such ambiguous expressions in Python. Multiplication operator has higher priority than +. i.e. Operands of * operator are determined before determining operands of +. Hence, we can rewrite the above expression as

```
m=7*3+  8
```

In this case, * operator has higher priority than +. Hence, 7*3 is calculated first. i.e. 21 is added to 8 i.e. 29.You need to refer the Operator precedence table given in the appendix to evaluate such expressions.

Now, let's evaluate following expression.

```
m=7*3+  8 / 2
```

It is seen that * and division / operator are used in above expression. Further, it is seen from the table that some operators like * and / have same priority. Then, what should be the order of evaluation? When the operators of same precedence are present in an expression, evaluation order is decided by the *associativity*. Associativity decides the order of the evaluation. i.e. always from left to right in Python.

Let's evaluate this expression. m=7*3+ 8 /2

Associativity comes in picture when same precedence operators are present in an expression. As per associativity rules, the same precedence operators are always evaluated from left to right. Hence,

```
m=(7*3)+  8 /2
m=21+  8 /2
m=21+ ( 8 /2)
m=21+ 4
m=25
```

Associativity of operator decides which operator in the expression should be given higher priority when two or more operators of same priority are present in an expression.

Operators * and / have same precedence and their associativity is from left to right. It means the operands of the operator which is present at the left of the expression are determined first and so on.

Q. Guess the output of the following Program.

```
PhysicsMarks = 95
ChemistryMarks =99
MathsMarks = 99
PCM_Average = PhysicsMarks+ChemistryMarks+MathsMarks/3
print ("Average of Physics,Chemistry and Mathematics marks are: ", PCM_Average)
```

Output:

227.00

As division operation has higher priority than addition, operands close to division operator are bound tightly in the parentheses as shown below. It means the operands of the operator division / are determined first. Hence, the correct output is

```
PCM_Average=Mathematics+ Chemistry+MathematicsMarks/3
           =95+99+(99/3.0)
           =95+99+33.00
           =227.00
```

If you use parentheses as shown below, you will get the output 97.666%.

PCM_Average= (Mathematics+ Chemistry+ Mathematics Marks) / 3.0
= **(95+99+99)/3.0**
=**293**/3.0
=**97.666%**

Let's try to guess the output of the following program.

```
x=3.0/6*8
print("x=" , x)
```

Hence, the above expression is evaluated as x= (3.0/6)*8 which results into 4.

Q. Guess the output of the following program.

I.

```
x =2*3+4/5-6*7
print ("x=" +str(x))
```

Output:
x= -35.2
Hope, you understand the following evaluation very easily.

X = 2*3 + (4/5)-6*7
= (2*3) + 0.8 - (6*7)
= (6+0.8)- 42
= -35.2

II.

```
a=1
b=2
c=3
d=4
a=b=d=10-c
print ("a=", a)
```

Output:
a=7

a=b=d=(10-c)
a=b=(d=(10-c))
a=(b=(d=(10-c)))

III.

```
a=1
b=2
c=3
d=4
a -= b*c
d=10
print  ("a=" +str(a))
```

Output:
a= -5

3.4 Logical Operators

Logical expression is that which contains logical operator(s). Logical expression results into a true or false value. Any non-zero value is treated as true, while other values are treated as false. *AND, OR, NOT* are logical operators.

Let's understand these operators one by one.

```
k= (1<9) and (51>12) # k is true.
```

In case of *and* operator, both the operands must be true to get the evaluation of expression true.

Input 1	Input 2	Output
True	True	True
True	False	False
False	True	False
False	False	False

Operand 1 is evaluated first. If it is true, then only operand 2 is evaluated.

```
k= (7==7) and (108>98) # true. Both the operands are evaluated
k=false and (6>2)  # false  Operand 2 is not evaluated i.e. (6>2)
k=(6>2) and (0 ==1)  # false Both the operands are evaluated
k= (7>7) and (7!=7)  # false Operand 2 is not evaluated i.e. (7!=7)
```

Let's write a program that accepts 3 integers (say num1, num2 and num3) from user and displays the highest number. The logic of this program is as below.

1. Accept 3 integer numbers from user.
2. *if* (num1 is greater than num2 and num1 is greater than num3)
   Display num1 as highest number
   else if (num2 is greater than num1 and num2 is greater than num3)
   Display num2 as highest number
   else
   Display num3 as highest number.

Program 3.16:

```
num1= int(input("enter num1: "))
num2= int(input("enter num2: "))
num3= int(input("enter num3: "))
if (num1 >num2) and  (num1>num3):  #suite 1
    print  ("Highest number= " + num1)            #suite 2
elif (num2 >num1) and (num2>num3): #suite 1
    print ("Highest number= " + num2)              #suite 3
else :                                             #suite 1
    print ("Highest number= " + num3)              #suite 4
```

In case of logical *or* operator, either one of the operands must be true to get the evaluation of expression true.

Input 1	Input 2	Output
True	True	True
True	False	True
False	True	True
False	False	False

In order to evaluate conditional expression in the if statement, we need to use logical or operator. The evaluation of logical expression is true if either of the operand of logical or operator is true.

```
k=  ( 1 = = 1) or ( 6 > 3 )   # true. Operand 2 is not evaluated i.e. (6>3)
k= (1 ! = 1 ) or ( 6  > 3 )   # true Operand 2 is  evaluated i.e. (6>3)
k= ( 6 > 3)   or  (1 ! = 1 ) # true Operand 2 is not evaluated i.e. (1!=1)
k=  ( 1 ! = 1 ) or ( 1 ! = 1 ) # false Operand 2 is evaluated i.e. (6>2)
```

Let's write a program that accepts input number from user and prints the message.
"Number entered is either divisible by 3 or 5."
The algorithm is simple

1. Accept input number from user.
2. *if* ((number is divisible by 3) or (number is divisible by 5))
   Display "Number entered is either divisible by 3 or 5."
3. "Number entered is either divisible by 3 or 5." can be written as below

Program3.17:

```
num= int(input("Enter the number: "))
if (num%3==0) or (num%5==0):
    print ("Number entered is either divisible by 3 or 5.")
```

Let's modify program 3.17 so that it should print following message.
"Number entered is either not divisible by 3 or 5."

It means we need to check the number entered is either not divisible by 3 or 5. (num%3==0) expression is true when number is divisible by 3. When number is not divisible by 3, we need to invert the result.

not operator operates on a single value. If the value is true, it inverts it to false and vice versa.

Input	Output
True	False
False	True

```
k=not 5; # k is false
k=not 0; # k is true
k=not -1; # k is false
```

Following program accepts integer from user and prints the following message.
"Number entered is either not divisible by 3 or 5."

Program 3.18:

```
num= int(input("Enter the number: "))
if not ( (num%3==0) or (num%5==0)):
    print ("Number entered is either not divisible by 3 or 5.")  #indented 3rd statement
```

We have started using indentation from Program 3.2. Let me explain the concept of indentation with the example of program 3.18. There are blank spaces at the beginning of " #indented 3rd statement:" Python uses indentation instead of curly braces to indicate group of instructions or instructions block. The statements with the same indentation belong to the same group called a *suite*. In the program 3.16, suites are shown for your understanding.

3.5 Point to Remember

1. *if*, *else*, *elif*, *and*, *or*, *not* are keywords in Python.
2. In Selection control flow Statements in a program are executed based on certain condition(s) defined in a program. e.g *if*, *if-else*
3. When one statement out of given 2 statements is executed based on the result of a condition then *if-else* statement is used.
4. Expression is a combination of an operator and operand that evaluates to a constant value.
5. Conditional operator can be nested**.**
6. Operators having same precedence are evaluated according to their associativity
7. Logical expression is that which contains logical operator(s). *and*, *or*, *not* are logical operators.
8. Logical expression results into a true or false value.
9. Any non-zero value is treated as true, while other values are treated as false.

3.6 Deep Knowledge Section

Q.1 Guess the output of the following program

I.

```
num1=11
num2=98
num3=78
if (num1>num2):
    print ("One")
if (num2>num3):
    print ("Two")
else:
    print ("Three")
```

Output:
Two

The ambiguity to evaluate statements in the above program is resolved as follows.

Associate *else* with the closest previous *if*.

else is attached with the closest else-less if. Hence,

```
if(num2>num3):
  print ("Two")
else:
  print("Three")
```

is treated as separately.

II.

```
num1=11
num2=98
num3=78
if (num1<num2):
#start of block statement
    print ("One")
if (num2>num3):
    print ("Two")
else:
     print ("Three")
# End of block statement
```

Output:

One
Two

Block statement is a statement that contain none or many statements. The above used block statement contains print method call as first statement and *if-else* statement as a second statement.1.

Questions

Q.1 State true or false.

i. In sequential control flow, instructions are executed one after another.

ii. When numerator and denominator are integer values the result is always integer.

iii. In selection control flow Based on certain condition defined in the program, statements in a program are executed.

iv. Expression can be any combination of variable, operator and/or constant

v. Any non-zero value in Python is treated as true, while, 0 is treated as false.

vi. Flowchart is a graphical representation of an algorithm

vii. No expression is optional in conditional operator.

viii. Operator precedence decides the operands of an operator and not the evaluation order.

ix. Negative value in Python is treated as false.

Q.2 Guess the output of the following program.

i.
```
k=0
if(k>=0):
    print("Mr. Sangram Kendre is very active and smart.")
else:
    print("Mr. Govind and Atul are ever smiling.")
```

ii.
```
k=2
if(k-2==k-2):
    print("Balasaheb Kushappa")
else:
    print("Nashik")
```

iii.
```
k=2
if(k=k-2):
    print("Calcutta")
else:
    print("Madras")
```

iv.
```
k=2
if(k==k-2):
    print("Shrihari")
else:
    print("Akshay")
```

v.
```
k=2
if(k-k):
    print("Govinda.")
else:
    print("Vittal.")
```

vi.
```
k=0
if(k++):
    print("Ajay Kumbhar.")
else:
    print("Nikhil Delekar")
```

vii.
```
k=0
if(++k):
    print("Gayatri Bahirat.")
 else:
    print("Shravani Bahirat.")
```

viii.
```
m=0
i=7
j=3
k=8
n=5
m=i*j +  k + n
print("m=" ,m)
```

Q.3 What is a control flow? Discuss various types of control flows with examples?

Q.4 Which control flows are features of structured programming?

Q.5 Which control flow is not a feature of structured programming?

Q.6 What is a side effect in case of expression evaluation?

Q.7 Compare the followings.

i. structured and unstructured programming

ii. if-elif

Q.8 Which expression is mandatory in case of *while* and *do-while* loop?

Q.9 Write a program that finds the average of 4 decimal values supplied by a user?

Q.10 Write a program that determines number supplied by a user is even or odd?

Q.11 Write a program using ***if-elif*** construct that checks the number supplied by a user is completely divisible by 2 , 4 and 16?

Q.12 A student will not be allowed to sit in exam if his/her attendence is less than 75%.Take following input from user:
Number of classes held, Number of classes attended and print percentage of class attended.Then check whether student is allowed to sit in exam or not.?

Q.13 Modify the above question to allow student to sit if he/she has medical cause. Ask user if he/she has medical cause or not ('Y' or 'N') and print accordingly.

Answers:

Q1.

i. True	**ii.** True	**iii.** True
iv. True	**v.** True	**vi.** True
vii. True	**viii.** True	**ix.** False

Q2.

i. Mr. Sangram Kendre is very active and smart.	**ii.** Balasaheb Kushappa	**iii.** Compilation error
iv. Akshay	**v.** Vittal	**vi.** Compilation Error
vii. Shravani Bahirat.	**viii.** m=34	

Operator Precedence Table

Precedence	Operator	Meaning	Associativity
12	()	Parentheses	Left to Right
11	**	Exponent	Left to Right
10	*, /, //, %	Multiplication, Division, Floor division, Modulus	Left to Right
9	+, -	Addition, Subtraction	Left to right
8	<<, >>	Bitwise shift operators	Left to right
7	&	Bitwise AND	Left to right
6	^	Bitwise XOR	Left to right
5	|	Bitwise OR	Left to right
4	==, !=, >, >=, <, <=, is, is not, in, not in	Comparisons, Identity, Membership operators	Left to right
3	not	Logical NOT	Left to right
2	and	Logical AND	Left to right
1	or	Logical OR	Left to right

Chapter

4

STRUCTURED PROGRAMMING: ITERATION CONTROL FLOW

If someone asks you the summation of numbers starting from 0 to 1000, how much time will you require to calculate it? Will you able to calculate it quickly & that too without mistake? Computer can do this task of summation in fraction of a second & that too without any calculation mistake.

Human beings have certain limitations where computer's use is highly beneficial. One of the main benefits of a computer is to perform repetitive task without doing any mistakes. This chapter discusses how one can write Python language loops (instructions) to perform repetitive tasks.

4.1 *while* loop

Let's write a program that displays five consecutive numbers (series of numbers) starting from 12 and the difference between two consecutive numbers should be 5. It should not include the starting number 12. i.e. Program should print output as 17,22,27,32,37.

The steps involved to do this task manually are as below.

1. Note the starting number as 12 and the difference between two consecutive numbers is 5.
   num = 12 diff = 5
2. Add the difference number "diff" to "num", store it into "num" and display the new number.
3. Add the difference number "diff" to "num", store it into "num" and display the new number.
4. Add the difference number "diff" to "num", store it into "num" and display the new number.
5. Add the difference number "diff" to "num", store it into "num" and display the new number.
6. Add the difference number "diff" to "num", store it into "num" and display the new number.

The complete program can be written as shown below.

Program 4.1:

```
num = 12
diff = 5

num = num+diff                    # Repetitive instruction
print ("Number = ", num)          # Repetitive instruction

num = num+diff
print ("Number = ", num)
num = num+diff
```

```
print ("Number = ", num)

num = num+diff
print ("Number = ", num)

num = num+diff
print ("Number = ", num)
```

Output:

```
17
22
27
32
37
```

Q. What is the meaning of the instruction num= num + diff?

In Mathematics, num = num + diff is not correct. In case of programming, right hand expression of equal to (=) operator is evaluated first and evaluated result is stored at the left-hand side variable.

In case of instruction 1, right hand side of expression i.e. num i.e. 12 is added to diff i.e. 5 in this case & stored in left hand side variable num i.e. 17. Previous value present in num is overwritten.

Left hand side of assignment operator (=) must have variable name & not any constant. Instructions *sum+3=sum*4* or *4=sum* etc. are not allowed as left side of = must have only unique variable name.

Python being a concise language, it allows to express num = num +diff like num +=diff.
+= is known as "addition assignment" operator.

The program 4.1 displays five consecutive numbers starting from 12 (excluding 12) and the difference between two consecutive numbers is 5. *Suppose your friend asks you to display 1000 consecutive numbers in this series, how can you enhance the above program?*

You can observe that there are repetitive instructions as shown by marked lines. You have to repeat this repetitive instruction(s) to calculate next number in the series of these numbers. As you want to display 1000 consecutive numbers in this series, you have to copy and paste the below instruction 995 times in the above program as 5 times repetitive instructions are already in this 4.1 program.

```
num=num+diff
print("Number=" +num)
```

But, is it a practical solution? There can be a mistake in writing such lengthy programs. Instead of copying these repetitive instructions 995 times, you may copy it less or more than 995 times by mistake.

Is there any way in Python by which one can write repetitive instruction(s) ***once*** *and execute it specified number of times?* Yes, Python provides loop programming construct that executes repetitive instruction(s) for the specified number of times.

Can I write following repetitive instructions once and execute it 1000 times? Yes.

```
num=num+diff
print("Number=" +num)
```

Loop is a programming construct by which a computer can execute set of instruction(s) repetitively for a specified number of times.

We should know following information to write a loop.

- Repetitive set of instruction(s)
- Number of times the loop is executed
- Initial values of the variables used in the repetitive set of instruction(s)

In case of the program 4.1, information is as below.

Repetitive instruction(s)	num=num+diff print("Number=" +num)
Number of times the loop is executed	5
Initial values of the variables used in the repetitive set of instruction(s)	num=12, diff=5

In order to count number of the times the loop is executed, there should be a variable in a program that counts the number of times executions of the repetitive instruction(s)[1] . This variable is known as a loop counter i.e. loop counter counts the repetition of the loop.

General form of a *while* loop:

while loop executes a set of instruction(s) repetitively until the conditional expression is true.

while <conditional expression>
    <Repetitive statement(s)> #

In case of the program 4.1, we can initialize the value of the loop_counter to 0 at the start of the loop. When the repetitive set of instruction(s) gets executed once, loop_counter should get incremented by 1. When the value of the loop_counter becomes 5, the loop should stop executing and the next statement in the program should get executed.

Program 4.2:

```
num = 12
diff = 5

loop_counter=0                      # loop counter is initiliased by value 0

while  loop_counter<5 :
   num = num+diff                   # Repetitive instruction
   print ("Number = ", num)          # Repetitive instruction
   loop_counter= loop_counter+1     # loop counter is incremented by 1
```

[1] Instruction(s) means instruction or instructions

Advantages of loops:

1) The instruction(s) written by using loop construct are compact (small in size).
2) It's easy to modify, enhance and maintain the programs that is written using loop construct.
3) Chances of making mistakes in writing a repetitive set of instruction(s) repetitively like a sequential control flow are more. In case of loops, repetitive set of instruction(s) are written only once, whereas, sequential control flow where repetitive set of instruction(s) are written more than once.
4) Suppose if the number of times of repetition of the instruction(s) set is changed, you have to add repetitive instruction(s) set or delete already written repetitive instruction(s) set(s) in case of a sequential control flow. In case of loops, you have to change the conditional expression. e.g. If tomorrow you want such 10000 numbers, you can easily do underlined change like this

Program 4.2 (Modified)

```
num = 12
diff = 5
loop_counter=1
while  loop_counter<10000 :   # Minor change
   num = num+diff
   print ("Number = “, num)
   loop_counter= loop_counter+1
```

4.2 *for* loop

Like *while* loop, it is possible to use *for* loop.

General form of a for loop:

for value in *sequence*:
    Repetitive instruction(s)

where
sequence: a list or a string.
value: it is a variable that stores the value of current item from the sequence.

For the above 4.2 program,

Sequence / list: [0,1,2,3,4]
Repetitive Statement:
    num=num+diff
    print(“Number=” +num)

Hence, the program 4.1 can be written using for loop as below.

Program 4.2:

```
num = 12
diff = 5

for i in [0,1,2,3,4]:
   num = num+diff                  # Repetitive instruction
   print ("Number = “, num)       # Repetitive instruction
```

Variable i sequentially iterates through the list [0,1,2,3,4] i.e. At first iteration, i gets automatically assigned to first value in the list i.e. 0 in the list [0,1,2,3,4] and then executes repetitive set of instruction(s). After the first iteration, i gets assigned with the next value in the list i.e. 1 in this case. This continues till i gets assigned with a value 4.

Now, as discussed earlier, we need to display 1000 consecutive numbers starting from 12 and with a difference of 5. We need to specify list as below.

sequence / list: [0,1,2,3,4,5,6,7……...,999]

It is not practical to write a list of such 1000 numbers. *Is there anyway by which we can generate such list automatically?* Yes, Python provides in-built function range(start, stop, step) which does it.

range(start, stop, step) where,

Start: [Optional][Default=0] An integer number specifying at which position to start.
Stop: [Required] An integer number specifying at which position to end (excluding the integer specified).
Step : [Optional][Default=1] An integer number specifying difference between two consecutive number in sequence.

range(5) produces list [0,1,2,3,4]

The complete program can be written as below by using for loop construct
Program 4.2 (*for* loop):

```
num = 12
diff = 5

for i in range (5):
   num = num+diff
   print ("Number = “, num)
```

Loop is an example of iteration control flow.
Let’s understand simpler example for the better understanding of for loop.

Practice Example 1: Calculate the sum of the numbers starting from 0 to 12.

Let us store the summation of numbers starting from 0 to 12 in a variable sum. Before beginning to add the numbers starting from 0, we have to initialize variable sum with value 0.

sum=0
sum=sum+0
sum= sum+1
sum= sum+2
sum= sum+3
sum= sum+4
sum= sum+5
sum= sum+6
sum= sum+7
sum= sum+8
sum= sum+9
sum= sum+10
sum= sum+11

sum= sum+12

It is seen that left hand operand of + is a variable name i.e. sum is a variable present at the left side of + operator. Right side operand is a constant value which is varying from 0 to 12. Hence, we can write it as sum=sum + value.

As sequence is varying from 0 to 12, we can initialize sequence to range (13). For the above program,
sequence / list: range (13)
Repetitive Statement: sum=sum + value

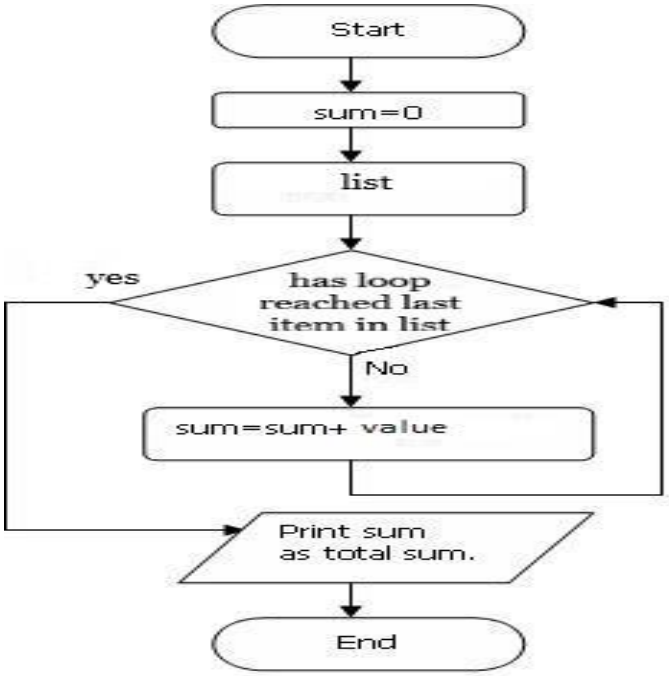


Fig.4.1

The complete program can be written as below.

Program 4.3:

```
sum = 0

for value in range (13):
    sum = sum + value

print ("Number = ",sum)
```

Practice Example 2: Write a program that calculates a factorial of a given number.

The steps involved are as follows.

1. Ask the user to enter a number for which he or she wants to calculate the factorial.
2. Receive an input number for which you want to calculate the factorial
3. As we know factorial of a number
   Fact=number*(number-1) * (number-2) *……...*1
   e.g. For a number 5
   Fact=5*4*3*2*1
   which can be written as

   Fact=Fact*5
   Fact=Fact*4
   Fact=Fact*3
   Fact=Fact*2
   Fact=Fact*1

   It means above operation is repetitive where num is varying from num to 1.
   The decrement is by 1. Hence, we can write

   Sequence: range(1,6)
   Repetitive Statement:
   fact=fact*value
4. Display the factorial value.

The complete program can be written as

Program 4.4:

```
factorial=1
num = int(input("Enter the number = "))
if if  num < 0:
     print("Sorry, factorial does not exist for negative numbers")
elif num == 0:
     print ("The factorial of 0 is 1")
else:
     for i in range(1,num + 1):
          factorial = factorial*i
 print("The factorial of",num,"is",factorial)
```

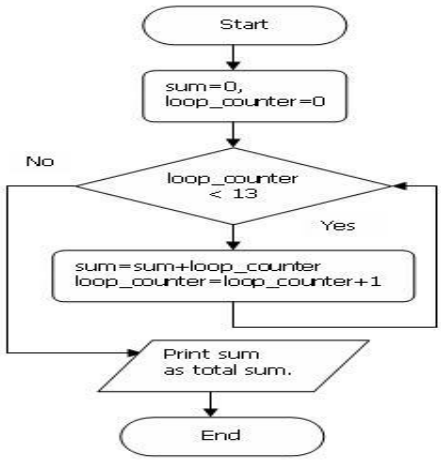
Let's write a program 4.3 using while loop that calculates sum of the numbers starting from 1 to 12.

Program 4.5:

```
sum = 0
loop_counter = 0

while (loop_counter<13):
   sum = sum+loop_counter
   loop_counter+=1

print("The total sum is = ",sum)
```



Q. Guess the output of the following programs.

I.

```
i=0
sum=0
for i in range( 11):
     pass
sum=sum+i
print("Number is:" ,sum)
```

Output:

Number is: 10

If you see carefully, there is a ***pass*** **keyword in for block, it indicates the repetitive statement is blank**. Hence, this loop gets executed 10 times. After the execution of the loop, the statement sum=sum+i gets executed. The value of the sum is 0 and the value of i becomes 10. Hence, the output is 10.

II.

```
sum = 0
i = 0
for i in range (0,10,2):
   if (i==10):
      i=i+1
      sum = sum+i
print (sum)
```

Output:

0

As variable i starts with value 0, it iterates like 0,2,4,6,8 and stops when its value becomes 10. Hence, if condition never becomes true and does not execute its body. Hence, the value of the sum variable remains 0 and 0 gets printed.

III.

```
a=[10,20]
for i in a:
    a.append(i)
print(a)
```

Output:

This is an infinite loop because the list *a* grows in each iteration. You will understand list in the later part of this book.

IV.

```
sum = 0
for i in range (0,21,2):
   if (i!=10):
      i+=2
      sum = sum+i
print (sum)
```

Output:

120

V. **Guess the value of the MAX so that following loop gets executed exactly 12 times.**

```
sum = 0
max = int(input("Enter the value of MAX so that following loop should get execute only 12 times"))

for i in range(1,max,2):
   sum = sum+i//2              # //  is integer division operator
   print("Iteration ",(1+i//2))
```

Output:

User should enter the value of MAX as 24 or 25 in order to get following output.

Iteration1
Iteration2
Iteration3
Iteration4
Iteration5
Iteration6
Iteration7
Iteration8
Iteration9
Iteration10
Iteration11
Iteration12

Q. Why multiple loop constructs are provided? Which loop construct should be used?

You can use any loop construct. Following is the detailed comparison chart. Multiple loop constructs are present only for programmer's convenience.

	for	***while***
1	*for* value in (sequence) statement	*while*(expression) statement
2	*for* loop iterates through a sequence.	*while* loop works based on a condition.
3	It may execute zero or multiple times based on the sequence provided.	It may execute zero or multiple times based on the condition specified as expression.
4	Generally used when the number of the repetitions of a loop are fixed or to iterate over a list or string.	Generally used when the number of the repetitions of a loop is not fixed.
5	*for* loop requires a list or iterable, there is no expression, condition or increment/ decrement expression.	Only 1 expression is mandatory. Initialization expression generally appears before while loop starts. Increment / decrement expression appears inside the statement part of *while* loop.
6	When continue is executed inside the *for* loop, it skips the rest of the code inside the loop for the current iteration only and loop continues on with next iteration.	When continue is executed inside the *while* loop, the control is transferred to the conditional expression which gets evaluated.
7	When accessing element from list or any iterable *for* loop is preferred.	When we have a condition after which loop has to be terminated *while* loop is preferred.

4.3 *break* (Jump control flow)

break is a jump control flow. When the *break* statement gets executed inside the loop, it immediately stops the execution of the instruction(s) inside that loop and the next statement after the loop gets executed.

Let's understand this concept by a simple program that calculates the number of the digits present in each number.

1. Receive an input number from the keyboard.
2. Assume the input number received is 8745. Divide the number by 10 and again divide the result of division by 10 unless the division is less than 1.
   i. Let's divide 8745 by 10 which results in 874.
   ii. Let's divide 874 by 10 which results in 87.
   iii. Let's divide 87 by 10 which results in 8.
   iv. Let's divide 8 by 10 which results in 0.8, which is less than 1.
   v. No. of digits in the number = No. of times the division operation carried as above

Program 4.6:

```
num=int(input())
count=1
while 1:
    if(num/10>1):
            count+=1
            num/=10
    else:
            break
print("No. of digits", count)
```

OR

Alternative method is as below.

1. Receive an input number from the keyboard.
2. When any integer number is divided by 10,100,1000,10000 etc. and if we get the division less than 1 and more than 0.1, then that particular number has n digits where divisor=10^n. i.e. If we divide 57382 by 100000 then we get the division 0.57382. Hence, the number is 5 digits.
   e.g. If the input number is 57382 then

Divisor	Division
10	5738.2
100	573.82
1000	57.382
10000	5.7382
100000	0.57382

3. As the number of the digits are not known in advance, we are not sure how many times the division operation by values 10, 100, 1000 etc. is needed to be repeated. Hence, we can operate this repetitive operation infinite number of times and exit from the loop when the division is less than 1 and above 0.1

When the condition i.e. division is less than 1 and above 0.1 is fulfilled, *break* statement should be used to exit from the loop.

if division < 1 and division >0.1
break

4. In this case divisor is needed to be changed by a factor of 10. Hence, we have to write it like this inside the loop.
   divisor = divisor*10
   Divisor is initialized by:
   divisor=1
5. We can define variable that stores the value of the number of times of repetition of a loop. The final value stored in this variable is the number of the digits in a number.
6. We need to check for '0' in input as 0 is one-digit number.

Program 4.7:

```
num=int(input())
count=0
divisor=1
while 1:
    if(num==0):                                  # if we encounter 0 as input
            count+=1
            break
    if(1<num/divisor>0.1):
            count+=1
            divisor*=10
    else:
            break
print("No. of digits",count)
```

4.4 *continue*

continue is a jump control flow. When the *continue* statement gets executed inside the loop, the program control gets immediately transferred to the conditional expression in case of *while* loop and next value in the list gets assigned to a loop counter in case of a *for* loop. If the conditional expression is true, the loop continues in a usual manner in case of *while* loop, whereas, the loop continues until value is present in the sequence.

	break	***continue***
1	When *break* statement gets executed inside the loops (like *for, while*), it immediately stops the execution of instruction(s) in the loop and the next statement after the loop gets executed.	When *continue* statement gets executed inside the loop, the program control gets immediately transferred at the beginning of the loop. The loop execution continues with its next iteration.
2	*break* statement can be used inside any loop.	*continue* statement can be used inside any loop.

Let's write a program that calculates the sum of numbers starting from 0 till 20 and the number should not be divisible by 2 or 3. In this program, we need to modify repetitive statement in such a manner that when any number that is completely divisible by 2 or 3, it should not be added.

We cannot use *if-else* statement because we need to exit from the loop. If we use *break* statement, the loop will get terminated & other numbers won't be considered. As we just want to skip that number and want to continue the loop for other consecutive numbers, we should use *continue* statement

Program 4.8:

```
rem1= 0
rem2= 0
sum= 0

for i in range (0,21):
   rem1=i%2
   rem2=i%3

   if (rem1==0 or rem2==0):
      continue
   sum+=i

print("The sum of all numbers starting from 0 to 20 and not divisible by 2 or 3 is ",sum)
```

4.5 Nested loops

Every loop has a repetitive statement. When any loop statement contains one or more loops inside it, it is called as a nested loop.

Let's write a program that displays the following pattern.

```
*
**
***
****
*****
******
*******
********
*********
**********
```

Explanation:

'*' is a character that needs to be displayed for fixed number of times based on the line number.
Line 1: Display character 1 time.
Line 2: Display character 2 times.
Line 3: Display character 3 times.
Line 4: Display character 4 times.
Line 5: Display character 5 times.
Line 6: Display character 6 times.
Line 7: Display character 7 times.

Line 8: Display character 8 times.
Line 9: Display character 9 times.
Line 10: Display character 10 times.

'*' is a character. Hence, we need a variable which will store it.
ch ='*' ;ch holds the character '*' where ch is the name used to refer memory location where '*' is stored.

The print method can be used to display a character.
print(ch)

Suppose we want to display '*' as output 10 times, the simplest logic to implement it is

```
for i in range(1,11) :
    print('*')
```

Above code prints exactly one star in each line, as print method in Python automatically appends newline character. To avoid printing to new line each time we must prove a blank string in 'end' argument of print method.

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

It will print in same line. However, we want to print the '*' as per the line number. Now, when the line number is 2, we need to print * twice. Hence, it is another loop that prints * equal to line number times. For this we need a new loop which would start iteration from 0 for every iteration. All these characters should be printed on a line and when the last character is printed, cursor should be moved to next line. Hence, print () should be the instruction after the execution of inner *for* loop.

Program 4.9:

```
ch = "*"

for i in range (1,11):
    for j in range (0,i):
        print(ch, end="")
    print()
```

The program 4.9 can be alternately written as below. This program executes *if-else* condition which requires more CPU cycles and subsequently time to execute the same. Hence, it's not advisable solution.

Program: 4.10:

```
ch = "*"

for i in range (1,11):
    for j in range (0,i):
        if (j==i-1):
            print(ch)
        else:
            print(ch, end="")
```

4.6 Counter Controlled and Sentinel Controlled Loops

When the number of the repetitions of a loop is known in advance, it is known as controlled or definite loop. e.g. All the programs discussed so far are examples of this category.

Suppose now we want to calculate the average marks of many friends standing in a queue in front of your computer, you need to click executable of this program for every friend. As we are doing the same operation, let's use the loop. However, we do not know the number of the repetitions of a loop in advance. Hence, we will define a condition that will be evaluated based on user input. The program should be continued unless the user enters N as a choice.

Program 4.11:

```
while 1:
   physics=int(input("Please Enter  Physics Marks : "))
   chemistry=int(input("Please Enter  chemistry Marks : "))
   mathematics=int(input("Please Enter  mathematics Marks : "))

   average= (physics+chemistry+mathematics)/3
   print("Average marks is ",average)
   choice=input("If don't want to continue, press 'N',Otherwise press any other character")
   if choice[0]=='N':
      break
```

Practice Example 3: Write a program that calculates LCM and GCD of 2 input numbers.

Program 4.12:

```
x = int(input("Enter a number"))
y = int(input("Enter a number"))

if x>y:
   smaller = y
   greater = x
else:
   smaller = x
   greater = y
for i in range(1, smaller//2+1):
   if((x % i == 0) and (y %i == 0)):
      gcd=i

while 1:
   if((greater % x == 0) and (greater % y == 0)):
       lcm = greater
       break
    greater += 1

print("LCM is ",lcm , " and GCD is ", gcd)
```

Practice Example 4: Write a program that generates following pattern using 4^2 for loops. e.g. If the input value is 5, it should generate a pattern as below.

```
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
1 2 3 4
1 2 3
1 2
1
```

Program 4.13:

```
inp = int(input("Enter input “))
for i in range(1, inp+1):
        for j in range(1,i + 1):
                print(j, end=””)
        print(“”)
for i in range(1, inp+1):
        for j in range(1, inp-i):
                print(j, end=”)
        print(“”)
```

Practice Example 5: Write a program that produces following pattern.

```
 *
 * *
* * *
* * * *
* * * * *
```

Program 4.14:

```
for i in range(1,6):
    print(" " *(5-i), "* " *(i),sep="")
```

Q. What is a structured programming? Does Python support structured programming?

Structured programming defines logical structure of instruction(s) that makes easy to read, write and maintain a program. It follows following approaches in general.

i. It sets certain rules to use nesting, branching and doing iterations in a program. Ideally, it allows sequential, selection and iteration control flow in a program.
ii. It divides a program into small units known as function (method).
iii. It defines a block structure that allows to define local variables.
iv. Python language supports methods, sequential, selection, iteration control flows and block structure. Hence, Python is a structured language.

Q. What are the different keywords used to to exit from a loop?

i. *break* ii. *return*

4.7 Points to Remember

1 *for, while, break, continue* are reserved keywords

2 *for* i in range(10):
    print(i)
    i = 5 #assigning any value here doesn't affect the execution of *for* loop as it is overwritten
        # in next iteration.

3 A loop that executes set of instruction(s) endlessly is called as an infinite loop. Conditional expression never becomes false in case of infinite loop if it does not contain any jump statement like *break, exit, return* etc. that gets executed in repetitive set of instruction(s) of a loop

Following program is an example of infinite loop.

```
i = 1
while i>0:
    print("I am learning to write a program")
```

4 In *while* loop Loop_counter can be incremented/decremented by any value based on the requirement as below.

```
loop_counter= loop_counter + 2
loop_counter= loop_counter - 1
loop_counter= loop_counter+0.45
```

Following program calculates sum of all even numbers starting from 0 to 1000 & displays it.

```
sum = 0
loop_counter=0
while(loop_counter<=1000):
   sum+= loop_counter
   loop_counter+=2
print ("The sum is ", sum)
```

4.8 Deep Knowledge Section

Q I. **Guess the output of the following program.**

```
t=[1,5]   # Pl, study this example after studying chapter 6 i.e. list
for i in t:
    t.remove(i)
    t.append(i+1000)
print(t)
```

Output:
[5, 2001]

The sequence is modified inside the loop, Python internally keep track of item to be used next iteration. As first item is removed, 5 becomes the first item & 1 is appended to list after adding 1000 to it since the actual length of list is unchanged Python in second iteration the internal counter uses the next element which is now 2001 hence, the output.

II. Write a program that calculates the sum of integers starting from 0 to 100. Do not use for, or while loop.

```
def add(num):
   if (num==0):
      return 0
   else:
      return (num+add(num-1))

a = add(100)
print(a)
```

III. Write a program that calculates the sum of integers starting from 1 to 100 using while loop that has conditional expression always true.

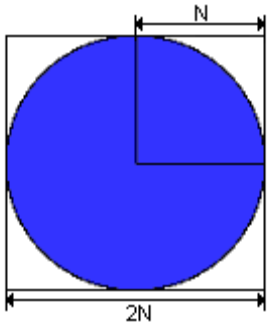
```
sum=0;i=1
while 1:          #Conditional expression is always true
  if i!=101 :
     sum=sum+i
  else:
    break
 i=i+1
print("The sum of the numbers starting from 1 to 100 is",sum)
```

IV. Write a program that generates following pattern using 1,2 and 3 for loops.
e.g. If the input value is 5, it should generate a pattern as below.

```
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
1 2 3 4
1 2 3
1 2
1
```

```
#for loop Approach:
inp = int(input("Enter input "))
for i in range(1, inp+1):
        for j in range(1,i + 1):
                print(j, end='')
        print("")
for i in range(1, inp):
        for j in range(1, inp-i+1):
                print(j, end='')
        print("")
```

V. Write a program that calculates π using Monte Carlo method?



Let's first understand how to calculate π (pi) manually. This method is invented by Monte Carlo. Fig. shows a circle having center(0,0) and radius N which is inscribed inside a square. Needless to mention, length of the side of a square is 2N.

Area of a Circle $=\pi*N^2$ Where N is the radius of a circle.
Area of a Square $=(2N)^2 = 4N^2$ Where 2N is the length of the side of a square.

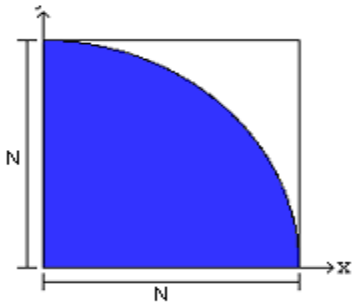
$$\frac{\text{Area of a Circle}}{\text{Area of a Square}} = \frac{\text{Number of points present inside a Circle}}{\text{Number of points present inside a Circumscribed Square}}$$

$$\frac{\pi*N^2}{4N^2} = \frac{\text{Number of points present inside a Circle}}{\text{Number of points present inside a Circumscribed Square}}$$

$$\frac{\pi}{4} = \frac{\text{Number of points present inside a Circle}}{\text{Number of points present inside a Circumscribed Square}}$$

$$\pi = 4 * \frac{\text{Number of points present inside a Circle}}{\text{Number of points present inside a Circumscribed Square}}$$

It means we can calculate π if we can count number of points present inside a circle considering all points inside a square. As square and circle are equally divided in 4 quadrants, let's consider all points in a 1st quadrant and inside a circumscribed square.



Coordinates of end point of a diagonal in a 1st quadrant are (N, N). Any other point whose x or y coordinate are greater than N are outside the square. Hence, we have to consider points whose x and y coordinates values lies in a range starting from 0 to N.

Let's check mathematically whether any point a(x_1, y_1) is inside a circle. Hope you remember the distance formula.

distance of point a(x1, y1) from origin(0,0)=$\sqrt{(x_1-0)^2 + (y_1-0)^2}$

$=\sqrt{(x_1^2 + y_1^2)}$ i.e. square root of $(x_1^2 + y_1^2)$

When distance of a point from origin is less than radius N, the point lies inside a circle. Hence, we can verify

$N=\sqrt{(x_1^2 + y_1^2)}$

i.e. $(x_1^2 + y_1^2) < N^2$ # Squaring both sides

Assume radius of a circle is 1000 i.e. N=1000

Let's divide x-axis of 1st quadrant of radius N into 1000 strips. For first strip, the value of x-coordinate is 1 and the value of y-coordinate varies from 1 to 1000. It means we need to check,

$(1)^2+(1)^2 <= (1000)^2$ # Point is inside a circle

$(1)^2+(2)^2 <= (1000)^2$

$(1)^2+(3)^2 <= (1000)^2$

.....

.....

$(1)^2+(999)^2 <= (1000)^2$

$(1)^2+(1000)^2 <= (1000)^2$

These are error prone and tedious calculations, let's use Python for loop.

```
for j in range(N+1):
    if (1*1+(j*j)<=N*N) :
        count+=1
```

Similarly, For 2nd strip

$(2)^2+(1)^2 <= (1000)^2$

$(2)^2+(2)^2 <= (1000)^2$

$(2)^2+(3)^2 <= (1000)^2$

.....

.....

$(2)^2+(999)^2 <= (1000)^2$

$(2)^2+(1000)^2 <= (1000)^2$

Similarly, For 1000th strip

$(1000)^2+(1)^2 <= (1000)^2$

$(1000)^2+(2)^2 <= (1000)^2$

$(1000)^2+(3)^2 <= (1000)^2$

.....

.....

$(1000)^2+(999)^2 <= (1000)^2$

$(1000)^2+(1000)^2 <= (1000)^2$

For 2nd strip

```
for i in range(N+1):
    if (2*2+(i*i) <=N*N):
        count+=1
```

It means we need above repetitive calculation for x coordinate values varying from 1 to 1000.

```
for i in range(N+1):
    for j in range(N+1):
        if (i*i+(j*j) <=N*N):
            count+=1
```

The complete program is as below.

```
n= int(input("Enter the value of N : "))
count = 0
for i in range (1,n+1):
   for j in range (1,n+1):
      if ((i*i)+(j*j)<=(n*n)):
         count+=1
pi = 4.0*count/(n*n)
print ("Value of Pi is: ",pi)
```

The above program may not calculate correct value of pi for smaller values of N like 5, 10, 100 etc. Following modified version of the program calculates correct value of pi for any value of N like 0.00000000000000000001, 0.000000.1,0.1,1,10 ,1000,100000 or more.

```
n= int(input("Enter the value of N : "))
count = 0
step = 5000
total_points = 0
i=n/step
while i<=n:
   j=n/step
   while j<=n:
      total_points+=1
      if ((i*i)+(j*j)<=(n*n)):
         count+=1
      j+=n/step
   i+=n/step
pi = 4.0*count/total_points
print ("Value of Pi is : ",pi)
```

Questions

Q.1 State true or false.

i. Loop is a programming construct by which a computer can execute set of instruction(s) repetitively for the specified number of times.

ii. *While* loop executes a set of instruction(s) repetitively while a condition is true.

iii. *For* loop in Python evaluates the condition in every iteration.

iv. A loop that executes sets of instruction(s) endlessly is called as infinite loop

v. When the *break* statement gets executed inside the loop, it immediately stops the execution of instruction(s) inside that loop and the program terminates.

vi. It is possible to write a nested loop.

vii. Range method takes up to two argument.

viii. Label keyword is present in Python.

ix. We can have nested for loop

x. Sequence(iterable) is optional in *for* loop.

xi. *For* loop is preferred in case of iterating through a list of element.

xii. Expression is mandatory in *while* loop

xiii. When the *break* statement gets executed inside the loop, it immediately stops the execution of instruction(s) inside that loop and the next statement after the loop gets executed

xiv. When the *continue* statement gets executed inside the loop, the program control gets immediately transferred to the conditional expression in case of *while* loop

Q.2 Guess the output of the following program.

i.
```
sum = 0
for loop_counter in range(10):
    sum+=loop_counter
print("The total sum is sum" ,sum)
```

ii.
```
t[1]
for i in t:
   t.append(i)
   print(i)
```

iii.
```
sum=0
for i in range(0,10,2):
    sum=sum+i
print("Sum is ", sum, end="")
```

iv.
```
sum=0; i=1
while i!=10 :
    sum=sum+i
    i+=2
    print("Sum is", sum)
```

v .
```
for i in range(1, 10):
  print ("*"*i)
```

vi.
```
t=list(range(1,10))
for i in t:
   if i%2 == 0:
      t.append(i*1.5)
print(t)
```

vii.
```
t=[100, 1000, 10000]
for i in t:
    t.remove(i)
    t.append(i+5)
print(t)
```

Q.4 Which control flow types are features of structured programming?

Q.5 Which control flow is not a feature of structured programming?

Q.6 Using *for* loop and range method create two lists, one of even and other of odd number till 100?

Q.7 Is sequence mandatory in case of *for* loop?

Q.8 Compare the followings.

I. *break* and *continue*

II. structured and unstructured programming

i. *for* and *while*

Q.9 Which expression is mandatory in case of *while*?

Q.10 Write a program that finds the average of 4 decimal values supplied by a user?

Q.11 Write a program that determines number supplied by a user is even or odd?

Q.12 Write a program using *if-elif-else* construct that checks the number supplied by a user is completely divisible by 2, 4 and 16?

Q.13 Write a program using *for* loop that checks the number supplied by a user is completely divisible by 5, 25 and 125?

Q.14 Write a program using any loop that prints following text.

```
Hi
Hi
Hi
Hi
Hi
Hi
Hi
Hi
```

Q.15 Write a program using any loop that prints following text.

Hi
Hi
Hi
Hi
Hi
Hi
Hi
Hello
Hi
Hi
Hi
Hi
Hi
Hi
Hi

Q.16 Write a program using *continue* statement that prints all numbers expect numbers which are not divisible by 7 and or 9.

Q.17 Write a program by using a *break* construct that prints all numbers starting from 10 to 100 until the sum of the digits of a number are divisible by 11. (Hint: Use *break* construct when this condition becomes true.)

Q.18 Write a program without using a *break* construct that prints all numbers starting from 10 to 100 until the sum of the digits of a number are divisible by 11.

Q.19 Write a program that calculates the average of 4 numbers until the user types the character Y?

Answers:

Q1.

i. True	**ii.** True	**iii.** False	**iv.** True
v. False	**vi.** True	**vii.** False	**viii.** False
ix. True	**x.** False	**xi.** True	**xii.** True
xiii. True	**xiv.** True		

Q2.

i. The total sum is sum 45
ii. NameError: Name t is not defined
iii. Sum is 20
iv. Infinite loop
vi. [1, 2, 3, 4, 5, 6, 7, 8, 9, 3.0, 6.0, 9.0, 12.0, 9.0, 18.0, 27.0]
vii. [1000, 105, 10010]

v.
```
*
**
***
****
*****
******
*******
********
 *********
```

Chapter

5

STRUCTURED PROGRAMMING: FUNCTION

5.1 Introduction

We have been using mobile in our daily life. But, do we know how mobile is designed and created? Without knowing the details of it, we are using it. Similarly, as a programmer, we have been using print function since we wrote first Python program. Do we know whether print is a single instruction or set of instructions? We don't know, but we are using print function to print output as per our requirement.

5.2 Function

5.2.1 Why Function?

Let's write a program that calculates the highest number from a set of 3 numbers i.e. 12, 79 and 18. One can easily write following algorithm to develop this program.

1. Store the numbers 12, 79 and 18 into a memory.
2. Compare number1 with number2. If number1 is greater than number2, store number1 as the highest number else store number2 as the highest number.
3. If number3 is greater than the highest number, number3 is the highest number.
4. Display the value of highest number.

Program 5.1:

```
number1=12; number2=79; number3=18

if(number1 > number2):
        highest = number1
else:
        highest = number2

if(number3 > highest):
        highest = number3

print('The highest number from a given set of three numbers is ',highest)
```

Output:
The highest number from a given set of three numbers is 79.

Good job! In the above program, you have written a set of instructions (code) that finds the highest number from 12, 79 and 18.

Assume for some reason, again you need to find highest number from following set of three numbers.
i. 12, 79, 18 ii.534, 8, 91 iii.314, 2, 10

You may write a program in a following manner.

Program 5.2:

```
number1=12
number2=79
number3=18

if(number1 > number2):
        highest = number1
else:
        highest = number2
if(number3 > highest):
        highest = number3

print('The highest number from a given set of three numbers is ',highest)

number1=534;number2=8;number3=91

" 'Repetition One'"
if (number1 > number2):
        highest = number1
else:
        highest = number2
if(number3 > highest):
        highest = number3
print('The highest number from a given set of three numbers is ',highest)

number1=314;number2=2;number3=10

'"Repetition Two"'
if(number1 > number2):
        highest = number1
else:
        highest = number2
if(number3 > highest):
        highest = number3
print('The highest number from a given set of three numbers is ',highest)
```

Output:
The highest number from a first set of three numbers is 79.
The highest number from second set of three numbers is 534.
The highest number from third set of three numbers is 314.

It is easily understood that the instructions shown in the bold letters in the above program are used 3 times to calculate the highest number from the sets. i.e. (12, 79, 18); (534, 8, 91) ;(314, 2, 10).

Suppose if you need to find the highest number from such 1000 sets of numbers, you may feel to copy and paste repetitive set of these instructions 1000 times.

But, do you think it is a practicable approach? Of course not! it's not at all practicable approach. There can be mistakes in copying and pasting these set of instructions. Moreover, **it increases the source code size** of a program. Naturally, following thought will pop up in your mind.

Is there any way by which one can write set of instructions once and use (refer) it for different values to perform the same task whenever needed? The solution is to use Python function feature that allows to write set of instructions once, refer it by name and use it whenever required.

It means it is possible to write the above bold lettered set of instructions once (that calculates highest number from 3 numbers), refer it by some name and use it to find the maximum number from any 3 integers whenever needed.

5.2.2 What is a Function?

In layman's language, function provides functionality. e.g. Watch provides functionality of showing you a correct time. Here, we pay money for the watch and purchase it. We are not bothered how watch is designed and developed to display the correct time. We are only interested to use the watch to see correct time whenever needed.

Function Name: PurchaseWatch
Parameters: Money and Name of the company of watch
Return Value: Watch

The idea is to identify repetitive set of instructions & develop a function so that instead of writing the same instructions (code) again and again for different inputs, we can call the function.

Function is a set of instructions arranged in a logical sequence to perform a certain task repeatedly.

Following steps can be used to define a function.

1. Identify set of instructions that needs to be repeated to perform certain task
2. Refer these set of instructions by a valid **identifier** (function name) & write it in this standard format. *def* <function name>**:** . Do not forget to write after writing the function name.
3. Mention the **input parameters** on which these set of instructions operate. All these input parameters must be mentioned after a function name **inside parentheses separated by a comma.**
4. Function operates on input parameters and returns result as it's return value. Unlike Java, the data type of this **return value is not mentioned before the name of a** function name**.**

Let's use above 4 steps to write a function that finds maximum number from three integer numbers.

Following are the steps to write a function that finds the highest number from 3 integer numbers.

Step 1: Identify repetitive set of instructions & write it as shown below.

```
if(number1 > number2):
        highest = number1
else:
        highest = number2
if(number3 > highest):
        highest = number3
```

Step 2: These set of instructions can be referred by any valid identifier/function name. e.g. find_max Do not forget to mention *def* keyword before writing function name and : after writing function name.

```
def  find_max():            # Do not forget to mention def keyword before writing function name
  if(number1 > number2):  # and : after writing function name
        highest = number1
  else:
        highest = number2
  if(number3 > highest):
        highest = number3
```

Step 3: These instructions operates on 3 numbers. Hence, input parameters number1, number2 and number3 must be mentioned inside parentheses separated by a comma after find_max.

```
def  find_max(number1,number2,number3) :
    if (number1 > number2):
        highest = number1
    else:
        highest = number2
    if(number3 > highest)
         highest = number3
```

Step 4: These repetitive set of instructions operate on the values number1, number2 and number3 and returns the highest value among them.

The complete function find_max can be defined as below.

```
def find_max( number1,number2,number3):
     if(number1 > number2):
       highest = number1
      else:
        highest = number2
       if(number3 > highest):
         highest = number3
       return highest
```

Program 5.1 can be rewritten using function concept as below.

Program 5.3:

```
def  find_max( number1,number2,number3):
    highest=0
    if(number1 > number2):
        highest = number1
    else:
        highest = number2
    if(number3 > highest):
        highest = number3
    return highest

# Driver function (Actual execution of program begins here)
highest=find_max(12,79,18)
print("The highest number from given set of three numbers is ",highest)
```

Output:

The highest number from given set of three numbers is 79.

FAQ:

Q. What is the meaning of the below instruction in the above program?

highest = find_max(12,79,18)

Here, find_max(12,79,18) is used to call the find_max function. As per the definition, find_max function accepts 3 numbers and returns a number. When compiler tries to compile this statement, it checks whether the definition of find_max function is present in the code or not. If it is not present, it gives compilation error. e.g. *def* find_max(num1, num2, num3) can be interpreted as

Function Name	find_max
Input Parameters	12,79,18
Return Value	Highest

Function parameters are separated by a comma as shown in the calling find_max function.

Q. How a function find_max(number1, number2, number3) can be called/invoked/used?

It can be called by passing input parameter values appropriately. e.g.

find_max(12,89,78) # Finds maximum value between 12,89,78 and returns it to the calling function

However, it is not a practicable approach as we are not storing value returned by a function find_max.

If you want to save the return value of the called function, you must define a variable that can store the return value of find_max function.

max_value = find_max(12,89,78)

Q. Is it mandatory to save the return value of a function when it is called?

No. It's optional.

Q. What are actual and formal parameters of a function?

A formal parameter, i.e. a parameter in the function definition. An actual parameter, i.e. a parameter, in a function call. Formal parameters need to be mentioned in the function definition.
e.g. *def* find_max(number1, number2, number3)

Here number1, number2 and number3 are the formal parameters. No memory is allocated for the formal parameters.

When a function is called as shown below.
max_value = find_max(number1, number2, number3) #Actual Parameters

values stored in variables number1, number2, number3 are copied to the corresponding formal parameters in a sequence. These are actual parameters because the actual values stored in these variables are passed to a function.

Q. What will happen if the statement returns highest; is not mentioned at the end of function find_max? Will it give compile time error?

No, this is allowed in Python. But it will not serve our purpose of writing and using function.

Q. Is it possible to write program 5.3 without storing result in a variable like highest? How?

Yes.

```
print("The highest number from a given set of three numbers is " ,find_max(12,79,18))
```

Q. Instead of using name find_max, can I use abc as a function name?

Yes, it is legal. Any valid identifier can be used. However, it is better to use function names which has relevant meaning to a functionality of that function. It improves the readability of a program and hence helps at the time of maintenance, enhancement of the application.

Q. How many input parameters a function can have?

Function can have 0 to any number of parameters depending upon the requirement.

Following program shows the use of a display function that accepts input as an integer, displays the word "Great" on a new line equal to the value of the integer passed as a parameter and does not return anything.

Program: 5.4

```
def display(n):
    while(n>0):
        print("Great")
        n=n-1

# Driver function (Actual execution of program begins here)
display(2)
```

Output:

```
Great
Great
```

Q. Give an example of a function that does not require any input parameter and returns nothing?

Assume that you need to write a postal address more than once in a particular situation. Writing an address of someone is tedious task. Being a repetitive task, let's write address inside a function once and (use) call a function whenever it is necessary to display.

Following program defines function showAddress that displays the address. It does not require any input parameter and does not return any value to the calling function.

Functions which do not have a *return* statement are called as void functions. void functions are those that do not return anything. *None* is a special type in Python which is returned after a void function end.

Program: 5.5

```
def showAddress():
    print(" Amit Deshapande.")
    print(" 34,Dhanraj Housing Society.")
    print(" Flat No.5.")
    print(" Shaniwar Peth.")
    print(" Pune.")
    print(" Maharashtra.")

# Driver function (Actual execution of program begins here)
showAddress()
```

Q. Is function feature of a structured programming? How it helps to write better programs?

Function is a feature of structured programming.

Function provides systematic structure to write better programs. By using function, one can reduce the size of the source program that improves the readability and helps at the time of maintenance/enhancement. Moreover, use of function allows us to divide the project work in a team that we will study in the later part of this chapter.

Q. Guess the output of the following program.

```
def fun(a,b,c):
    return (a + b + c)/3.0

# Driver function (Actual execution of program begins here)
print("Values is" ,fun(10,20,30))
```

Output:

Value is 20.0

(a + b + c)/3.0 represents an expression. As expression results into a value, function can return an expression. (a + b + c) / 3.0 is evaluated as value of float type and returned to the called function.

5.3 Scope of a Variable

One of my friends lost his driving license. When he visited police station, the concerned authority said "You cannot complaint at this police station. Please, visit the police station of area where you have lost the license." You must be thinking how this narration is related to Python programming? Yes, Python also defines certain rules so that defined variable or function can be accessed based on the visibility of that variable or function in the area of a program (project) from where it is accessed.
Let's guess the output of the following program.

Program 5.6:

```
def  find_max( number1,number2,number3):  # Function starts
   highest=0
   if(number1 > number2):
          highest = number1
   else:
          highest = number2
   if(number3 > highest):
           highest = number3
   return highest                              # Function ends

# Driver function (Actual execution of program begins here)
print("The highest number from given set of three numbers is ",highest)     #Line 13
```

You must be surprised to see this compilation error.

```
Traceback (most recent call last):
  File "Test.py", line 13, in <module>
    print("The highest number from given set of three numbers is ",highest)
NameError: name 'highest' is not defined
```

Yes, **highest** variable is defined inside find_max function. When variable is defined inside any function, it is accessible inside that function only. It means variable has function scope.

Scope of a variable depends upon a program unit (i.e. block, function, global etc.) where it is defined. e.g. If the variable is defined inside a function, it has a function scope

Similarly, if the variable is defined inside a block it is only accessible in that block.

Program 5.7:

```
a=10

def fun():
  if(a>0):                #if block starts
      b=201               # b is defined inside a block
      print(b)            #if block ends

# Driver function (Actual execution of program begins here)
print(b)    #Line 8
```

Again, you must be surprised to see the compilation error.

Traceback (most recent call last):
File "Test.py", line 8, in <module>
print(b)

Yes, **b** variable is defined inside if block as shown. When variable is defined inside any function, it is accessible inside that function only. It means variable has function scope.

Variable defined inside a function or block is known as local variable. The scope of a local variable starts from its definition and continues till the end of the block that contains the variable. A local variable must be declared before it can be used. Local variable must be initialized.

5.3.1 Nested Function

When a function is defined inside another function, it is called as nested function.

Program 5.8:

```
def fun():                      # Outer function fun1 begins
        i=10

        def fun1():                     # Inner function fun1 begins
                print(i)                # Inner function fun1 ends
        fun1()                  # Outer function fun1 ends


fun()
```

Output:
10

Here, fun1() is defined inside outer function fun() & referred as inner function.

It is important to note that the outer function must be called in order for the inner function to execute. If the outer function is not called, the inner function will never execute.

Let's try to guess the output of the following program.

Program 5.9:

```
def fun():
        i=10

        def fun1():
                print(i)

# Driver function (Actual execution of program begins here)
fun()
```

Output:

The output is blank because inner function fun1 is defined inside outer function fun(), but it is not called. Hence, it is necessary to call inner function from outer function to use it.

Let's try to guess the output of the following program.

Program 5.10:

```
def fun():
        i=10
        def fun1():
                i=20
                print(i)
        fun1()
        print(i)      # Accessing value of i from outer function
# Driver function (Actual execution of program begins here)
fun()
```

Output:

```
20
10
```

It is seen that though we have changed value of variable i inside inner function, it is not reflected (got changed) in outer function. So, how it is possible to achieve so that the value changed in inner function gets reflected in outer function as well? Yes, Python provides *nonlocal* keyword.

Program 5.11:

```
def fun():
        i=10
        def fun1():
                nonlocal i    # Modified value of i gets reflected in the outer function as well
                i=20
                print(i)
        fun1()
        print(i)      # Accessing value of i from outer function
# Driver function (Actual execution of program begins here)
fun()
```

Output:

```
20
20
```

When any variable is defined inside function, it is only accessible in that function. Is there any why by which we can declare a variable inside function, and it can be accessed outside? Yes, define that variable as *global* inside that function.

Program 5.12:

```
def fun():
        global j
        j=123
# Driver function (Actual execution of program begins here)
fun()
```

```
print(j)
```

Output:
123

The same is true for nested functions as well.

Program: 5.13

```
def fun():
        i=10

        def fun1():
                global i
                i=20
        fun1()
fun()
print(i)
```

Output:
20

It is important that the function where *global* variable is defined must get called at least once before it's used, otherwise, the instruction that defines *global* variable does not get called and variable is not initialized. Hence, you will get compilation error.

Let's try to guess the output of the following program.

Program: 5.14

```
def fun():
        i=10

        def fun1():
                global i
                i=20
        fun1()
        print(i)  # Though i is defined global, this i refers to the value stored in outer function i.

# Driver function (Actual execution of program begins here)
fun()
print(i)
```

Output:
10
20

Though i is defined *global*, this i refers to the value stored in outer function i.

- ✓ In general, indentation defines a scope.
- ✓ In Python, we can usually access a variable if it is defined with the same level of indentation until indentation level changes

5.3.2 Teamwork is Spirit

Do you think that it is possible to construct a building by a single person? Of course not, it's not practical. Many people work together to construct a building. Similarly, software application is developed by more than one person.

In the software industry, when any application is developed, it is difficult for a single person to write all functions. Hence, the work is divided into different team members.

Let's write a program that finds the difference between maximum and minimum number from given set of 3 numbers. Assume that there is a team of 3 members Amit, Nisha and Team Leader who are involved in the development of this program.

Let's divide the work among team members as below.
Amit: find_max function
Nisha: find_min function
Team Leader (TL): program from where we call the functions

As all these 3 team members cannot work in the same file, let them work independently. Initially, they require to develop separate programs and deliver these programs to a team leader so that he can integrate it.
As Amit has to write find_max function, he has to create a file Max.py where he can write the definition of a find_max function as below.

#Max.py

```
def find_max(number1,number2,number3):
    highest=0
    if(number1>number2):
        highest = number1
    else:
        highest = number2
    if(number3>highest):
        highest = number3
    return highest
```

Similarly, Nisha has to write find_min function, she has to create a file Min.py where she can write the definition of a find_min function as below.

#Min.py

```
def find_min(number1,number2,number3):
    lowest=0
    if(number1<number2):
        lowest = number1
    else:
        lowest = number2
    if(number3<lowest):
        lowest = number3
    return lowest
```

What about Team Leader's work? Team leader must write a driver file (main file) which can use find_max and find_min functions to find the difference between maximum & minimum number from given set of numbers. Max.py, Min.py and Driver.py should be in the same directory.

In order to access functions defined in Max.py, it is necessary to define below instruction before you use it.
from Max import find_max

#Driver.py

```
from  Max import *
from  Min import *

max1 = find_max(34,23,22)
max2 = find_max(314,123,122)
max3 = find_max(234,223,222)
max4 = find_max(4,23,5)

min1  = find_min(34,23,22)
min2  = find_min(314,123,122)
min3  = find_min(234,223,222)
min4  = find_min(4,23,5)

print("The difference between highest and lowest number is ",(max1-min1))
print("The difference between highest and lowest number is ",(max2-min2))
print("The difference between highest and lowest number is ",(max3-min3))
print("The difference between highest and lowest number is ",(max4-min4))
```

Output:

```
The difference between highest and lowest number is 12
The difference between highest and lowest number is 192
The difference between highest and lowest number is 12
The difference between highest and lowest number is 19
```

This program successfully executes and displays the expected result.

Suppose if there are many functions and we want to use only one function; we can use below instruction. Below is the instruction to access find_max function from Max.py file assuming there is more than one function inside Max.py file.

from Max import find_max

5.4 Advantages and disadvantages of using Functions

Advantages:

1) Function **avoids writing the same instructions repetitively** to achieve the same functionality for different parameters. This reduces the number of the instructions in the source program. It helps at the time of maintenance or enhancement of an application.

2) **Reusability of a code** is one of the main advantages of using functions. Functions once written and tested thoroughly can be used from anywhere.

Function follows the approach **"Write once, use forever"**. e.g. In the calculation of highest marks in Physics, Chemistry and Mathematics subjects' program, we can use find_max function developed here.

3) **Standard library** contains many functions like print() that is used frequently in a program without knowing how these functions are developed. Otherwise, one must write around 125-150 instructions to achieve a functionality of print() function which is not advisable.

4) It's possible to **divide the project work** among different team members to complete the project in time. e.g. One member can write find_max function, another find_min and TL can write driver function. This helps to team members to work independently and develop defect free code.

5) **Independent software vendors** can write function implementations as per the requirement. It means they need not have to provide function implementation and hence they can keep their code secret. At the same time, client can use it without any problem. In a big project, it is not possible for a client to develop whole project on own. Hence, he/she can ask the vendors to write functions.

More Explanation point 3:

Do you know how mobile is designed and created? Without knowing the details of it, you are using it. You are least bothered if it is fulfilling your expectation. In daily life, we use many standard readymade units (Components) instead of reinventing the wheel. Similarly, it's better to use standard independently functioning separate units to develop a program so that

- It saves the time in program development as no pains to develop these units.
- As these units are tested well, chances are rare to cause any defect because of the incorrect functioning of these readymade standard units. e.g. We have used print() function to display data on the console without knowing the details of the implementation of print() function. If a function concept wouldn't be available, one had to copy and paste around 100 lines each time a print needs to be invoked.

Explanation Point 4:

If function construct is not used and work is divided in a team, it is necessary to copy and paste the code written by every member in the main program. If there is any error in the work of any member, whole instructions in the main function needs to be checked which is time consuming and not practicable. Suppose if the member who has written find_max function leaves the company and new member joins, he/she must understand the whole main program which is not desirable. New member can only understand find_max function.

Q. Are there any disadvantages of using a function?

When a function is called,
i. Address of the instruction that is present after a function call is made is copied on the stack.
ii. Values of actual parameters passed to function are copied on the stack as formal parameters.
ii. Local variables of a functions are created on the stack.

It means function call requires more memory and consumes time in copying the data. Hence, it reduces speed and memory.

Python function is a collection of statements that are grouped together to perform an operation. When you call the print() function, for example, the system executes several statements in order to display a message on the console.

Q. Is it possible to find the highest number between 6 whole numbers by using a find_max function that just finds highest number between a set of 3 numbers?

Yes, it is possible.

Program 5.15

```
def find_max(number1,number2,number3):
   highest=0
   if(number1>number2):
      highest = number1
   else:
      highest = number2
   if(number3>highest):
      highest = number3
   return highest

# Driver function (Actual execution of program begins here)
max1 = find_max(17,111,67)
max2 = find_max(9,78,6)
highest = find_max (max1, max2,max2)
print("The highest number is ",highest)
```

Q. Is it possible to use the find_max function to find highest number between these 2 numbers like 789,112? If yes, write a Python program that calls a function find_max to find the highest value between 789,112?

Yes. Following is the instruction that finds highest number between 789 and 112.
max_num = find_max(789,112)

Q. Can function max find the highest number between 11.5, 19.1, 581?

Yes. Python provides a special library function max(). If we write the following instruction
max_num = max(11.5,19.1,581) It will give us the maximum value between them.

Following chart gives detail information of scope

Type of variable	Scope	Life	Initial Default Values	Storage
Local	Function/Block level Scope	Until the block/function gets executed where it is declared/defined.	Need to be initialized	Stack
Class/Static Variable	Class level scope	Till a program execution completes.	0	Stack
Instance	Determined by access specifier	Until the object to which it belongs is defined	Need to be initialized	Heap

NOTE: Static variables & instance variables will be discussed in the later part of this book.

5.5 Recursive Function

A recursive function is a function that calls itself.

Let us go through a program that calculates addition of numbers starting from 1 to that number. e.g. If the input number is 5, the addition of all numbers starting from 1 to 5 is 1 + 2 + 3 + 4 + 5 = 5

Program 5.16:

```
def add(num):
   sum=0
   for num in range(1,num+1):
      sum = sum + num
   return sum

# Driver function (Actual execution of program begins here)
addition =add(3)
print(" ",addition)
```

Explanation:
If you think slight differently, you will feel addition of numbers starting from 5 to 1 can be written as below. You can pass 5 as input parameter to a function add.

Addition of number starting from 5 to 1 = add(5)
= 5 + add(4)
= 5 + 4 + add(3)
= 5 + 4 + 3 + add(2)
= 5 + 4 + 3 + 2 + add(1)
= 5 + 4 + 3 + 2 + 1 + add(0)

It is clear that
- add(5) can be computed if add(4) is known.
- add(4) can be computed if add(3) is known.
- add(3) can be computed if add(2) is known.
- add(2) can be computed if add(1) is known.
- add(1) can be computed if add(0) is known.

You can easily interpret add(0) as addition of numbers starting from 0 and ending with 0. i.e. add(0) is 0. It means we know return value of add(0) without any computation. If we write add function specifying the value of add(0) as 0, add function can easily compute add(1),add(2)…,add(number) where number is an input parameter to a function add.

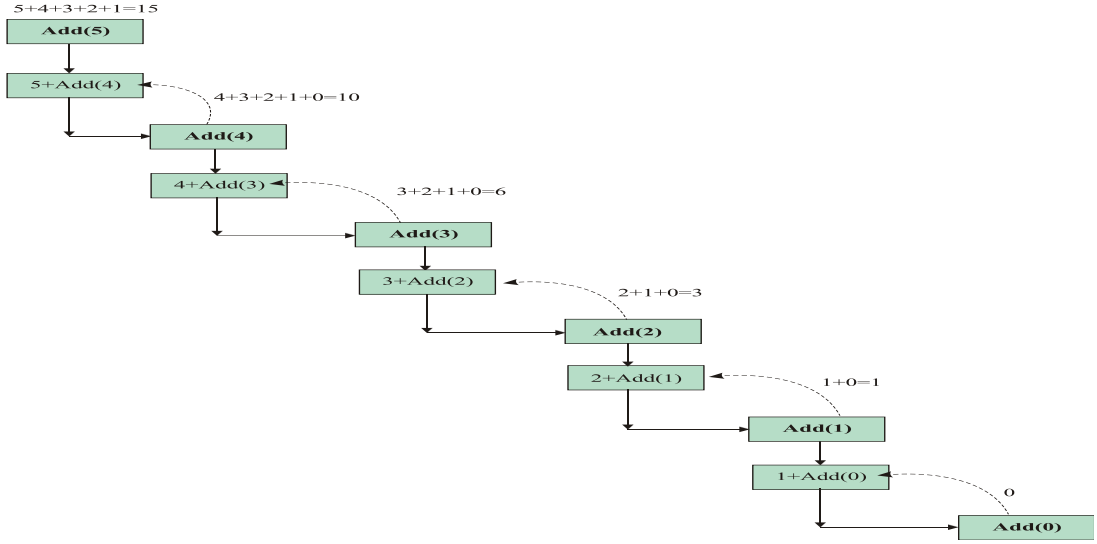
add function can be written by using below approach.
"If the input number is 0, *return* 0 as a return value, *else return* number + add(number); "

```
def add(number):
   if(number==0):
      return 0
   else:
      return number + add (number - 1)
```

Let's assume that number 0 is passed as input parameter, then 0 is returned (as a return value) which is the expected behavior of a function. If number is 2, then

```
def add(number):
    if(2==0):
        return 0
    else:
        return 2 + add(1)
```

Function add(2) calls itself by passing a parameter1. Hence a new space on the stack for add(1) is created for function add(1). Now, add(1) calls add(0). Hence, again new space in the memory for a function add(0) is created on the stack. When 0 is passed as input parameter, 0 is returned. Hence, we get the value of add(1) as 1. Subsequently, add(2) is 3 which is returned as a *return* value to the calling function.



Following program defines function addOrFactorial. addOrFactorial function accepts 2 parameters.

- It calculates the addition of the first parameter if second parameter passed is 0.
- It calculates the factorial of numbers starting from that number to 1 if second parameter passed is 1.
- If second parameter is neither 0 nor 1, function *returns* -1.

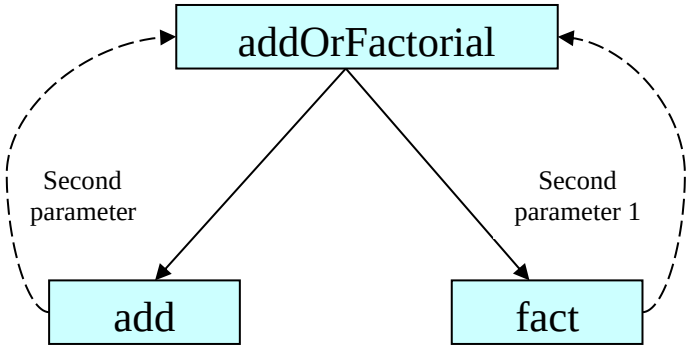


Fig.5.2

Program 5.17

```
def addOrFactorial(x, y):
   if (x == 1):
      return 1
   elif( y == 0):
      return add(x)
   elif(y ==1):
      return fact(x)
   else:
      return -1
def add(y):
   return(y + addOrFactorial( (y - 1),0))
def fact(y):
   return(y*addOrFactorial( (y - 1),1))

# Driver function (Actual execution of program begins here)
print("Addition of numbers starting from 1 to 5 is ",addOrFactorial(5,0))
print("Factorial of 5 is ",addOrFactorial(5,1))
```

Output:

```
Addition of numbers starting from 1 to 5 is 15
Factorial of 5 is  120
```

It is seen that addOrFactorial function calls either add or fact function which again calls addOrFactorial. As addOrFactorial function does not call itself directly but calls another function which in turn calls again addOrFactorial, it is known as indirect recursion.

When a recursive function is indirectly called, it is known as indirect recursion.

If recursive function "fun" calls another function "fun1" which in turn again calls function "fun", then it is an example of indirect recursive function.
Hence, we can write the definition of recursive function more precisely as below.

When a function calls itself directly or indirectly, it is known as recursive function.

It is little difficult to write indirect recursion function when the number of the instructions in a function implementation increases. These functions sometimes become difficult to understand. Hence, it should be used wherever required with precaution.

5.5.1 Practice Program 1:

Q. Write a recursive function power that calculates power of a given integer number.

Program 5.18:

```
def power(x,y):
   if (y==0):
      return 1
   else:
      return x*power(x, y-1)
```

```
#Driver function(Actual execution of program begins here)
print(power(3,2))
```

5.5.2 Practice Program 2:

Q. Write a recursive function factorial that calculates factorial of a given number.

Program 5.19:

```
def factorial(num):
    if (num==1):
        return num
    else:
        return num*factorial(num-1)
print(factorial(5))
```

5.5.3 Practice Program 3:

Q. Write a program that calculates a Fibonacci number of a given number.

The series of numbers that starts with 2 numbers 0, 1 and then each next number in the series is the sum of the two preceding numbers is known as the Fibonacci series. 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597, 2584, 4181, ... (each number is the sum of the previous two).

Every number in the Fibonacci series is Fibonacci number. e.g.

First Number: 0 **Second Number**: 1
Third Number: First Number + Second Number = 0 + 1 = 1
Fourth Number: Second Number + Third Number = 1 + 1 = 2
……
…….
n^{th} Fibonacci Number = $(n-1)^{th}$ Fibonacci number + $(n-2)^{th}$ Fibonacci number

Program: 5.20

```
def fibonacci(num):
    if (num==0):
        return 0
    elif(num==1):
        return 1
    else:
        return fibonacci(num-1) + fibonacci(num-2)

#Driver function(Actual execution of program begins here)
print(("Fibonacci is ", fibonacci(10))
```

5.5.4 Practice Program 4:

Q. Write a recursive function display that accepts integer value as a parameter and display "Hi" on separate line equal to the value of the argument passed.

Program: 5.21

```
def display(i):
   if(i==0):
      return i
   else:
     print("hi")
   display(i-1)
display(4)
```

5.6 Lambda (Anonymous) Function

Lambda or anonymous function is a function that is defined without a name.

lambda arguments: expression

lambda is a keyword in Python. Lambda function can have any number of parameters, but, only 1 expression. The expression is evaluated & returned as a result (return value).

Let's write lambda function that accepts x,y as arguments and return multiplication of x,y.

lambda x,y: x*y # lambda function begins with lambda keyword as shown,

This can be called by writing lambda function as shown and providing arguments in the parentheses.

```
(lambda x,y : x*y)  (3,4)
```

This will print nothing. However, if you use print statement, you will able to see output.

Program 5.22:

```
print( (lambda x,y: x*y)  (3,4)   )
```

Output:

12

So, whenever you want to use lambda function, you have to write definition and then provide parameters inside parentheses as shown in the above program. This can be avoided by referring lambda function by any variable .

Program 5.23:

```
Multiply=lambda x,y : x*y
print(Multiply(2,3))
print(Multiply(9,8))
```

Output:

6

72

Lambda function is used when we require a function without name and that too for short time.

Let's use lambda function as a first argument of filter function. Filter function takes first argument as a function and second argument as list[1] of parameters.

Let's write filter() function that filters out the list and returns only odd numbers

filter(lambda x: (x%2 != 0) , my_list)

```
oldList = [1, 5, 4, 6, 8, 11, 3, 12]
filter( lambda x: (x%2 != 0) , oldList )
```

As there will not be output, let's create list of filtered out odd numbers. i.e. odd numbers are stored in the lst and displayed.

Program 5.24:

```
oldList = [1, 5, 4, 6, 8, 11, 3, 12]
lst=list( filter( lambda x: (x%2 != 0) , oldList ) )
print(lst)
```

Output:
[1, 5, 11, 3]

5.7 Points to Remember

1. Function definition/implementation is always written with respective indentations.
2. Python supports dynamic typing.
3. It is not necessary that function should always have last instruction as a return value instruction. Multiple *return* statements can exist for a single function. However, when 1 *return* statement is executed, no other statement inside that function gets executed.
4. Function can only return one value at a time. It is not possible to return more than 1 value from a function at a time.
5. Parameters must be passed to a function in the order mentioned in a function declaration/prototype/signature.
6. We can define one function inside another function.
7. Function employs top down design approach. One can start with the main function and later he or she can divide a program in many functions.

[1] You can study this part of lambda function after understanding the concept of list in the chapter 6.

8. When we call any function by passing variable names as below,

```
one = 9
two = 12
k = sum(one, two)
```

Variable names are not passed. Values of the variables are passed. It means in the above function, 9 and 12 are passed as parameters to a function.

9. Static variables are known as class variable. Static variables are not instance or local variable. We cannot define static variables inside a function.

10. If static variable is not explicitly initialized, it gets automatically initialized with a value 0.

11. *Return* statement can return an expression because expression results into a value.

```
def sum(int a, int b):
    return (a + b)
```

12. When function calls itself, more complicated call to a function is expressed as a simple calculation. e.g. fact(5) is expressed as 5*fact(4) and so on.

13. A case for which the answer is known (and can be expressed without recursion) is called a base case. e.g. In case of factorial function, return value of fact(0) is known in advance. Each recursive algorithm must have at least one base case as well as the general (recursive) case. e.g. fact function cannot be expressed recursively if return value of fact function for any number is known.

14. Advantages of using recursive function are it can be written more elegantly.

15. It reduces source code size of that function.

16. Disadvantages of using recursive functions is that more space (memory) is required on the stack to store return address of calling function, local variable as well as parameters passed to that function. The speed of the execution is also lesser than the iterative function calls because the same function is called which requires copying of the parameters while passing to it.

17. In case of recursive function, if the condition is not satisfied that calls further execution of the same recursive function, program abnormally terminates as it goes into the infinite loop.

18. The static function cannot use non-static data member or call non-static function directly. (You will study this in the class related concept)

19. Function body can't be empty.

```
def fun():
 # Empty function body. This will not work.
```

20. It is possible to define a function that does not do anything.

```
def fun():
    pass
```

5.7 Deep Knowledge Section

Q 1. What is a Python decompiler?

Decompiler is to reverse engineer Python bytecode back into equivalent Python source code. A decompiler for Python should get the respective source file.

Q2. Guess the output of the below programs.

i.

```
def main():
   k=1
   print("inside")
   if(k==1):
      k+=1
      main()
   print("k=",k)
main()
```

This program displays "inside" as output continuously and terminates as it is an infinite loop.

ii.

```
def display(j):
   print(j )
   return(j)
i = 5
print(display(i) + i)
```

Output:

5

10

Questions

Q.1 State true or false.

i. The default return data type of a function is an integer.
ii. Function is a set of instructions arranged in a logical sequence to perform a certain task.
iii. Function can return one or more parameters at a time.
iv. Maximum 3 parameters can be passed to a function.
v. The scope of local variable is the block or function where it is declared/defined.
vi. Life of static variable is till the end of a program.
vii. Function employs bottom up approach.
viii. Every function should contain at least one statement.
ix. Recursive function can be called directly or indirectly.
x. One function cannot call another function.
xi. Instance variable can be speedily accessed as compared to class variable.
xii. print() is a user defined function.
xiii. Function is a primitive data type.
xiv. Function is a user data type.
xv. Though standard print() function is available, one can write print() function as per his/her requirement and use it.

Q.2 Guess the output of the following programs.

i.
```
class test:     # Note: Pl, solve this after studying class concept in chapter 9.
   x=0
   def stat(self):
      self.x+=1
      print("x= ",self.x)

t=test()
k=0
j=0
for j in range(1,4):
   t.stat()
print("x = ",t.x)
```

ii.
```
def sqr(x):
   x=x*x
   return x
t=sqr(3)
a=10
print(a+1,sqr(a+1))
```

iii.
```
def fun(x):
   y=0
```

```
   y=x*2
   return y
k=7
j=3
print(fun(j))
```

iv.

```
def fun(x):
   y=0
   y=x*2
   return y
k=7
k=fun(fun(k))
print(fun(k))
```

v.

```
class Test:
   count = 0
   @staticmethod
   def display():
      print(Test.count)
      Test.count = Test.count+1
Test.display()
Test.display()
```

Q.3 What are the advantages and disadvantages of using a function?

Q.4 Write a function that finds highest number from a set of 4 decimal numbers.

Q.5 Write a function that finds lowest number from given 5 numbers.

Q.6 Explain in detail how a function is called?

Q.7 Explain the meaning of the following keywords.

a. return
b. static
c. local instance

Q.8 Discuss the situations where you will use following type while defining a variable.

a. local
b. instance
c. static

Q.9 What is a scope of a variable? What different types of scopes a variable can have?

Q.10 Write a program that calculates simple and compound interest. Create separate functions for simple and compound interest and write them in the same source file. Before program termination, display the count which shows number of times these functions called in the program.

Q.11 Write a program that calculates simple and compound interest. Create separate functions for simple and compound interest and write them in separate files. Before program termination, display the count which shows number of times these functions called in a program.

Q.12 Write a program that calculates factorial of numbers starting from 2 to 8.

Q.13 What are the advantages and disadvantages of using recursive function?

Q.14 Compare iterative and recursive function.

Q.15 What are actual and formal parameters in a function

Q.16 What is called function and calling function? Explain with an example

Answers:

Q1.

i) False	ii) True	iii) False	iv) False
v) True	vi) True	vii) False	viii) True
ix) True	x) False	xi) True	xii) False
xiii) False	xiv) True	xv) True	

Q.2

I	x = 1,2,3	ii 11 121	iii 6	iv.56	v 0
	x = 3				1

General Information

The Fibonacci numbers are Nature's numbering system. They appear everywhere in Nature, from the leaf arrangement in plants, to the pattern of the florets of a flower, the bracts of a pinecone, or the scales of a pineapple. The Fibonacci numbers are therefore applicable to the growth of every living thing, including a single cell, a grain of wheat, a hive of bees, and even all of mankind.
For more information, please visit http://www.world-mysteries.com/sci_17.htm

Chapter

9

CLASS

9.1 What, Why and When Class Data Type?

Think for a moment that you are staying in a hostel. Let's say that there are 253 students staying in 253 rooms. Consider for a while that no room is allocated to a student, anyone can stay in any room and keep his/her books, clothes, bags and other items in any room.

- Will it be easy task for you to locate your item like book, bag etc. after say 2 weeks?

Practically this problem can be solved by allocating a room to a student where he/ she can keep his/her books, clothes, bag and other items. It means **your books and items are bound to a room now** and can be easily located.

Similarly, there are many entities involved in programming (project), Every entity has many attributes. Managing data associated with entities become challenging & leads to cause errors. Hence, systematic organization of data helps to develop better programs (projects). It's better idea to group attributes related to a particular entity into a single unit. **At this stage, you can think entity is an item that has physical existence. e.g. person, car, book.** Attributes of a book can be book name, author name, number of pages, price, ISBN number, Edition etc.

We will learn soon how this (grouping of attributes related to an entity) approach enables systematic organization of data, reduces number of variable names (identifiers) in a program which automatically improves readability and hence, maintenance becomes easy.

Let me define class at primary level as below. We will improve its definition eventually as your knowledge enhances ☺.

Class is a data type that provides a convenient means of grouping attribute(s) and method[1](s) related to a particular entity under a single name. It helps for easier handling and identification of data. These attributes can be of similar or dissimilar (int or float or char etc.) data types.

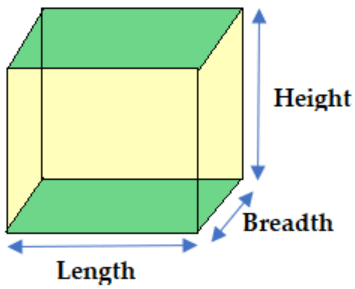
Entity **time** is associated with 3 attributes. i.e. seconds, minutes and hours. Entity **tile** is associated with attributes: type of tile (like ceramic, marble etc.), length, breadth, price, weight etc.

---

[11] Function defined inside the class is known as method.

It might be difficult for you to digest this concept at this stage, but, let us try to understand the concept of a class practically by understanding the following program.
Let's write program checks and displays whether individual metal brick purchased is pure or impure. Assume that the shape of the metal brick is cuboid as shown in the figure.

Metal Name	Length (cm)	Breadth (cm)	Height (cm)	Mass (gm)	Accurate Density gm/cm³
Silver	10	9.5	2.8	2793	10.5
Gold	7.1	5.8	9.1	89	19.3
Platinum	3.5	2.7	1.8	189	21.4



Hint: i. Calculate the volume of a cuboid=length*breadth*height;
ii. Then, calculate density=mass/volume;
iii. Check whether calculated and accurate density of a metal are equal.

- **If the calculated density and accurate density of metal are equal, metal is said to be pure.**

Every metal has a unique density. It means density of silver is always 10.5. If the calculated density of a brick is exactly equal to accurate density, the metal is pure.

Program 9.1:

```
# Silver Metal Plate. Initialization of attributes associated with a metal brick.
s_name="Silver"
s_length=10.00
s_breadth=9.5
s_height=2.8
s_mass=2793
s_accurate_density=10.5

#  Gold Metal Plate. Initialisation of attributes associated with a metal brick.
g_name ="Gold"
g_length=7.1
g_breadth=5.8
g_height=9.1
g_mass=89
g_accurate_density=19.3

#  Platinum Metal Plate. Initialisation of attributes associated with a metal brick.
p_name="Platinum"
p_length=3.5
p_breadth=2.7
p_height=1.8
p_mass=189
p_accurate_density=21.4
```

```
def determinePurity(name,length,breadth,height,mass,accurate_density):
    volume=length*breadth*height
    density=mass/volume
    if(density == accurate_density):
        print("Metal " + name + " is pure.")
    else:
        print("Metal " + name + " is not pure.")

#Driver function (Actual execution of program begins here)
determinePurity(s_name,s_length,s_breadth,s_height,s_mass,s_accurate_density)
determinePurity(g_name,g_length,g_breadth,g_height,g_mass,g_accurate_density)
determinePurity(p_name,p_length,p_breadth,p_height,p_mass,p_accurate_density)
```

Output:

```
Metal Silver is pure.
Metal Gold is not pure.
Metal Platinum is not pure
```

Q. What is the functionality of determinePurity( ) function?

This function accepts parameters associated with a cuboid shaped metal brick referred here. It calculates density of a given metal brick and compares with the accurate density of a given metal. If the calculated density and accurate density of a given metal is equal, the metal is said to be pure.

In the above program 9.1, there are 6 variables associated with every metal brick i.e. Metal name, length, breadth, height, mass, accurate_ensity. Hence, total variable names associated with 3 metal plates are 6 ×3=18.

If we want to check purity of such 1000 metal bricks, we require 6000 (Number of variable names per metal brick × Number of bricks=6×1000=6000) variables. As the number of the variables increases, the program 9.1 becomes unreadable. Consequently, it becomes difficult at the time of debugging[2] the instructions and subsequently also in the maintenance[3]/enhancement of an application.

Is there any data type in Python which allows user to define all attributes associated with an entity so that number of variables gets reduced in the program? Yes, it's class.

As metal brick is associated with the attributes name, length, breadth, height, mass and accurate density, let us define class CuboidMetal that associates these mentioned attributes.

```
class CuboidMetal        #Beginning of CuboidMetal class
    name
    length
    breadth
    height
    mass
    accurate_density     #End of CuboidMetal class
```

[2] Debugging is a process of finding and reducing the number of bugs/defects in a program.
[3] Software Maintenance is the process of enhancing the application or remedying defects.

Further, it is seen that values of the attributes related to class CuboidMetal are initialized as below.

```
# Silver Metal Plate
name="Silver"
length=10.00
breadth=9.5
height=2.8
mass=2793
accurate_density=10.5
```

This has been repeated 3 times to initialize values of attributes associated with Silver, Gold & Platinum metal plates. The instructions (code) are repetitive. *Instead of writing the same instructions, is it possible to write a method to initialize these attributes associated with every metal plate? Yes.*

```
def createCuboidMetal(name,length,breadth,height,mass,accurate_density):
    name = name
    length = length
    breadth = breadth
    height = height
    mass = mass
    accurate_density = accurate_density
```

In case of referred metal brick, createCuboidMetal and determinePurity methods are associated with entity CuboidMetal. Hence, class CuboidMetal can be defined as

```
class CuboidMetal                                    # Beginning of  CuboidMetal class
  name
  length
  breadth
  height
  mass
  accurate_density

def createCuboidMetal(name,length,breadth,height,mass,accurate_density):
    name = name
    length = length
    breadth = breadth
    height = height
    mass = mass
    accurate_density = accurate_density

def determinePurity(name,length,breadth,height,mass,accurate_density):
    volume=length*breadth*height
    density=mass/volume
    if(density == accurate_density):
      print("Metal " + name + " is pure.")
    else:
      print("Metal " + name + " is not pure.")          # End of CuboidMetal  class
```

Important point is that there is no need to declare attributes of the class as Python does not declare any variables. e.g. We write *a=10* and not *int a=10* in Python as declaration is not required. Hence, removing the declaration of these attributes and rewriting class CuboidMetal as below.

```
class CuboidMetal:                                          # Beginning of CuboidMetal class
  def createCuboidMetal(name,length,breadth,height,mass,accurate_density):
      name = name
      length = length
      breadth = breadth
      height = height
      mass = mass
      accurate_density = accurate_density

  def determinePurity(name,length,breadth,height,mass,accurate_density):
      volume=length*breadth*height
      density=mass/volume
      if(density == accurate_density):
         print("Metal " + name + " is pure.")
      else:
         print("Metal " + name + " is not pure.")          # End of CuboidMetal class
```

Let's discuss more about the special method createCuboidMetal that is used to initialize the attributes associated with CuboidMetal. e.g. "Silver", 10.00, 9.5, 2.8, 2793, 10.5. values are assigned to name, length, breadth, height, mass, accurate_density of silver metal brick.

The method which is used to initialize the attributes associated with a particular entity is called constructor. Constructor is a special method and it is referred by a reserved name __init__. Note that there are 2 underscores at the beginning and at the end of init.

Aim of the object-oriented programming languages is to create variables of class in the same way we create variables of built in data types. e.g. a=10

Similarly, we can create variable of a class as below.

s=CuboidMetal ("Silver", 2793, 10.00, 9.5, 2.8, 10.5) # Variable of a class (i.e. object)

Constructor does not have any *return* data type. It does not *return* any value. Also, first parameter of constructor is always *self*. Let's write a complete CuboidMetal class[4] as shown below.

```
def __init__(self,name,length,breadth,height,mass,accurate_density):      #Constructor
      self.name = name
      self.length = length
      self.breadth = breadth
      self.height = height
      self.mass = mass
      self.accurate_density = accurate_density
```

[4] Reader must understand class CuboidMetal. In this book, majority of Python concepts are explained with this class by gradually enhancing it.

Let's write a complete CuboidMetal class as shown below.

```
class CuboidMetal:                                          # Beginning of CuboidMetal class
   def __init__(self,name,length,breadth,height,mass,accurate_density):    # Constructor
      self.name = name
      self.length = length
      self.breadth = breadth
      self.height = height
      self.mass = mass
      self.accurate_density = accurate_density

   def determinePurity(self):
      volume=self.length*self.breadth*self.height
      density=self.mass/volume
      if(density == self.accurate_density):
         print("Metal " + self.name + " is pure.")
      else:
         print("Metal " + self.name + " is not pure.")          # End of CuboidMetal class
```

Let's write a complete program 9.1 with class CuboidMetal class as shown below.

Program 9.2:

```
class CuboidMetal:                                          # Beginning of CuboidMetal class
   def __init__(self,name,length,breadth,height,mass,accurate_density):          #Constructor
      self.name = name
      self.length = length
      self.breadth = breadth
      self.height = height
      self.mass = mass
      self.accurate_density = accurate_density

   def determinePurity(self):
      volume=self.length*self.breadth*self.height
      density=self.mass/volume
      if(density == self.accurate_density):
         print("Metal " + self.name + " is pure.")
      else:
         print("Metal " + self.name + " is not pure.")          # End of CuboidMetal class

# Driver Function  (Actual execution of program begins here)
s = CuboidMetal("Silver",10.00,9.5,2.8,2793,10.5)    # s is object of CuboidMetal class
s.determinePurity()

g = CuboidMetal("Gold",7.1,5.8,9.1,89,19.3)
g.determinePurity()

p = CuboidMetal("Platinum",3.5,2.7,1.8,189,21.4)
p.determinePurity( )
```

Output:

```
Metal Silver is pure.
```

```
Metal Gold is not pure.
Metal Platinum is not pure.
```

Q. What is the meaning of the instructions?

s = CuboidMetal("Silver",10.00,9.5,2.8,2793,10.5)
g = CuboidMetal("Gold",7.1,5.8,9.1,89,19.3)
p = CuboidMetal("Platinum",3.5,2.7,1.8,189,21.4)

s, g and p are objects of class CuboidMetal. Like a is variable in case a=10 instruction, s, g, p are objects of data type class CuboidMetal. We would be studying objects concept in the next section.

Q. What is *self* ?

self refers to the object itself. It is used to bind attributes of a class to the given parameters of a method or constructor. e.g. self of the object s in the above program associates "Silver",10.00,9.5,2.8,2793,10.5 values to the name, length, breadth, height, mass & accurate_density of the object s respectively.

Q. What is the meaning of the s.determinePurity()?

s associates with following values of the attributes.

Name	"Silver"
Length	10.00
Breadth	9.5
Height	2.8
Mass	2793
accurate_density	10.5

These values can be accessed & used from determinePurity() method. Hence, there is no need to pass any value separately as a parameter list.

Q. Is it possible to directly call determinePurity() method defined inside class CuboidMetal?

No. Object of CuboidMetal class is required to call determinePurity() method.

Let's enhance the definition of class now.

***Class* is a data type by which attributes & methods associated with an entity can be grouped together and referred as a single unit.**

9.2 Class and Object

9.2.1 Class

9.2.1.1 Definition

class is a keyword used to define class data type.
General form of a class definition is given as below.

```
class  <class_name>:
    def __init__(self, <one or more parameters>):                    #optional
    def <method_name> (self, attribute 1, attribute 2, ……, ….., attribute n):  #optional
```

By using the "***self***" keyword, we can access the attributes and methods of the class in Python. We will discuss further about it in chapter 11.

9.2.2 Object Definition

A variable of integer data type is known as an integer variable. e.g. a=10; Though, it seems easy to think that variable of integer data type is integer variable, variable of a class type is a class variable, but it is referred as an object of that class and not as a class variable.

A variable of a class is known as an object.

General Form:

class_object= class_name(value of variable1, value of variable2, ….,value of last variable)

Class CuboidMetal object can be created as below:

```
s = CuboidMetal ("Silver",10,9.5,2.8,2793,10.5)
```

```
#s is a class object of class CuboidMetal. Definition of s indicates
s.name="Silver"
s.length =10
s.breadth =9.5
s.height =2.8
s.mass =2793
s.accurate_density=10.5
```

Let me explain class & object concept using practical example. Amit lives in a big lavish bungalow; he has lavish car. We say Amit is from high class family. Trupti is also from wealthy family with bungalow, lavish car. We say Trupti is also from high class family. Here, we can say Amit, Trupti are objects of high class. **Object is also known as an instance of a class.**

Following table gives better picture of class & object concepts.

Class Definition	class CuboidMetal:
	  def__init__(self,name,length,breadth,height,mass,accurate_density):
	  self.name = name
	  self.length = length
	  self.breadth = breadth
	  self.height = height
	  self.mass = mass
	  self.accurate_density = accurate_density
	  def determinePurity(self):
	  volume=self.length*self.breadth*self.height
	  density=self.mass/volume
	  if(density == self.accurate_density):
	  print("Metal " + self.name + " is pure.")
	  else:
	  print("Metal " + self.name + " is not pure.")

Object Definition	s = CuboidMetal ("Silver",10,9.5,2.8,2793,10.5)

Q: How can we create an object of a class?

Object is created by calling constructor of a class.

s = CuboidMetal ("Silver",10,9.5,2.8,2793,10.5)

Q: How can we access objects?

Just like we need variables to access and use values, we need variables to access and reuse the objects that we create. Such variables that are used to access objects are called reference variables. In above example s, g, p are reference variables.

Q: Can we have method in a class which has the same name of a class attribute?

You can't have both method and an attribute with the same name.

Q: What is the meaning of instruction CuboidMetal ("Silver",10,9.5,2.8,2793,10.5)?

CuboidMetal ("Silver",10,9.5,2.8,2793,10.5) is used to create the object of the class CuboidMetal.

The enhanced complete definition of a class is as below.

***Class* is a data type that provides a convenient means of grouping attributes & methods related to a particular entity under a single name. It helps for easier handling and identification of data. These attributes can be of similar or dissimilar (int or float or char etc.) data types.**

Don't you think class definition is analogous to an architectural plan of a building? Based on the plan, many similar types of buildings can be constructed. Similarly, based on class definition, many objects can be created. Moreover, building's plan is made on the paper/computer and does not require any physical space, but space is required to construct a building.

Memory representation of object of *class* CuboidMetal is as shown below.

metal_type	length	breadth	height	mass	accurate_density
String	float	float	float	Float	Float
1245040	**1245050**	**1245054**	**1245058**	**1245062**	**1245066**

Memory representation of object of class CuboidMetal

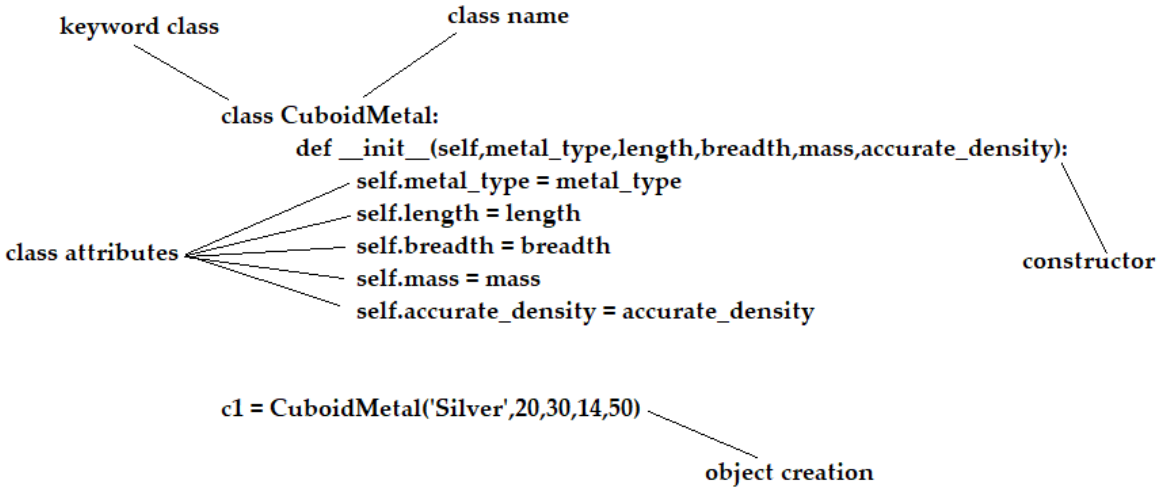


Fig. 9.1: Class & Object Concept

FAQ:

Q. How many classes are defined in the above program 9.2? Enlist them.

There is 1 class in the program 9.2. i.e. class CuboidMetal

Q. How many objects are defined in the above program 9.2? Enlist them.

Three objects defined are s, g and p are below the class.

Q. How one can access attributes of a *class*?

Object of a class can be used to access every attribute of a class. Attributes of a class can be accessed using . (dot operator) in the following manner:

```
s.name  # In the above example, s is an object of a CuboidMetal class.
```

Q. What are the differences that you have observed in Program 9.1 and Program 9.2?

Sr. No.	**Program 9.1 (Without *Class* Concept)**	**Program 9.2 (With *Class* Concept)**
1	Program 9.1 uses variables of built in data type to store actual data.	Program 9.2 defines class to associate attributes and methods of CuboidMetal.
2	Number of variable names are 18 to store data related to 3 metal bricks.	Numbers of variable names are only 6+3=9 to store data related to 3 metal bricks.
3	6 separate instructions are used to initialize variables associated with a single metal brick. name="Silver" length=10.00 breadth=9.5 height=2.8 mass=2793 accurate_density=10.5	Constructor is used to initialize variables associated with a single metal brick. s=CuboidMetal("Silver",10,9.5,2.8,2793,10.5)
4	Size of the source code (instructions) is more in case of this approach.	Size of the source code (instructions) is less in case of this approach.
5	Method signature is lengthy as every attribute associated needs to be passed. determinePurity(name,length, breadth,height,mass, accurate_density)	Attributes associated with an object are implicity to the called method. t = CuboidMetal() t.determinePurity()
6	Not concise and often ambiguous.	Compact coding with no repetition resulting in quicker workflow.
7	Program 9.1 is little difficult to understand as compared to program 9.2. Hence, enhancement and maintenance are comparatively little difficult.	Program 9.2 is easy to understand as compared to program 9.1. Hence, enhancement and maintenance are comparatively easier.

Q. Does program 9.2 saves memory as compared to program 9.1?

No. Memory allocated for variable storage is same in both the cases.

Q. Is there any difference between class and object? If yes, compare class with an object?

Sr. No.	Class	Object
1	*Class* is a blueprint (Master copy) to create objects.	It is a variable/instance of *class*.

2	class CuboidMetal:
	def init__(self,name,length,breadth,height,mass,accurate_density):
	self.name = name
	self.length = length
	self.breadth = breadth
	self.height = height
	self.mass = mass
	self.accurate_density = accurate_density
		CuboidMetal is a class and any variable of it is an object.

3	No memory is allocated when class is defined.	When we define an object, memory is allocated corresponding to the size of the class.
4	Class can be defined only once in a scope. e.g. Once CuboidMetal class is defined, it cannot be redefined in the program in the same scope.	Many class objects of a class can be defined.

5	Class data types cannot be assigned.
	class A:
	pass
	class B:
	pass
	A=B which is meaningless and cannot be compiled.
		Objects of same class can be assigned.
		c1=CuboidMetal ("Silver",10,9.5,2.8,2793,10.5)
		c2=CuboidMetal ("Gold",7.1,5.8,9.1,7103,19.3)
		c1=c2

Q. Is it possible to assign values of one class object to another object of the same class?

Yes, it is possible and implemented like this
s = CuboidMetal (“Silver”,10.00,9.5,2.8,2793,10.5)
p = CuboidMetal ("Platinum",10,9.5,2.8,5400,21.4)
s=p

Q. What is the difference in copying and assigning an Object?

The difference is that in case of copying an object, a new object is created, while, in case of assignment, new copy of the object is not created. Rather, a reference to the same object is facilitated.

Q. Is it possible to define multiple objects of same *class*?

Yes.

```
class CuboidMetal:
   def __init__(self,name,length,breadth,height,mass,accurate_density):
      self.name = name
      self.length = length
      self.breadth = breadth
      self.height = height
      self.mass = mass
      self.accurate_density = accurate_density

#Driver function (Actual execution of program begins here)
s = CuboidMetal("Silver",10.00,9.5,2.8,2793,10.5)
g = CuboidMetal("Gold",7.1,5.8,9.1,89,19.3) #Another object
p = CuboidMetal("Platinum",3.5,2.7,1.8,189,21.4) #Another object
```

Q. Guess the output of the following program.

I.

```
class CuboidMetal:
   def __init__(self,name,length,breadth,height,mass,accurate_density):
      self.name = name
      self.length = length
      self.breadth = breadth
      self.height = height
      self.mass = mass
      self.accurate_density = accurate_density

t = CuboidMetal ("Silver",10.00,9.5,2.8,2793,10.5)
print(t.name)
print(t.length)
print(t.breadth)
print(t.height)
print(t.mass)
print(t.accurate_density)
```

Output:

```
Silver
10.00
9.5
2.8
2793
10.5
```

II.

```
class CuboidMetal:
   def __init__(self,a,b,c,d,e,f):
      self.name = a
```

```
        self.length = b
        self.breadth = c
        self.height = d
        self.mass = e
        self.accurate_density = f

t = CuboidMetal ("Silver",10.00,9.5,2.8,2793,10.5)
print(t.accurate_density)
print(t.length)
```

Output:
10.5
10.00

III

```
class CuboidMetal:
    def __init__(self,name,length,breadth,height,mass,accurate_density):
        self.name = name
        self.length = length
        self.breadth = breadth
        self.height = height
        self.mass = mass
        self.accurate_density = accurate_density

t = CuboidMetal()
print(t.name)
print(t.length)
print(t.breadth)
print(t.height)
print(t.mass)
print(t.accurate_density)
```

Output:
TypeError: __init__() missing 6 required positional arguments: 'name', 'length', 'breadth', 'height', 'mass', and 'accurate_density'
When we called default constructor, it should have 6 arguments.
Note: Constructor will be discussed later in this book.

9.3 Objects and Method

Like any other built in data type, it's possible to pass an object as a parameter to a method.

determinePurity(< object of CuboidMetal class>) # program 9.2

As determinePurity method is directly called by object, then, there is no need to pass the same object as a parameter. i.e s.determinePurity(). However, we need to pass the object(s) as parameter when the object is of not that class which calls a method or even in the cases where we require more than one object of the same class to pass to the method. i.e.

s.determinePurity(g) # s & g are objects of CuboidMetal class

From above declarations, it's clear that it is possible to just pass an object instead of passing all the parameters associated with a particular entity individually. Needless to mention, it improves the readability and thereby makes maintenance / enhancement of the program easy.

Let's write a program to calculate the money required to paint 4 rooms of a bungalow. Dimensions of a bungalow are as given below. Also, print the message if the volumes of these 2 rooms are same.

	Length	**Breadth**	**Height**
Room1	8.5	7	9
Room2	8	8	9

Painting rate per square feet is Rs 98.
Hint: Area of a room=2*(length*breadth+ length*height+ height*breadth)

Program 9.3:

```
class Paint:
  def __init__(self,length,breadth,height):
     self.length = length
     self.breadth = breadth
     self.height = height

  def  compareVolume(self,  p1): # Object is passed to a method
     if (self.length*self.breadth*self.height) == (p1.length*p1.breadth*p1.height) :
        print ("Volume of the both the rooms are equal")
     else :
        print ("Volume of the both the rooms are not equal")

  def calculatePaintingCost(self):
     area=2*(self.length*self.breadth + self.length*self.height + self.breadth*self.height)
     money=98*area
     return money

#Driver function (Actual execution of program begins here)
t1 = Paint (8.5,7,9)
t2 = Paint (8,9,9)
total_cost = t1.calculatePaintingCost() + t2.calculatePaintingCost()
print("Total Rupees required to paint all 4 rooms is " + str(total_cost))
t1.compareVolume(t2) # Object that calls method is implicitly, while other object is passed explicitly
```

Output

```
Total Rupees required to paint all 4 rooms is 83104.0
Volume of the both the rooms are not equal
```

9.4 Objects and List

As we know, list is a data type which stores variables of similar or dissimilar data type elements in adjacent memory locations.

Let's declare a list that stores integers.
arr = []
data_type: int
list_variable: arr

Similarly, we can write
CuboidMetal_arr = []
data_type: class CuboidMetal
list_variable: arr

Let's rewrite the program 9.1 using list concept. List of three CuboidMetal objects is created and determinePurity method is called. It helped to use for loop and code size is reduced.

Program 9.4:

```
class CuboidMetal:
    def __init__(self,name,length,breadth,height,mass,accurate_density):
        self.name = name
        self.length = length
        self.breadth = breadth
        self.height = height
        self.mass = mass
        self.accurate_density = accurate_density

    def determinePurity(self):
        volume=self.length*self.breadth*self.height
        density=self.mass/volume
        if(density == self.accurate_density):
            print("Metal " + self.name + " is pure.")
        else:
            print("Metal " + self.name + " is not pure.")

#Driver function (Actual execution of program begins here)
CuboidMetal_arr=   [  CuboidMetal("Silver",10.00,9.5,2.8,2793,10.5),
                      CuboidMetal("Gold",7.1,5.8,9.1,89,19.3),
                       CuboidMetal("Platinum",3.5,2.7,1.8,189,21.4)  ]
for i in CuboidMetal_arr:
    i.determinePurity()
```

Output:

```
Metal Silver is pure.
Metal Gold is not pure.
Metal Platinum is not pure.
```

FAQ:

Q. Can I define a list arr as below?

```
s=90
g=1045
p=78
arr=[s, g, p]
```

Yes.

Q. Can I define a list arr as below?

```
arr = []
arr.append(90)
arr.append(1045)
arr.append(78)
```

Yes.

Q. Can I define a list arr as below?

```
t1= CuboidMetal ("Silver",10,9.5,2.8,2793,10.5)
t2= CuboidMetal ("S",10,9.5,2.8,2793,10.5)
t3= CuboidMetal ("Si",10,9.5,2.8,2793,10.5)
arr1=[t1,t2,t3]
```

Yes.

Q. Can I define a list arr as below?

```
arr = []
arr.append({"Silver",10,9.5,2.8,2793,10.5})
arr.append({"Gold",7.1,5.8,9.1,7103,19.3})
arr.append ({"Platinum",10,9.5, 2.8,5400,21.4})
```

Yes. This is a list of sets.

Q. Write an instruction that displays accurate_density of Platinum?

```
t= CuboidMetal("Platinum",10,9.5, 2.8,5400,21.4)
print(t.accurate_density)
```

9.5 Nested Class- Class as a Data Member of Another Class

Entity is associated with various attributes. Sometimes, associated data can be another entity. e.g. Entity **Identity card** contains name, roll number, permanent address. Permanent address being a separate entity, it's advisable to define it as a separate class and declare its object in the definition of IdentityCard class as below.

Program 9.5:

```
class Address:
    def __init__(self,lane, city, colony, pin_code):
        self.lane_no=lane
        self.city=city
        self.colony=colony
        self.pin_code=pin_code

class IdentityCard:
    def __init__(self,roll_no, name, surname, address):
        self.roll_no=roll_no
        self.name=name
        self.surname=surname
        self.address=address

#Driver function (Actual execution of program begins here)
a1=Address(189, 'Pune','col',411010)
t=IdentityCard(11, "Sameer","Mane",a1)
print('Roll Number = ',t.roll_no)
print('Name =',t.name)
print('Surname =',t.surname)
print('Lane Number =',t.address.lane_no)
print('City =',t.address.city)
print('Colony =',t.address.colony)
print('Pin Code =',t.address.pin_code)
```

Output:

```
Roll Number=11
Name =Sameer
Surname =Mane
Lane Number =189
City =Pune
Colony =col
Pin Code =411010
```

Let's write a program that displays gender, 10th class percentage and average marks obtained by Amit, Atul and Nisha in Physics, Chemistry and Mathematics in 12th standard. Following chart shows data of the students as below.

	Gender	Tenth Percentage	Physics	Chemistry	Mathematics
Amit	M	88.5 %	97	98	96
Atul	M	89.33%	89	99	99
Nisha	F	90.00%	96	94	99

Program 9.6:

```
class Average:
    def __init__(self, Tenth_Percentage, Gender, Physics_Marks, Chemistry_Marks,
            Mathematics_Marks):
        self.Tenth_Percentage = Tenth_Percentage
        self.Gender = Gender
        self.Physics_Marks = Physics_Marks
        self.Chemistry_Marks = Chemistry_Marks
        self.Mathematics_Marks = Mathematics_Marks

    def calculate_average(self):
        return ((self.Physics_Marks + self.Chemistry_Marks + self.Mathematics_Marks)/3)

#Driver function (Actual execution of program begins here)
student_Amit=  Average (88.5,'M',97,98,96)
student_Atul=  Average (89.3,'M',89,99,99)
student_Nisha= Average (90.0,'F',96,94,99)
Amit_PCM_Avg = student_Amit.calculate_average()
Atul_PCM_Avg = student_Atul.calculate_average()
Nisha_PCM_Avg = student_Nisha.calculate_average()

print("Amit's gender, tenth percentage and PCM average marks are ",
    student_Amit.Gender , "," , student_Amit.Tenth_Percentage , ","
    , Amit_PCM_Avg , " respectively.")

print("Atul's gender, tenth percentage and PCM average marks are " ,
    student_Atul.Gender ,"," ,student_Atul.Tenth_Percentage ,","
    ,Atul_PCM_Avg , " respectively.")

print("Nisha's gender, tenth percentage and PCM average marks are " ,
    student_Nisha.Gender ,",", student_Nisha.Tenth_Percentage ,","
    ,Nisha_PCM_Avg ," respectively.")
```

Output:

```
Amit's gender, tenth percentage and PCM average marks are M, 88.5, 97.0 respectively.
Atul's gender, tenth percentage and PCM average marks are M, 89.3, 95.66666666666667
respectively.
```

```
Nisha's gender, tenth percentage and PCM average marks are F, 90.0, 96.33333333333333 respectively.
```

After successful completion of this program 9.8, we get a new enhancement/requirement to find average marks obtained by every student in Mathematics, Science and English apart from finding average of PCM in 12th. We can define following variables to enhance the older version of the program.

```
Tenth_Mathematics_Marks
Tenth_Science_Marks
Tenth_English_Marks
```

After inclusion of these new elements in the class student, it will be

```
class student:
  def __init__(self, Twelth_Physics_Marks, Twelth_Chemistry_Marks,
    Twelth_Mathematics_Marks, Tenth_Mathematics_Marks,
    Tenth_Science_Marks, Tenth_English_Marks)
```

Further, new requirement pops up stating to find average marks obtained by student in Mathematics, Science, English in 7th standard; again, the class definition becomes lengthier. Don't you think to organize this data in a simpler manner so that it will be easier at the time of modification / enhancement of the program?

Let us define separate class element for every standard and then we can make it part of the main class i.e. student.

```
class Twelth_Subject_Marks:
  def __init__(self, Twelth_Physics_Marks, Twelth_Chemistry_Marks,Twelth_Mathematics_Marks)
class Tenth_Subject_Marks:
  def __init__(self, Tenth_Mathematics_Marks,Tenth_Science_Marks, Tenth_English_Marks)

class Seventh_Subject_Marks:
  def __init__(self, Seventh_Physics_Marks,Seventh_Maths_Marks, Seventh_English_Marks)
```

Q. How class object helps in writing better programs?

- It reduces number of identifiers in the program where entity(s) are involved. By creating object, one can easily use attributes associated with that class.
  If we do not use class, number of the identifiers in the program to represent an entity increases.
- Instead of passing every attribute individually to a method, a single class variable can be passed. Method can access attributes associated with an object easily. It improves program readability thereby makes easy maintenance.
- When multiple entities are involved in case of a program or project, work can be easily divided into team members based on entities.
- Class object can be defined once, and it can be used in other relevant projects with little or no modification.

9.6 Points to Remember

1. *Class* represents certain types of information related to a particular entity into a single unit.
2 *Class* is the basic of any object-oriented programming language.
2 *Class* concept helps to develop big projects systematically.
3 It is possible to define and use list of objects.
4. *Class* can contain object as its attributes.
5. *Class* cannot be compared for equality.
6 Within a given class, the member names of a class must be unique.

```
class student :
  def __init__(self, a ,b ,a)   # it is not allowed.
```

7 Different classes can have same names for its attribute types.

```
class student :
  def __init__(self, a ,b)

class faculty :
   def __init__(self, a ,b)
```

This is perfectly allowed. Object declaration of one class does not relate to object of any other class.

Questions

Q.1 State true or false.

i. *Class* stores similar or dissimilar data associated with a particular entity.
ii. *Class* is a blueprint (Master copy) to create objects.
iii. Size of the class is the size of its largest data type that it contains.
iv. Memory is allocated when the class is defined.
v. It is possible to assign object to another variable of the same class.
vi. When we define object, memory is allocated to it.
vii. We can define many objects of a class.
viii. It is possible to assign the value of one object to another object of same class.
ix. We can have nested class.
x. Class enables to write program systematically and improves readability.
xi. It is not possible to pass object as parameter to a method.
xii. Method cannot *return* object.
xiii. Class cannot be compared for equality.
xiv. One can define one class inside another class.
xv. It is possible to display the memory address of a class.

Q.2 Guess the Output of the following programs.

i.
```
class Time:
    def __init__(self,a,b,c):
        self.hour = a
        self.minute = b
        self.second = c

t = Time(10,56,55)
print(t.hour)
print(t.minute)
print(t.second)
```

ii.
```
class Time:
    def __init__(self,a,b,c):
        self.hour = a
        self.minute = b
        self.second = c

t = Time(10,56)
print(t.hour)
print(t.minute)
print(t.second)
```

iii.
```
class Time:
    def __init__(self,a,b,c):
        self.hour = a
        self.minute = b
        self.second = c

t1 = Time(10,56,55)
t2 = Time (10,22,33)
print(t1.hour)
print(t1.minute)
print(t1.second)
print(t2.hour)
print(t2.minute)
print(t2.second)
```

iv.
```
class Test:
 def __init__(self,a,b):
        self.a = a
        self.b = b

t = Test("m","n"")
t.show()
```

v.

```
class Time:
   def __init__(self,hour,minute):
      self.hour = hour
      self.minute = minute
   def addTime(self,x,y):
      z = Time(0,0)
      z.hour= x.hour + y.hour
      z.minute= x.minute + y.minute
      return z

t1 = Time(1.5,4.7)
t2 = Time(1.25,6.34)
t3 = Time(0,0)
t3 = t3.addTime(t1,t2)
print("Addition hour part= " + str(t3.hour)+ " Addition min part= " + str(t3.minute))
```

vi.

```
class Test:
 def __init__(self,a,b,c):
      self.a = a
      self.b = b
      self.c = c

t = Test("m","n","o")
t.c = "C"
print(t.c)
```

Q.3 What is a class? How it helps to write better programs?

Q.4 Define a class for an entity rectangle. Needless to mention rectangle associates with length and breadth as individual data types. Create a rectangle of length 10.5 and breadth 8.6. Calculate its perimeter and print it.

Q.5 Mention any 4 entities. Identify any 4 attributes associated with each entity. Define class for an individual entity. (e.g. time is associated with hour,minute,second.)

Q.6 Differentiate.

A. class and object
B. List and class

Q.7 Write a program to calculate addition of following complex numbers. Use class to define a complex number. Also, define a method complexAdd that accepts 2 complex numbers as input parameters and return their addition as another complex number.

a) 7+9.1j; 10+8.78j
b) 2.3+5.22j;9.1+7.12j
c) 5j;1+8.1j

Q.8 Time can be represented by hours; minutes; second. Define class Time. Represent following data.

a) 5 hours;13 minutes;10 seconds

b) 2 hours;10minutes;56 seconds

Write a method that accepts object of data type Time, convert time in seconds and return it to the calling method.

Q.9 What is a nested class? Give 2 examples. Define them.

Answers:

Q1.

i. True **ii.** True **iii.** False **iv.** False
v. True **vi.** True **vii.** True **viii.** True
ix. True **x.** True **xi.** False **xii.** False
xiii. True **xiv.** True **xv.** True

Q2.

i. 10
56
55

ii. Compilation Error

iii. 10
56
55
10
22
33

iv. Compilation Error

v. Addition hour part = 2.75
Addition min part= 11.04

vi. C

Core Java

for

Everyone

by

Madhusudan Mothe

Sample Reference Copy:
http://bit.ly/javacareerbook

Now-a-days, Python has become trending and first choice languages in the IT world. This book is written by Madhusudan Mothe provides an easy way to understand the core concepts of python. Basic programs are explained with the same Cuboid metal example throughout the book which is an interesting feature of this book. In a nutshell, any novice can learn Python with the help of this book. The Frequently Asked Questions (FAQ) and the 'Points to Remember' at the end of each chapter makes the subject very easy to revise and makes learning interactive. Beginner who will start the journey of learning Python via this book will definitely acquire sound knowledge of Python.

Dr. D. R. Nandanwar
Joint Director, Technical Education, Maharashtra

Python has become most demanding language these days in the IT industry because it's not only easy to write programs using Python, but also, Python provides huge library (modules). Usage of Python has been growing since its development. This book "Core Python for Everyone" by Madhusudan explains basic concepts with real time examples to the depth of the concepts. Study of this book will not only make reader strong in Python, but also in object oriented programming. This book can surely serve as a text book to the students as well as a reference book. This book can be used by a wide range of readers, right from budding programmers to practicing professionals. Happy Python Programming!

Prof. C. T. Kunjir
Deputy Director, Technical Education, Maharashtra

About the Author

Mr. Madhusudan Mothe is Bachelor of Engineering from Government College of Engineering, Pune (COEP) and M. Tech. from Indian Institute of Technology (IIT), Bombay. His all-India GATE rank was 46. His books on C and C++ are part of the syllabus of various universities. In all his books, he has explained complicated concepts in easily understandable language. He is currently working as a Senior Technical Manager in Edgeverve, a subsidiary of Infosys. He has had the distinction of being interviewed by the Pune campus of Infosys as "The Rising Star of Pune Development Centre". **His webpage is http://bit.ly/mothemadhusudan**