BASICS OF PYTHON PROGRAMMING: A QUICK GUIDE FOR BEGINNERS

Krishna Kumar Mohbey
Malika Acharya

Bentham Books

Basics of Python Programming: A Quick Guide for Beginners

Authored by

Krishna Kumar Mohbey
Department of Computer Science
Central University of Rajasthan
Rajasthan, India

&

Malika Acharya
Department of Computer Science
Central University of Rajasthan
Rajasthan, India

Basics of Python Programming: A Quick Guide for Beginners

Authors: Krishna Kumar Mohbey and Malika Acharya

ISBN (Online): 978-981-5179-63-7

ISBN (Print): 978-981-5179-64-4

ISBN (Paperback): 978-981-5179-65-1

First published in 2023.

CONTENTS

FOREWORD

The fact that Python programming is employed across most fields contributes to its growing popularity in recent years. It is possible to generate software at any level, from the simplest of programs to a fully functional application. Even every company, institution, organization, or researcher uses Python programming in their work, albeit they may do so in various methods. Python makes it possible to develop algorithms that are both effective and efficient as well as to manage more significant amounts of data in today's world. The proposed book covers various Python programming topics, ranging from the fundamentals to more advanced techniques. This book, which serves as a helpful guide to becoming a programmer, will significantly benefit the community.

D. S. Rajput

School of Information Technology and Engineering
Vellore Institute of Technology
Vellore, India

PREFACE

It gives immense pleasure to bring the book "**Basics of Python Programming: A Quick Guide for Beginners**." The most extensively used programming language today is Python, which also happens to be the most popular programming language. Students enrolled in various classes who can efficiently use this high-level programming language as a problem-solving tool are this Book's target audience. Python is not only employed in the field of computer science; instead, it is used in the development of applications for all areas.

Python programming can be swiftly learned by anybody with a fundamental understanding of computers and the ability to reason logically. Because of this inspiration, we could write this Book clearly and concisely. After reading the Book, you will have a fundamental understanding of how to do programming in Python. We have attempted to present the intricacies of Python in a very colloquial language such that the potential readers require no special expertise to refer to the book. It is apt for beginners as the concepts are explained in simple language with suitable demonstrative examples to facilitate both theory and practical learning.

Our primary goal in writing this book is to provide an approachable resource for beginners new to programming or with limited coding experience. We understand that learning a programming language can be intimidating, especially for those starting from scratch. With "Basics of Python Programming: A Quick Guide for Beginners", we have consciously designed the content to be beginner-friendly, focusing on simplicity and clarity of explanations. We believe that our book's accessible style will empower beginners to grasp the fundamental concepts of Python programming swiftly and confidently.

We recognize that many aspiring programmers are looking for a resource that allows them to learn Python quickly and efficiently. While there are extensive books available that provide in-depth coverage of Python, our book takes a different approach. We have distilled the core concepts and essential components of Python programming into a concise guide that can be absorbed quickly. By focusing on the basics, we aim to provide beginners with a solid foundation in Python programming without overwhelming them with excessive information.

We firmly believe that practical application is critical to mastering Python programming. Therefore, our book emphasizes hands-on learning and incorporates numerous practical examples throughout the chapters. By engaging in coding exercises and mini-projects, readers can actively apply the concepts they learn, solidifying their understanding of Python. Our approach encourages learners to gain practical experience alongside theoretical knowledge, enabling them to build their coding skills from the very beginning.

We firmly believe that "Basics of Python Programming: A Quick Guide for Beginners" offers a unique value proposition to individuals looking to kick-start their journey in Python programming. Its simplicity, efficiency, hands-on approach, clear progression, and supplementary resources set it apart from other books. We hope this book is a valuable tool for learning Python programming and unlocks the door to exciting possibilities.

Krishna Kumar Mohbey
Department of Computer Science
Central University of Rajasthan
Rajasthan, India

&

Malika Acharya
Department of Computer Science
Central University of Rajasthan
Rajasthan, India

CHAPTER 1

Introduction to Python

Abstract: Python is an object-oriented programming language that can support a wide range of applications like web development, desktop applications, *etc*. Its general-purpose programming features make coding easy, comprehensive and readable even for beginners.

Keywords: Development environments, Object-oriented programming, Operating system, Shuffle exchange network, Virtual machine.

INTRODUCTION

Python has grown to be a mainstream language. Its simple syntax and comprehensible code make it a popular general-purpose programming language among developers, engineers, and amateurs with limited programming skills. Its open-source features with a wide variety of libraries facilitate efficient programming. The first chapter of this book provides a brief introduction to Python along with a stepwise guide to installation on Windows and Linux. It will familiarize you with the Python IDE and editors, thus paving your journey to Python programming.

Guido van Rossum, Python's Benevolent Dictator for Life, developed Python in the 1990s. After that, several versions of Python were released. With the end of life of version 2.7, currently, versions 3.6 and 3.9 are widely used. Python is an open-source project maintained by the Python Software Foundation. We can take a tour of the Python universe at www.python.org.

Python is an object-oriented language that, due to its interoperability with existing codes in C and Fortran, gears up to the developer's demands and enhances the programmer's productivity while cutting down the time consumed. Python has become the developer's choice in various fields. For example, some of the prominent areas where Python is used are NASA's Jet Propulsion Laboratory, the Lawrence Livermore National Laboratory, Shell Research Boeing, Industrial Light and Magic, Sony Entertainment, and Procter & Gamble. Python is a high-level language; thus, the programs coded in it are easy and comprehensible. Python codes execute on a virtual machine; hence, a layer of abstraction exists between the code and the executing platform. Thus, Python is an interpreted language and produces portable codes that are cross-platform executable. Fig. (**1**) depicts the execution of a simple Python program. Unlike Fortran, Python is a dynamically typed language that uses an interpreter to interpret the representative types at the run time. The layer of abstraction abstracts the underlying optimization from the code. Python binds Fortran and C libraries using an interpreter to perform intensive computation.

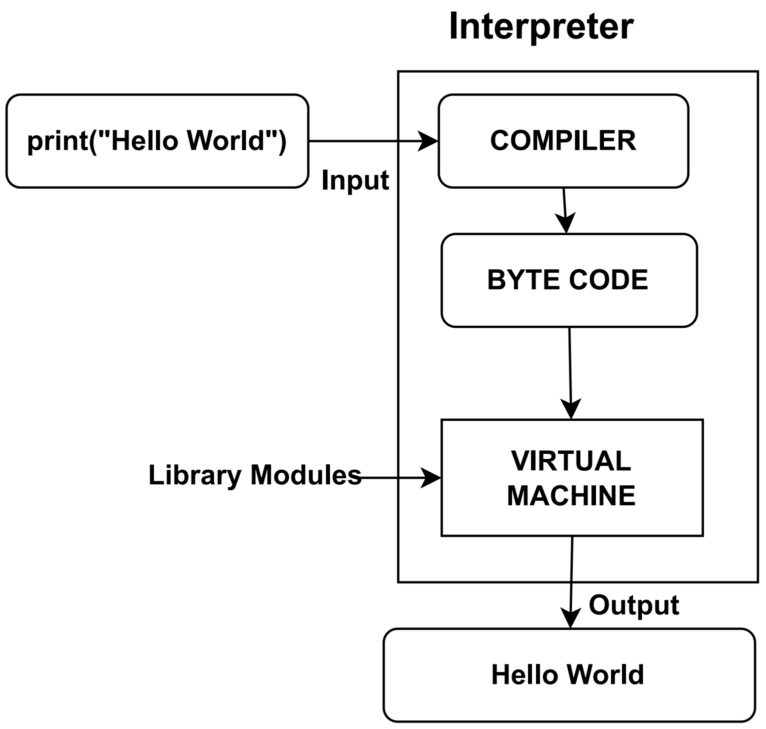

Fig. (1). Execution schematic of Python code.

You must have previously understood various languages like C++, Java, Perl, Scheme, or BASIC. All these languages are high-level languages. Nevertheless, computer hardware understands only low-level language called machine language. So, high-level language needs to be converted into machine language. For this, we have two translators: a *compiler* and an *interpreter*. A compiler is a program that translates the entire program into machine language. The high-level program is known as the source code, and the output is machine code. An interpreter is also a program that takes high-level language and converts it into machine code instruction by instruction. Compiled code can be run any number of times without repeated compiling or the source code.

In contrast, interpreted code requires an interpreter and source code every time. Thus, interpreted code makes the programming environment more user-friendly and enhances portability. Every CPU has its machine code, but we can run a high-level language program with the interpreter.

TECHNICAL STRENGTHS OF PYTHON

Portability

As already discussed, Python is a portable language that requires Byte Code for execution on any platform. An executable byte code converts the source code to platform-independent code. Python consists of a Tkinter toolkit to support the Tk GUI interface so the graphical user interface can run on all GUI's supported platforms without program changes. Python's original, standard implementation was given in ANSI C, making it executable on all major platforms ranging from PDAs to supercomputers.

Object-Oriented

Python is an object-oriented programming language. The language is extensible, *i.e.*, the programs can be extended to C, C++, and Java. It uses a class model to support polymorphism, operator overloading, and other notions. It is a powerful scripting tool for other object-oriented system languages, like C++, Java, and C#. The recent versions of Python also support functional programming. This includes generators, comprehensions, closures, maps, *etc*.

Community Support

Python community support is quite active and responsive to the user's queries. The community consists of Python creator Guido van Rossum, the Benevolent Dictator for Life (BDFL), and a crew of thousands of workers. Python is more conservative than other languages in terms of changes, *i.e.*, the changes need to be approved by the community, especially BDFL.

Advanced Features

Python provides full support for features of scripting languages like PERL. The scheme facilitates the use of software development tools that are easily found in compiled languages. It is a product of the FLOSS community, *i.e.*, Python is a Free, Libre, and Open-Source Software that assists in knowledge sharing. Some of the salient features of Python are:

CHAPTER 2

Data Types, Operators, and Expressions

Abstract: Now that you have been familiarised with the installation and basics of Python programming, it's time to dig in a little more and understand the different data types that are available along with the operators and expressions that make the programming more user-friendly. In this chapter, we will learn about data type categorization and the operators present. Here you will learn the following nuances:

1. Data types supported by the Python.
2. Use of variables to store and access the data.
3. Operators do mathematical works and logical functions.
4. Variety of expressions to serve the range of applications.

Keywords: Associativity, Literals, Operators, Precedence.

INTRODUCTION

In computer programming, a data type is a classification of the many kinds of information that may be saved in a variable. Since Python is a dynamically typed language, we do not need to specify the variable type when we declare it.

The value is bound to its type in a way that the interpreter does not explicitly specify—for example, **X=100**. We did not declare the type of the variable **X**, yet it now holds the integer value 100. In Python, the integers are by default treated to be of integer data type. To check the type of the variable, we use **type()** function available to us in Python, and it returns the type of the variable that was handed in. The following example illustrates how to specify the values for a variable and to check their types.

Example:

```
x=100
y="Python Programming"
z= 12.59
print (type(x))
print (type(y))
print (type(z))
```

Output:

```
<class 'int'>

<class 'str'>

<class 'float'>
```

Python has a wide variety of standard data types, each of which can have a unique storage mechanism defined for it. The data types defined in Python are listed below:

1. Numbers
2. Sequence
3. Dictionary
4. Boolean
5. Set

NUMBER

The number is responsible for storing numerical values. A Python Numbers data type is responsible for storing all integer, float, and complex values. To check the data type of a variable, Python has a function called **type()**. It also has the **isinstance()** function that checks whether or not an object belongs to a specific class. When a number is assigned to a variable in Python, Number objects are created automatically. A number data type can store int, float, and complex type values.

- The integer value can be any length. There is no limitation on the length of an integer in Python.
- Float is used to store floating-point numbers which is accurate up to 15 decimal points.
- A complex number is specified in the form of **a + ib**, where **a** and **b** stand for the real and imaginary components of the number, respectively.

Example:

```
a = 10
print ("The type of a is", type(a))

b = 10.15
print ("The type of b is", type(b))

c = 2+4j
print ("The type of c is", type(c))
print ("c is a complex number", isinstance(2+4j,complex))
```

Output:

```
The type of a is <class 'int'>

The type of b is <class 'float'>

The type of c is <class 'complex'>

c is a complex number True
```

Sequence

The language treats Python's string, list, and tuple datatypes as sequence types. The character sequence enclosed in quote marks is an example of the string, which may be defined as that sequence. When defining a string, Python allows us to use single quotes, double quotes, or triple quotations. As Python includes predefined functions and operators that may be used to execute various actions on strings, managing strings in Python is a simple process. The *"Hello World!"* is the result of the operation *"Hello"* + *"World!",* which uses the plus (+) operator to concatenate the two strings. Similarly to repeat the strings we can use the repetition operator (*). For example "India" *7 outputs IndiaIndiaIndiaIndiaIndiaIndiaIndia '.

Example:

```
S1= "Hello World"
print (S1)
```

Output:

```
Hello World
```

CHAPTER 3

Control Flow

Abstract: It's now the time for the readers to acquaint themselves with the control flow in the programming. So far the users have seen the linear path of execution, where the execution commences from the top and moves sequentially to the bottom. Further, our world is filled with tasks required to run in loops, for example in banking applications, unless the user enters the correct password the system prompts the dialogue “Please enter the correct password”. In this chapter, we introduce the concept of control flow through which the users can decide the execution path for the program and the looping constructs to iterate through the tasks. Our key takeaways from this chapter are listed below:

1. Understanding the decision control statements.
2. Programming with *if-else* ladder and its variants.
3. A hand on *for*, and *while* loop statements.
4. Understanding the control flow alterations the *jump* statements like *break*, *continue* and *pass*.

Keywords: Control flow, Conditional processing, Looping constructs.

INTRODUCTION

The control flow of a program is illustrated by a statement known as a control flow statement. In addition, it determines the sequence in which the code of the program is executed. Conditional statements, loops, and function calls are the primary mechanisms that direct how a Python program executes its instructions. Python programming utilizes three distinct types of control structures in its many applications.

1. Sequential statements
2. Decision control statements
3. Looping statements

SEQUENTIAL STATEMENTS

It uses a default mode because the control will move line by line in a program. Moreover, it is a series of statements that are executed in a sequence.

Example:

```
# Python Program to find the area of a triangle
a = 5
b = 6
c = 7
# calculate the semi-perimeter
s = (a + b + c) / 2
# calculate the area
area = (s*(s-a)*(s-b)*(s-c)) ** 0.5
print ('The area of the triangle is %0.2f' %area)
```

Output:

```
The area of the triangle is 14.70
```

DECISION CONTROL STRUCTURES

There are a few other names for a decision control statement, including a selection control statement and a branching statement. The condition serves as the foundation for the selection statements. If a condition is met, then the statement will be carried out. In addition to that, we utilize it as a checking and testing mechanism. There are wide distinct varieties of decision control statements, including the following:

- *if*
- *if-else*
- *nested if*
- *if-elif-else*

If Statements

If statements are utilized in programming to determine whether or not a particular section of code should be executed. If a condition is met, the action will be successful; otherwise, it will not.

Syntax:

```
if condition:

# Statements to execute if

# Condition is true
```

The flow diagram for the if statement is shown in Fig. (**1**).

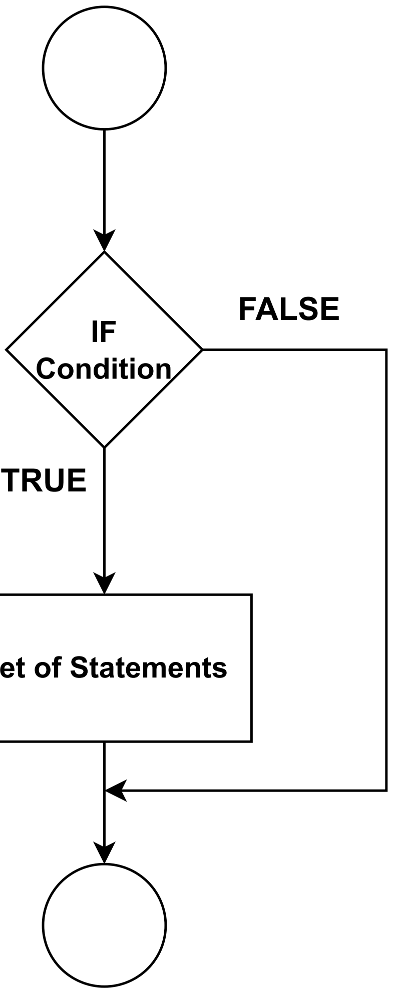

Fig. (1). Flow diagram of if statement.

Example:

```python
# Program to check whether a number is positive or Negative
n = 10
if n > 0:
    print (n, "is a positive number.")
n1 = -2
if n1<0:
    print (n1, "is a Negative number.")
```

CHAPTER 4

Functions

Abstract: Functions are one of the primary concepts in every programming language. They provide an easy way to package the programming logic and use it as and when required as many times at any place. Thus they help to reduce the redundancy in code and increase the reproducibility. With the increase in the length of code, it is often a good idea to divide the code into separate modules by splitting the code into different functions based on its utility. This is a much- sought practice to organise the lengthy code. They also help in unit testing the code as testing small units in isolation is quite an easy task. This deliberate use of functions thus supports language flexibility and a user-friendly interface. In this chapter the pertinent takeaways would be:

1. Understanding *functions* and *function calls*.
2. Comprehending the concept of *local* and *global* variables.
3. Programming with the recursive functions.

Keywords: Reproducibility, Redundancy, Reusability, Recurison.

INTRODUCTION

The code may be organized, made more understandable, reused, and repurposed with the use of functions, which are a helpful approach to separate the code into more manageable portions. In addition, a function is a piece of code that is put into action when the function itself is invoked. It can take in data in the form of a parameter or argument and then return the result.

Definition

A function is an area of code that comprises a block of statements that carry out a particular operation. When creating a function, there are a few fundamental guidelines to follow:

1. Function blocks start with the **def** keyword. After that, the **function name** and parentheses (**()**) are used.
2. An argument or parameters should be passed inside the parentheses.
3. Any function's code block begins with a colon (:) and is indented.

Syntax:

```
def function_name(parameters):
  Statement 1
  Statement 2
  Statement 3
        -
        -
        -
   Statement n
  return [expression]
```

There are two types of functions, as described below:

- Predefined functions
- User-defined functions

PREDEFINED FUNCTIONS

It is sometimes referred to as a built-in function because Python has its functionality established in the language. The Python interpreter has several built-in functions that are always available for usage. There are many kinds of built-in functions, and here is an example of some of those built-in functions (Table **1**):

Table 1. Predefined functions.

abs()	It returns the absolute value of a number
bin()	It returns the binary version of a number
float()	It returns a floating-point number
hex()	It converts a number into a hexadecimal value
int()	It returns an integer number
len()	It returns the length of an object
list()	It returns a list
max()	It returns the largest item in an iterable
min()	It returns the smallest item in an iterable
oct()	It converts a number into an octal
pow()	It returns the value of x to the power of y
print()	It prints to the standard output device
range()	It returns a sequence of numbers, starting from 0 and increments by 1 (by default)
round()	It rounds a numbers

Let's look at an example of each of the built-in functions discussed so far to better grasp how they should be used.

Example:

```
#Return the absolute value of a number
x1 = abs(-5.37)
print (x1)

# Return the binary version of 24
x2 = bin(24)
print (x2)

#Convert the number 7 into a floating-point number
x3 = float(7)
print (x3)

#Convert 321 into hexadecimal value
x4 = hex(321)
print (x4)

#Convert the number 7.2 into an integer
x5 = int(7.2)
print (x5)

#Return the number of items in a list
list1 = ["Malika", "Krishna", "William", "Sam"]
x6 = len(list1)
print (x6)

#Return the largest number
x8 = max(245, 433)
print (x8)

#Return the lowest number
x9 = min(343, 434)
print (x9)

#Convert the number 12 into an octal value
x10 = oct(32)
print (x10)

x11 = pow(5, 5) #Return the value of 5 to the power of 5 (same as 5 * 5 * 5 * 5)
print (x11)

#Print a message onto the screen
Print ("It is a message")

#Create a sequence of numbers from 0 to 7, and print each item in the sequence
x12 = range(8)
for n in x12:
  print (n)

#Round a number to only two decimals
x13 = round(6.66543, 2)
print (x13)
```

CHAPTER 5

Sequence-String and List

Abstract: Unlike primitive data types like integers, floats and Boolean, a string is an ordered sequence of characters each of which can be accessed easily. Further, one of the important built-in data types of Python is lists. Lists and strings share many similarities like, lists are a sequence of values. List indices work similarly to string indices. But unlike strings lists are mutable. In this chapter, we introduce the core concepts of lists and strings and several operators that are used to make programming with these user-friendly.

Keywords: Mutable, Slicing, Ordered collection.

INTRODUCTION

When we want a collection of characters that are pretty like one another, we have to use a sequence of characters. For instance, if you want to keep a record of your name in the computer's memory, you will need a variable capable of storing your name. However, it is necessary to have a series of characters because the name is a collection of characters. This string consists of nothing more than a succession of characters.

Python's equivalent of the sequence data type is called a list. It is the most effective and may be expressed as a list of values delimited by commas and enclosed in square brackets. A list is frequently utilized to store the sequence of various kinds of data. The list is changeable, which indicates that its elements can be altered after the list has been constructed. This chapter discusses a variety of operations that may be carried out on a list's elements.

STRING

A series of characters is referred to as a string. Python relies heavily on this idea to function correctly. There are a few key aspects to consider about string.

- Strings are amongst the most popular types in Python.
- It can create them simply by enclosing characters in quotes (single, double, or triple).
- Python treats single quotes the same as double quotes.
- A string is a sequence of Unicode characters, and a character is simply a symbol.

To generate a list of names, it must type each name surrounded by quotation marks, such as “Krishna”. The string may also be assigned to a variable to carry out additional operations and make further use of that string. Let's start with the most fundamental examples to grasp the idea of string.

Example:

```
str = "This is a string."
print (str)
```

Output:

```
This is a string
```

Python does not have a character data type in its standard library. If we consider a single character to be a string, then the length of that string would be one. You may access the string's constituent components using square brackets ([]).

Example:

```
str = "Python"
print (str[0])
print (str[1])
print (str[2])
print (str[3])
print (str[4])
print (str[5])
```

Output:

```
P
y
t
h
o
n
```

The string can use to loop through character by character.

Example:

```
for i in "Krishna":
  print (i)
```

Output:

```
K
r
i
s
h
n
a
```

Other concepts about string include the *len()* function for getting the length of a string, keyword *in*, and *not in* for checking whether the substring or character is present in a string.

Example:

```
#print a length of a string
str = "Python"
print (len(str))  #String Length

#print "simple" in a string
str1= "Python is so simple"
print ("simple" in str1)

#Print only if "simple" is present in a string:
str2 = "Python is so simple"
if "simple" in str2:
  print ("Yes, 'simple' is present.")

#print "simple" is not in a string
str3 = "Python is so simple"
print ("programming" not in str3)
```

Output:

```
6
True
Yes, 'simple' is present.
True
```

CHAPTER 6

Tuple and Dictionaries

Abstract: Another important data type of Python is dictionary. In the previous chapter, we acquainted the readers with the lists and strings. In this chapter, we shall discuss dictionaries and lists. lists are the ordered collection of objects but dictionaries are an unordered collection of objects. dictionary values are referred to using key-value pair instead of positional offset. Due to this, they have found great usage in search tables, records and aggregation. Another concept introduced in this chapter is tuples. These are immutable like strings and represent a stable collection of arbitrary items.

Keywords: Hash tables, Immutable and mutable, Mappings.

INTRODUCTION

One of Python's most significant data types is called the tuple, which holds several components as an object. It is also known as an immutable data type, which indicates that its members cannot be altered in any way after they have been set. A dictionary is another data type that may be used in Python, and it functions similarly to tuples and lists. An associative array is another name for this structure. A dictionary is a collection of “key-value pairs”. Each key-value pair connects the key to the value that corresponds to it. In this chapter, you will learn about various operations and attributes that can be used with dictionaries and tuples. In addition, the many features and actions of dictionaries and tuples are broken down and illustrated using examples and codes.

TUPLE

A tuple is a group of ordered, immutable items arranged in a specific way. Tuples are structured similarly to lists and strings. In this context, the definition of immutable is that the components of the tuple are not subject to change. Once a tuple has been generated, it is not possible to add or remove items from the tuple. Even we cannot change the order in which the tuple members are presented. In addition, the length of the tuple cannot be altered. It is necessary to generate a new tuple if we wish to modify an existing one by adding or taking something away from it. Unlike lists, tuples are denoted by parentheses and cannot be edited. Tuples are a sort of sequence equivalent to strings in terms of structure. Tuples can hold any components, in contrast to strings, which can only store characters. It indicates that the tuple contains either a list of students' names or employee IDs, depending on which one is selected. Tuples may also be used to store varied elements, implying that a single tuple can contain components of several data formats, such as decimal formats, integers, and characters. A sequence of music files, picture files, and other data types can also be stored in tuples.

To create a tuple in Python, all the elements are enclosed in () parenthesis, separated by a comma. A tuple can store heterogeneous data elements. Below are examples of creating tuples.

Example:

```
# Create an empty tuple
Tuple1 = ()
# tuple with 5 elements
Tuple2 = (10,20,30,40,50)
print (Tuple2)
# tuple of character elements
Tuple3 = ('x', 'y', 'z')
print (Tuple3)
# tuple of strings
Tuple4 = ("Python", "Programming", "Book")
print (Tuple4)
# tuple of heterogeneous data elements
Tuple5 = (100, 20.123, "Python Programming")
print (Tuple5)
# tuple of string and List
Tuple6 = ("Python Book", [10, 20, 25])
print (Tuple6)
```

Output:

```
(10, 20, 30, 40, 50)

('x', 'y', 'z')

('Python', 'Programming', 'Book')

(100, 20.123, 'Python Programming')

('Python Book', [10, 20, 25])
```

To create a tuple of a single element, it should be followed by a comma. The following example creates a tuple of a single element.

Tup1 = (100,)

In the above case, if we do not put a comma after 100, then Python would treat **Tup1** as an integer rather than a tuple variable.

Example:

```
Tup1 = (100,)
type (Tup1)
Tup2=(100)
type (Tup2)
```

Output:

```
<class 'tuple'>

<class 'int'>
```

The*Tuple ()*Function

Python has a built-in function called ***tuple()*** that may be used to make tuples. While we can build tuples without utilising this function, it offers a different method. The ***tuple()*** function is used to construct a tuple in the following example.

Example:

```
# example of tuple() function
# creating an empty tuple
T1 = tuple()
print (T1)
# creating a tuple from List
T2 = tuple([1,2,3,4,5])
print (T2)
# creating a tuple from strings
T3 = tuple("Python Programming")
print (T3)
```

Output:

```
()

(1, 2, 3, 4, 5)

('P', 'y', 't', 'h', 'o', 'n', ' ', 'P', 'r', 'o', 'g', 'r', 'a', 'm', 'm', 'i', 'n', 'g')
```

CHAPTER 7

File Handling

Abstract: From the previous chapters, we anticipate that the readers have garnered enough working knowledge of Python and its elementary concepts. In this chapter, we introduce to the user the concept of files. These are one of the major built-in object types in Python. We can create, call, work, and close the files using several functions as enunciated below. One of the primary tasks of files is method exporting and common file-processing tasks such as input and output display to external files, flush buffers, *etc*.

Keywords: Buffering, Directory path in OS, Storage in OS.

INTRODUCTION

When information must be stored in a file in an unalterable manner, file management is essential. A file is a designated location on the disc where data is kept that is important to the purpose of the file. Once the program has been closed, we will be able to retrieve the information that was previously stored. The concept of file management has been adapted and implemented in various other languages. However, doing so can be difficult or time-consuming, depending on the language.

However, unlike other Python ideas, this one is not complicated and can be understood easily. You will learn various theories and procedures about the management of files in this chapter. In addition, we have illustrated a variety of file actions by using examples and programs to illustrate our points.

FILE

A collection of bytes that may be used to store information is called a file. This data is structured in a specific format, which might be anything from a straightforward text file to an intricate program executable, depending on its complexity. In the end, these byte files are changed into binary, consisting of 1s and 0s so that the computer can process them more quickly. Much like many other programming languages, Python can handle files, allowing users to read, write, and execute many other file-related operations. This functionality is known as file handling. It is essential to understand that Python processes files differently based on whether they contain text or binary data. Each line of code is made up of a string of individual characters that, when combined, make up a text file. A special character terminates each line in a file referred to as the EOL (End of Line) character. Examples of EOL characters include the comma and the newline character. It signals to the interpreter that the line being read has finished and that a new line has begun. On most of today's file systems, files are broken up into three distinct parts:

- *Header:* details about the file's contents (file name, size, type, and so on).
- *Data:* the file's contents as written by the author or writer.
- *End of file:* a unique character that denotes the file's termination.

In programming, it is possible that a specific piece of input data must be created more than once. Occasionally, it is insufficient only to display data on the console. Large amounts of data may be presented. The console can only display a certain amount of data; as memory is volatile, it is difficult to restore programmatically created data frequently. The local file system, which is volatile and always available, is where we can store things if we need to. The file-handling capabilities of Python must be used for this. We can use our Python application to create, modify, read, and destroy files on the local file system, thanks to file management in Python. When accessing a file on an operating system, a file path is necessary. A string indicates the location of a file called a file path. There are three main sections in it:

- **Directory Path:** the location of a file or folder on a file system, separated by a forward slash (/) in Linux or Unix or a backslash (\) in Windows.
- **File Name:** the file's real name.
- **Extension:** it defines the file type.

Let's say you needed to open the T1.txt file, and the position you were in now was the same as the **location**. You must first travel to the **Location** folder, then to the **Folder1** directory, and then to the **T1.txt** file to access the file. The path of this file is "***Location/Folder1/T1.txt.***"

Open() Function

It must be open first to perform reading and writing operations on a file. To open a file in Python, the user must first create a file object associated with a physical file. In addition, the ***open()*** function is used to open a file in Python. Python's open() function takes two arguments: the file name and the access mode. The function returns a file object, which can be used for reading, writing, and other operations. The below syntax is used to open a file in Python.

Opening a file is a prerequisite for performing any action on it, including reading and writing. To access the contents of a file using Python, the user must first build a file object corresponding to the file's actual location. In Python, opening a file is also accomplished with the help of the ***open()*** function. Python's ***open()*** function requires two pieces of information before it can be called: the *file name* and the *access mode*. The method will return a file object that may be utilized for various tasks like reading, writing, and others. In Python, opening a file is accomplished with the syntax that is presented below:

Syntax:

```
File_object_Name = open(<file_name>, <access_mode>, <buffering>)
```

Various modes, such as read, write, and append, are available for accessing the files. The access mode to open a file is defined in Table **1**.

Table 1. File Access Modes.

R	The file is opened in read-only mode. The file pointer is present at the start. If no access mode is defined, the file is opened in this mode by default.
rb	It converts the binary file into a read-only mode. The file pointer is present at the start of the file.
r+	It opens the file for both reading and writing. The file pointer is present at the start of the file.
rb+	It opens the file in binary format for reading and writing. The file pointer is present at the start of the file.
W	It only allows you to write to the file. If a file with the same name already exists, it is overwritten; otherwise, it is created. The file pointer is present at the start of the file.
wb	It opens the file so it can only be written in binary format. If the file already exists, it is overwritten; otherwise, it generates a new one. The file pointer is present at the start of the file.
w+	It opens the file for both writing and reading. It differs from r+ in that it overwrites the previous file if one exists, while r+ leaves the previously written file alone. If no such file exists, it creates one. The file pointer is present at the start of the file.
wb+	It opens the file in binary format for both writing and reading. The file pointer is present at the start of the file.
A	The file is opened in append mode. If there is one, the file pointer is at the end of the previously written file. If no file with the same name exists, it creates a new one.
ab	It opens the file in binary format in append mode. The pointer is at the end of the file that was previously written. If no file of the same name remains, it produces a new binary file.
a+	It opens a file for both appending and reading. If a file exists, the file pointer stays at the end of it. If no file with the same name exists, it produces a new one.
ab+	It opens a binary file for appending and reading. The file pointer is already at the file's end.

CHAPTER 8

Exception Handling

Abstract: In this chapter, we introduce the concept of exception handling in Python. They are used to specify the alternate sequence of actions the program needs to jump to at the occurrence of the event. For example, if we want to print several pages from the printer and somewhere in the middle of the job the paper gets stuck in the printer. In such as situation we would want to jump to the function that aborts the printing and handles this situation by instant shut down of the printer. In such events comes the exception handling. When the program jumps to the exception handler part the current sequence of commands is abandoned and the commands given to the exception handler are executed. After the exception is tackled the programming returns to the point where the marker left.

Keywords: Error handling, Event notification.

INTRODUCTION

Error handling makes your code more robust by shielding it from the kinds of mistakes that may result in an abrupt shutdown of your application. On the other hand, Python exceptions can be handled in contrast to errors. Errors can be syntax errors, and although various exceptions might happen during execution, they aren't always unusable. An error could be a syntax mistake. An application that is reasonable and well-designed should avoid the critical issues indicated by an Error.

In contrast, an application that is suitable and well-designed should try to capture the conditions that are indicated by an Exception. Programmers should avoid handling errors wherever possible since they are a form of uncontrolled exception that cannot be recovered from. An example of this type of error is the **ZeroDivisionError**. Think about what would happen if you produced code that was later used in production but still ended because of an error. Because the customer would be dissatisfied, handling the exception in advance and eliminating uncertainty is preferable. There are two different kinds of mistakes, the first being syntax errors and the second being exceptions. A syntax error will occur when the parser identifies a grammatical problem in your code. Syntax errors are also commonly referred to as parsing errors. To better understand it, let's look at an example.

Example:

```
#Syntax error example
x = 1
y = 2
z = x y
```

Output:

```
File "<string>", line 3

z = x y

^

SyntaxError: invalid syntax
```

The arrow in the report shows that the code parser ran into an error when executing the program. The failure may be traced back to the token before the arrow. Because it will output the file's name and the line number where the issue occurred, Python will handle a significant portion of the troubleshooting work for you when attempting to resolve errors of this type.

An error that arises as a result of the execution of a program is referred to as an exception. Exceptions are occurrences that do not obey a general rule, which is how non-programmers understand the term. When a statement or expression is executed, an error through the syntax may occur. Python's exceptions are faults that may be seen during the execution of the program but are not always catastrophic. An exception object is created whenever a Python program generates a runtime error. The program will be terminated unexpectedly and without warning if the code does not expressly handle the exception. In most cases, programs will disregard exceptions, which will lead to error messages such as the following:

Example:

```
# Example of type error
name= 'David'
age=45
age+name
```

Output:

```
Traceback (most recent call last):
  File "<string>", line 3, in <module>
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

Another example of division by zero exception can be seen in the below example.

Example:

```
#Example of division by zero exception
10 * (15/0)
```

Output:

```
Traceback (most recent call last):
  File "<string>", line 1, in <module>
ZeroDivisionError: division by zero
```

Python exceptions can take several forms, indicated right next to the message they produce. For example, the types of exceptions that have just been discussed are **TypeError** and **ZeroDivisionError**. Both error messages include, the kind of exception, and the name of the Python built-in exception that was encountered. The remaining half of the error line comprises information about what led to the mistake; the specifics of this information are determined by the type of exception that was thrown.

HANDLING EXCEPTIONS

Python's approach to managing exceptions is similar to Java's. The code that might result in the throwing of an exception is contained within a **try block**. In Java, exceptions are handled using catch clauses, but in Python, exceptions are handled by sentences inserted using the **except** keyword. Personalized deviations from the norm can be accommodated if necessary. By utilizing the “raise” command, it is possible to coerce the occurrence of an exception. You may safeguard your application by enclosing any potentially malicious code that can cause an exception in a try: block. This will prevent the exception from being thrown. Place an except declaration after the try: block, then immediately after that, a block of code that fixes the problem in the most elegant way possible should follow it. The following is an example of the syntax for **try...except...else** blocks:

CHAPTER 9

Modules and Packages

Abstract: Useful codes are often stored as separate files to increase modularity and reusability. Modules refer to a single file of code while a package is a collection of modules. A good programmer utilises both these aspects to enhance the program view and manage the hierarchy. In this chapter, we introduce the basics of working with the modules and packages.

Keywords: Scoping, Modular programming, Standalone script.

INTRODUCTION

MODULES

A complicated and unmanageable program can be broken down into several manageable subprograms, each referred to as a **module** through modular programming. One activity can be carried out by utilizing each component individually. The creation of modules in Python may be accomplished by utilizing Python files, which can include a variety of *and* statements. It is possible to define variables, classes, and functions in a module. A module can also make use of code that is executable. When the code is divided into modules, it is much simpler to comprehend and more convenient to utilize. Additionally, it logically arranges the code.

To put it another way, the file containing our Python source code with the extension (.py) is considered the module. Python modules can store code that can be executed. When we want to use the features of one module in another, we must import the specific module first. Let's say you've generated a file on your computer named **module1.py**, and inside it, you have the following code:

Filename: module1.py

```
#(module1.py)
def add_value (x, y):
 return (x+y)
def sub_value (x, y):
 return (x-y)
```

To call the functions *add_value()* and *sub_value()* specified in the module named file (**module1.py**), we must include this module in our main module. To use the module's functionality, we must first load it into our Python code. Python has ***import*** and ***from..import*** statement to include a module.

Import Statement

Our Python program can connect to a module by utilizing an import statement. The import of many modules may be accomplished with a single import line; however, despite the number of times a module has been imported into our register, it is only loaded once each time. The syntax of the import statement is as follows:

Syntax:

```
import module1, module2, module3, ........ module n
```

When an import statement is found the interpreter imports the module specified within the search path. The interpreter explores every directory in the search path when importing a module. For instance, add the following line to the program's top to import the module **module1.py**.

Example:

```
# importing module module1.py
import module1
print (add_value (2, 4))
print (sub_value (10, 50))
```

Output:

```
6

60
```

From…Import Statement

Python enables users to import only a module's specified properties into the namespace rather than the entire module. For this, the from…import statement may be utilized. The following syntax makes use of the ***from...import*** expression.

Syntax:

```
from <module_name> import <name_1>, <name_2>….<name_n>
```

Consider the following module, **module 1**, which includes the functions *add_value()* and *sub_add()*.

Example:

Filename: module1.py

```
# module1.py
def add_value(x, y):
 return (x+y)
def sub_value(x, y):
 return (x-y)
```

If we want to import only *add_value()* function from this module, then the following code will be used.

Filename: main.py

```
# importing add_value()from module1.py
from module1 import add_value()
print (add_value(100, 200))
```

Output:

```
300
```

We can also import any built-in module in our program. The below example imports the **pi** function from the *math* module.

Example:

```
#importing pi from the math module
from math import pi
print ("pi value=", pi)
```

CHAPTER 10

Object-Oriented Programming

Abstract: In this chapter, we explore the OOP concepts in programming that offer an effective way of making coding more easy and comprehensive. It facilitates redundancy and allows customizing the existing code.

Keywords: Namespace, Superclass and Subclass.

INTRODUCTION

Object-Oriented Programming, sometimes called OOP, organizes a computer program's components into distinct objects with similar characteristics and functions. Classes and objects are the fundamental building blocks of object-oriented programming. The class serves as the blueprint, while the objects themselves are living, breathing entities capable of carrying out various operations. An object is composed of its data elements, characteristics, and behaviour, which may include actions or functions.

In procedural programming, a program is structured similar to a recipe by providing several stages, including functions and code blocks, that flow sequentially to achieve a goal. This programming paradigm is one of the most common programming paradigms. Python has always been object-oriented, much like other languages designed for general-purpose programming. It makes it possible for us to construct programs using an object-oriented methodology. Python makes it very simple to create and work with objects and classes. An object-oriented paradigm refers to building software by utilizing classes and objects. The item is connected to things in the real world, such as a computer, a house, a mobile phone, and so on. The definition of OOPS emphasizes the creation of code that may be reused. Putting together new things to use as solutions is a frequently utilized strategy. An object-oriented programming paradigm can be broken down into its core ideas, which are as follows.

CLASS AND OBJECT

A **class** is a name that can be given to a collection of objects. It is a logical entity with specific methods and unique properties. For example, a student's class should contain an attribute and a method, such as their name, age, address, and the classes they are enrolled in. The **object** functions as its independent entity, complete with a state and a set of behaviors. It may be a notebook, computer, pencil, or other things. In Python, everything is an object, and nearly anything may have both attributes and methods applied to it. When a class is defined, the class must first create an object before it may assign memory.

DATA ABSTRACTION

The practice of abstraction is a method that hides information on the system's internal workings and shows only its functionalities. Data encapsulation and data abstraction are two terms that are frequently used interchangeably. Because data encapsulation is the means through which data abstraction is achieved, it is possible to use either word interchangeably.

ENCAPSULATION

Encapsulation is a key notion in object-oriented programming, one of today's most prevalent programming paradigms. It explains the concept of enclosing the data and the methods that operate on the data within a single unit. This limits direct accessing methods and variables, which helps avoid unintentional data change. Only the object's method can change the object's variables; this is done to prevent unintentional changes. These particular variables fall under the category of “private variables.” Encapsulation can be illustrated by how a class stores all information about its member functions, variables, and so on. The objective of information hiding is to ensure that the state of an object is always valid by regulating access to its attributes while keeping those properties hidden from the view of the outside world.

INHERITANCE

Inheritance is the most fundamental component of object-oriented programming, miming the inheritance process that occurs in real life. Inheritance is an essential component. It states that all of the characteristics and behaviors of the parent object are passed down to the child object through inheritance. Through inheritance, we can build a class capable of taking on all of the characteristics and actions of another class. The new class is considered to be a derived class or a child class, whereas the base class or the parent class is regarded as the class whose properties are gained. It assures that the code can be utilized in different applications.

POLYMORPHISM

The term "polymorphism" originates from the combination of the words "poly" and "morphs." The prefix poly denotes "many," while the suffix morph refers to "shape." The capacity to carry out a single activity in multiple guises is what we mean when discussing polymorphism. It employs a single category of items, such as a method, operator, or object, to stand in for several different types in various contexts. For example, we have a single addition operator capable of adding a wide variety of value types.

Defining a Class

The keyword **class**, followed by the class name, is used to build a class in Python. The following is the syntax for creating a class.

Syntax:

```
class <Class_Name>

        # data members

        # member functions
```

A class declares all its attributes in a new local namespace. Data members or functions may be included as attributes.

It also contains unique attributes that start with double underscores. For example, ***__doc__*** returns the class's docstring. The following statement can access it.

```
<class-name>. __doc__.
```

A new class object with the same name is generated when we define a class. We may use this class object to access the various attributes and create new objects of that class.

CHAPTER 11

Python for Machine Learning

Abstract: Anticipating that the user has a good knowledge of the core elements of Python we now explore the applicative aspect of Python. In this chapter, we will look at Python, especially from the Machine Learning (ML) point of view. We will discuss the various libraries and their utility in ML and then lay hands over the programming demonstrations.

Keywords: Libraries, Machine learning, Packages, Prediction and classification.

INTRODUCTION

We assume the readers now have enough preliminary knowledge to dive deep into programming with Python. Python has been the developer's choice, and Machine Learning is one of the major application areas of Python. Machine Learning is the field of computer science that allows computer programs to attain the capability much like a human brain, *i.e.*, learn from past experiences and perform future tasks. In this chapter, you will learn to program Machine Learning algorithms with Python.

Anomalous to traditional programming, Machine Learning requires no pre-defined rules but the design of a mathematical model for decision-making rather than human interference. Figs. (**1** and **2**) show the disparity between the two paradigms of programming. In traditional programming, rules, and data are fed to the computer, and the results are evaluated. In case of an error, the problem is studied and analyzed, and changes are made to the rules. But in the Machine Learning paradigm, the learning is leveraged on the training data, and results are evaluated. The defined model is then tested over test data, and output is produced.

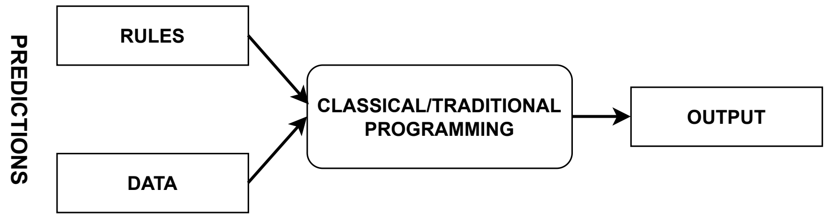

Fig. (1). Traditional Programming.

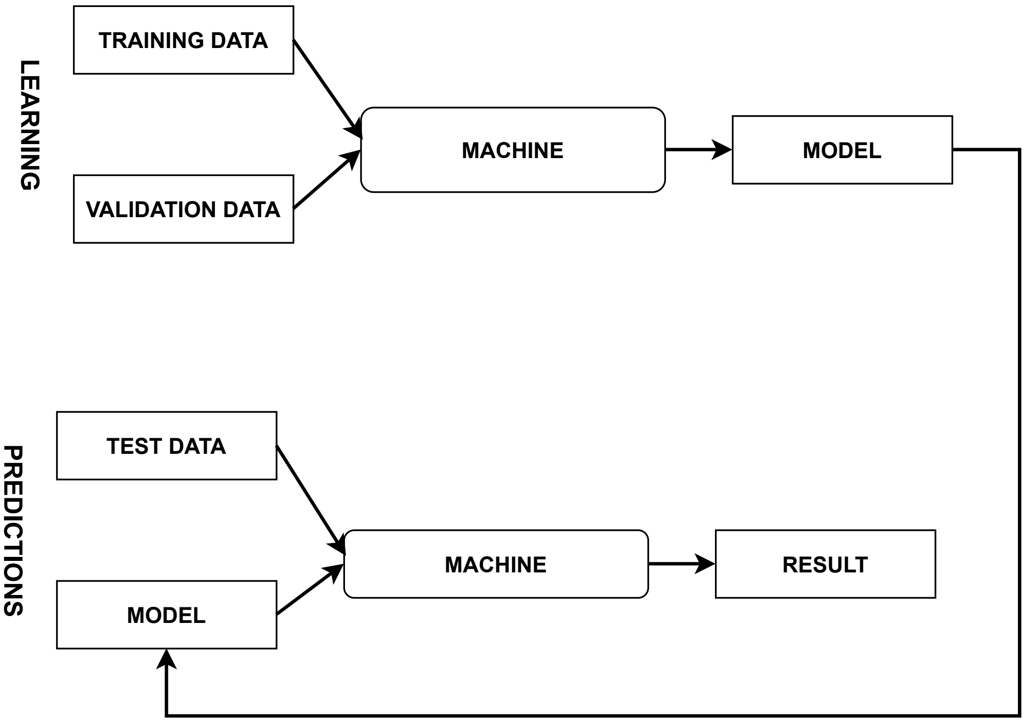

Fig. (2). Machine Learning Programming Paradigm.

Self-learning is the crux of Machine Learning. The performance based on data extraction, preprocessing, and analysis without being explicitly programmed defines the objective of Machine Learning. It aims to facilitate the machine to work directly without being programmed. Decision-making is an important task that relies on pattern extraction and information modeling based on trial and error and probabilistic reasoning. Thus, one can say that decisions are not based on the pre-set rules but on the input data. To minimize the errors, the programmers can tweak model settings called hyperparameter tuning. For learning, the programmer splits the data into training and testing data. The training set helps to learn the patterns in the data and validate the results. Finally, the developed model can be evaluated over test data. The model can be deployed for other applications if the performance is satisfactory.

Machine Learning has been divided into two categories based on the training procedure. If the machine is trained with the labeled training data, *i.e.*, the data that has been classified under different classes, then that type is called supervised learning. And, if the learning over training data is without human interference, *i.e.*, no defined classes, then that is called unsupervised learning. The difference between supervised and unsupervised learning is given in Table **1**.

Table 1. Difference between Supervised and Unsupervised Learning.

Supervised Learning	Unsupervised Learning
Input data is labeled	Input data is unlabeled
Feedback mechanism present	A feedback mechanism is absent
Data is classified	Properties are assigned to data
Suitable for prediction tasks	Suitable for analysis tasks
A known number of classes	An unknown number of classes
Consists of explanatory and response variables	Consists only of explanatory variables

Supervised learning is applicable in two major domains, namely, classification and regression. Classification is the process of categorizing data into different categories based on the labeled data used for training. Regression is similar to classification except that it can also be applied to continuous data, unlike classification, which can only be applied to discrete values. Unsupervised learning is suitable for clustering and association tasks. Clustering is used to discover the groups in the data, while the association is used for extracting the rules from a large amount of data.

IMPORTANT PYTHON LIBRARIES

Now we move to some important Python libraries that are used in Machine Learning.

• NUMPY

It's an array-processing package suitable for processing large multi-dimensional arrays and matrices.

Example:

```
import numpy as np
arr = np.array([[4, 13, 17, 6], [2,4,3,8]])
print(arr)
```

Output:

```
[[ 4 13 17  6]

 [ 2  4  3  8]]
```

CHAPTER 12

Programming with Python

Abstract: After the successful comprehension of the different aspects of Python and its applications in ML, it's now the time to look at how they can be combined and put to work using common examples. This will surely increase the reader's comprehension of the intricacies of Python and demonstrate the efficiency of the language in making the programming simple.

Keywords: Binary search, Factorial, Time series.

INTRODUCTION

After going through the previous chapters, we are sure you have a better understanding of the basics of Python. Now let's gear up to some programming exercises with Python. In the beginning, we provide some simple and basic Python programs using functions, lists, dictionaries, arrays, *etc*. Then we discuss some basic machine learning applications with Python programming.

BASIC PYTHON PROGRAM

Program to Solve a Quadratic Equation

Example:

```
# Import complex math module
import cmath
a = input('Enter your choice: ')
b =input('Enter your choice: ')
c = input('Enter your choice: ')

# Calculate the discriminant
d = (b**2) - (4*a*c)

# Find two solutions
sol1 = (-b-cmath.sqrt(d))/(2*a)
sol2 = (-b+cmath.sqrt(d))/(2*a)
print('The solution are {0} and {1}'.format(sol1,sol2))
```

Output:

```
Enter a: 1

Enter b: 2

Enter c: -15

The solution are (-5+0j) and (3+0j)
```

Program to Swap Two Numbers

Example:

```python
P = int( input("Please enter value for P: "))
Q = int( input("Please enter value for Q: "))
# To swap the value of two variables
# We will user third variable which is a temporary variable
temp_1 = P
P = Q
Q = temp_1
print ("The Value of P after swapping: ", P)
print ("The Value of Q after swapping: ", Q)
```

Output:

```
Please enter value for P: 21

Please enter value for Q: 81

The Value of P after swapping:   81

The Value of Q after swapping:   21
```

Program to Find the Factorial of Two Numbers

Example:

```python
num = int(input("Enter a number: "))
factorial = 1
if num < 0:
   print(" 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)
```

Output:

```
Enter a number: 9

The factorial of 9 is 362880
```

Program for Fibonacci Series Using Recursion

Example:

```
def recur_fibo(n):
    if n <= 1:
        return n
    else:
        return(recur_fibo(n-1) + recur_fibo(n-2))

nterms = int(input("How many terms? "))
# Check if the number of terms is valid
if nterms <= 0:
    print("Plese enter a positive integer")
else:
   print("Fibonacci sequence:")
   for i in range(nterms):
       print(recur_fibo(i))
```

Output:

```
How many terms? 5

Fibonacci sequence:

0

1

1

2

3
```

BIBLIOGRAPHY

[1] Available at: www.python.org

[2] Available at: https://docs.anaconda.com

[3] M. Lutz, *Learning Python: Powerful Object-Oriented Programming.* 5th O'Reilly Media, Incorporated, 2009.

[4] D.M. Beazley, *Python Essential Reference.* Addison-Wesley, 2009.

[5] C.P. Milliken, *Python Projects for Beginners: A ten-week bootcamp approach to python programming.* (1st ed.) Apress, 2019.

[6] A. Harris, *Python for Beginners: Learn Computer Programming with Python Now and How to Use It with This Step by Step Guide That Gives You the Basics of Python Coding + Practical Exercises*. Independently Published, 2019.

[7] F. Romano, and H. Kruger, *Learn Python Programming: An in-depth introduction to the fundamentals of Python 3rd Edition.* (3rd ed.). Packt Publishing, 2021.

[8] A. Müller, and S. Guido, *Introduction to Machine Learning with Python: A Guide for Data Scientists.* (1st ed.). O'Reilly Media, 2016.

[9] Availabel at: https://www.nltk.org/ For a complete reference to download and usage to NLTK.

[10] D. Paper, *Hands-on scikit-learn for machine learning applications. Data Science Fundamentals with Python.* Apress: Berkeley, CA, 2020.[http://dx.doi.org/10.1007/978-1-4842-5373-1]

SUBJECT INDEX

In a landscape where programming resources can be overwhelming, Basics of Python Programming: A Quick Guide for Beginners is an accessible, user-friendly, and effective learning companion. It's an indispensable resource for anyone stepping into the vast realm of coding. It is highly recommended for beginners eager to begin their Python programming journey.

Dharmendra Singh Rajput
Vellore Institute of Technology
India

Krishna Kumar Mohbey

Dr. Krishna Kumar Mohbey is an assistant professor of Computer Science at the Central University of Rajasthan, India. He received his Bachelor's in computer application from the MCRPV Bhopal in 2006, Master's in computer application from the Rajiv Gandhi Technological University Bhopal in 2009, and Ph.D. from the Department of Mathematics and Computer Applications from the National Institute of Technology Bhopal, India, in 2015. His areas of interest are data mining, mobile web services, big data analysis, and user behaviour analysis. He has published more than 50 research articles in various journals and conferences of international repute.

Malika Acharya

Ms. Malika Acharya is a research scholar at the Central University of Rajasthan, India. She holds a bachelor's degree in Technology (B.Tech) from Amity University Noida (2019) and a master's degree in Technology (M.Tech) from Rajasthan Technical University (2021). She is currently a research scholar at Central University of Rajasthan, Department of Computer Science. Her research areas include big data analysis, data mining, machine learning, social networking, and recommendation systems. She has published research articles in various journals and conferences of international repute.