1...

Introduction to Python

Learning Objectives...

- To learn Basic Concepts in Python.
- To study Basics of Python.
- To understands Data Types, Variables, Constants etc. in Python.

1.1 INTRODUCTION

- Python is a high-level, interpreted, interactive and object-oriented programming language. Today, Python is the trendiest programming language programming.
- There are several reasons for why Python programming language is the preferable choice of the programmers/developers over other popular programming languages like C++, Java and so on.
- Python is popular programming language because of it provides more reliability of code, clean syntax of code, advanced language features, scalability of code, portability of code, support object oriented programming, broad standard library, easy to learn and read, support GUI mode, interactive, versatile and interpreted, interfaces to all major commercial databases and so on.
- There are two major Python versions namely, Python 2 and Python 3. Python 3.0 was developed with the same philosophy as in prior versions.
- Python 3.9.5 is the newest major release of the Python programming language. Python scripts normally have the file extension .py.
- Some common **applications of Python Programming** are listed below:
  1. Google's App Engine web development framework uses Python as an application language.
  2. Maya, a powerful integrated 3D modeling and animation system, provides a Python scripting API.
  3. Linux Weekly News, published by using a web application written in Python programming.
  4. Google makes extensive use of Python in its Web Search Systems.

5. The popular YouTube video sharing service is largely written in Python programming.
6. The NSA (National Security Agency) uses Python programming for cryptography and intelligence analysis.
7. iRobot uses Python programming to develop commercial and military robotic devices.
8. The Raspberry Pi single-board computer promotes Python programming as its educational language.
9. Netflix and Yelp have both documented the role of Python in their software infrastructures.
10. Industrial Light and Magic, Pixar and others uses Python programming in the production of animated movies.

1.1.1 History

- Python laid its foundation in the late 1980s. Python was developed by Guido Van Rossum at National Research Institute for Mathematics and Computer Science in Netherlands in 1990.
- Inspired by Monty Python's Flying Circus, a BBC comedy series, he named the language Python.
- Python is derived from many other languages, including ABC, Modula-3, C, C++, Algol-68, SmallTalk, and Unix shell and other scripting languages.
- ABC programming language is said to be the predecessor of Python language which was capable of Exception Handling and interfacing with Amoeba Operating System.
- Like Perl, Python source code is now available under the GNU General Public License (GPL). In February 1991, Guido Van Rossum published Python 0.9.0 (first release). In addition to exception handling, Python included classes, lists and strings.
- In 1994, Python 1.0 was released with new features like lambda, map, filter, and reduce which aligned it heavily in relation to functional programming.
- Python 2.0 added new features like list comprehensions, garbage collection system and it supported Unicode.
- On December 3, 2008, Python 3.0 (also called "Py3K") was released. It was designed to rectify fundamental flaw of the language. In Python 3.0 the print statement has been replaced with a print() function.
- Python widely used in both industry and academia because of its simple, concise and extensive support of libraries.
- Python is available for almost all operating systems such as Windows, Mac, Linux/ Unix etc. Python can be downloading from http://www.python.org/downloads.

1.1.2 Features of Python

- Features of Python programming language are given below:

1. **Simple and Easy-to-Learn:** Python is a simple language with few keywords, simple structure and its syntax is also clearly defined. This makes Python a

beginner's language. In Python programming programs are easy to write and execute as it omits some cumbersome, poorly understandable and confusing features of other programming language such as C++ and Java.

2. **Interpreted and Interactive:** Python is processed at runtime by the interpreter. We need not compile the Program before executing it. The Python prompt interact with the interpreter to interpret the programs that we have written. Python has an option namely interactive mode which allows interactive testing and debugging of code.
3. **Object-Oriented:** Python supports Object Oriented Programming (OOP) concepts that encapsulate code within objects. All concepts in OOPs like data hiding, Operator Overloading, inheritance etc. can be well written in Python. It supports functional as well as structured Programming.
4. **Platform Independence/Cross Platform Language/Portable:** Python can run on a wide variety of hardware and software platforms and has the same interface on all platforms, such as Windows, Unix, Linux and Macintosh. Fig. 1.1 shows execution of Python code by interpreter.

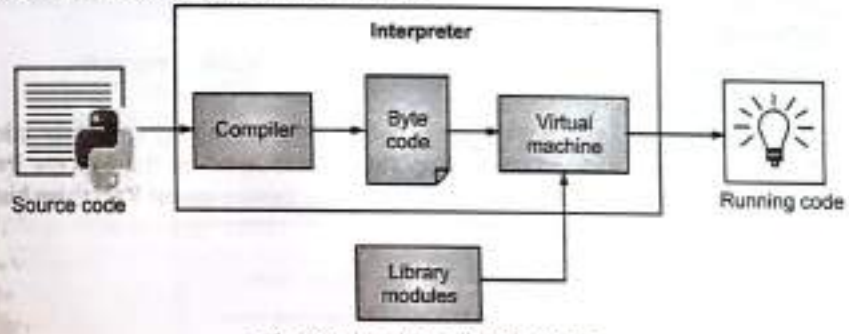


Fig. 1.1: Execution of Python Code

Python source code goes through Compiler which compiles the source code into a format known as byte code. Byte code is a lower level, platform independent, efficient and intermediate representation of the source code. As soon as source code gets converted to byte code, it is fed into PVM (Python Virtual Machine). The PVM is the runtime engine of Python; it's always present as part of the Python system, and is the component that truly runs the scripts. Technically, it's just the last step of what is called the Python interpreter.

5. **Scalable:** Python provides a better structure and support for large Programs than shell scripting. It can be used as a scripting language or can be compiled to byte code (intermediate code that is platform independent) for building large applications.
6. **Extendable:** You can add low-level modules to the Python interpreter. These modules enable Programmers to add to or customize their tools to be more efficient. It can be easily integrated with C, COM, ActiveX, CORBA, and Java.
7. **Dynamic:** Python provides very high-level dynamic data types and supports dynamic type checking. It also supports automatic garbage collection.

8. **GUI Programming and Databases:** Python supports GUI applications that can be created and ported to many libraries and windows systems, such as Windows Microsoft Foundation Classes (MFC), Macintosh, and the X Window system of Unix Python also provides interlaces to all major commercial databases.
9. **Broad Standard Library:** Python's library is portable and cross platform compatible on UNIX, Linux, Windows and Macintosh. This helps in the support and development of a wide range of applications from simple text processing to browsers and complex games.
10. **Free and Open Source:** Python programming language is developed under an OSI approved open source license making it freely available at official web address. The source code is also available for use. The Python software can be freely distributed and any one can use and read its source code make changes/modifications to it and use the pieces in new free programs.

1.1.3 Setting-up Path

- Before starting working with Python, a specific path is to set.
- Your Python program and executable code can reside in any directory of your system, therefore Operating System provides a specific search path that index the directories Operating System should search for executable code.
- The Path is set in the Environment Variable of My Computer properties.
- Steps are as follows:

1. First install the python. Then go to this pc (My Computer) and Right click on My Computer and click on properties. This should open up the **System Properties** window. Go to the **Advanced** tab and click the **Environment Variables** button:

Fig. 1.2

2. Click on new tab of user variables.

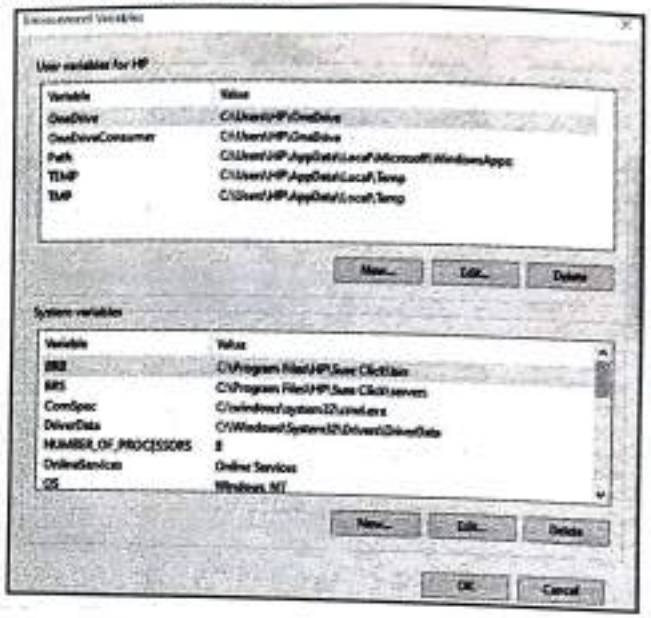


Fig. 1.3

3. Write path in variable name.

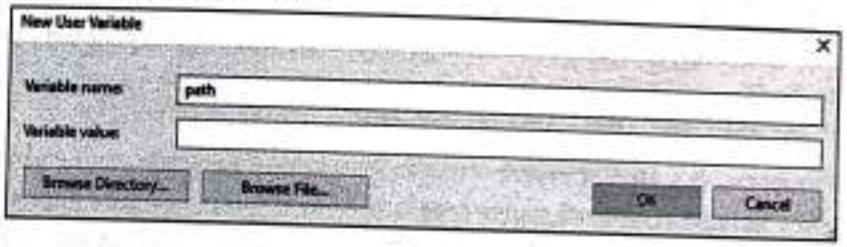


Fig. 1.4

4. Copy the path of Python folder and Paste path of Python in variable value.

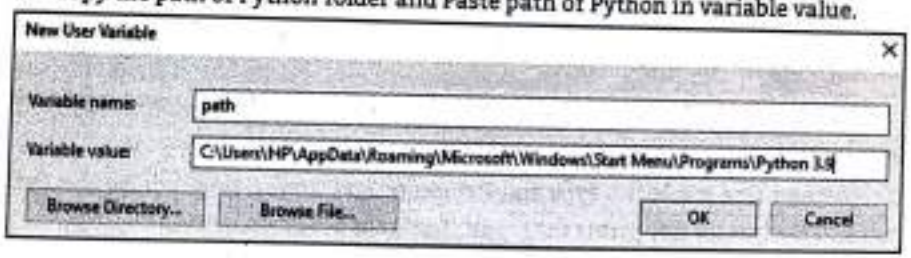


Fig. 1.5

5. Click on Ok button:

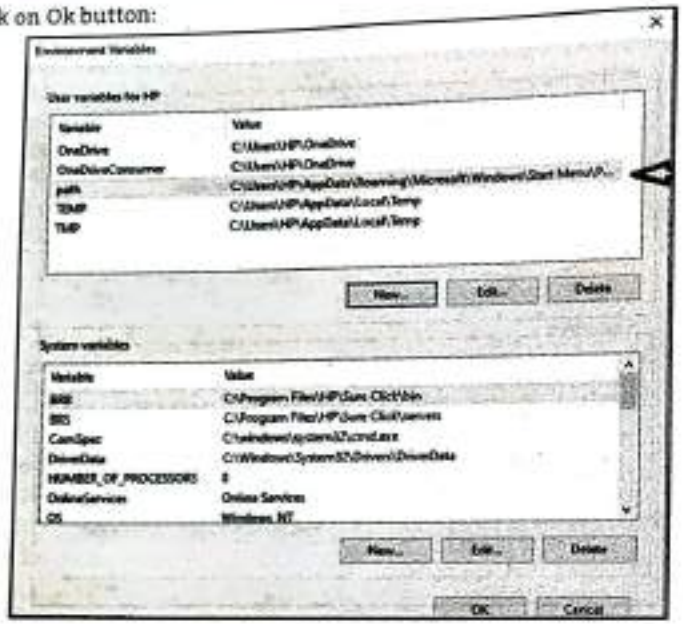


Fig. 1.6

6. Click on Ok button:

Close all windows. Now you can run **python.exe** without specifying the full path to the file:

1.1.4 Working with Python Interpreter

- An interpreter is a kind of program that executes other programs. When you write Python programs, it converts source code written by the developer into intermediate language which is again translated into the native language / machine language that is executed.

Starting Python in different Modes:

1. **Starting Python Command Line:**

- A Python script can be executed at command line also. This can be done by invoking the interpreter on the application.
- In command line mode, we type the Python programming program on the Python shell and the interpreter prints the result. The steps are given below:

Step 1 : Press Start button, (See Fig. 1.7).

Fig. 1.7

Step 2 : Click on All Programs and then click on Python 3.9 (64 bit) as shown in Fig. 1.7. We will see the Python interactive prompt in Python command line.

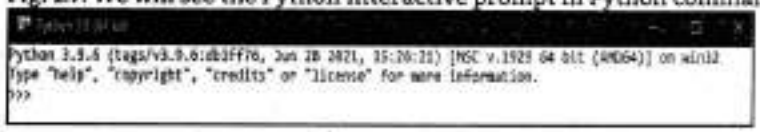


Fig. 1.8

Python command prompt contains an opening message >>> called command prompt. The cursor at command prompt waits for to enter Python command. A complete command is called a statement. For example check first command to print message, in Fig. 1.9.

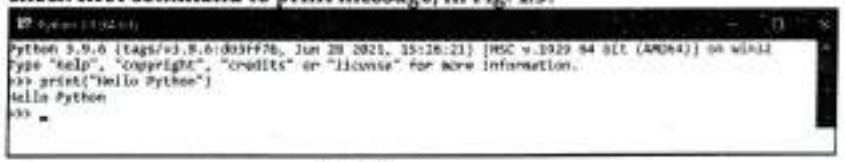


Fig. 1.9

Step 3 : To exit from the command line of Python, use Ctrl+z or quit() followed by Enter.

2. **Starting Python IDLE:**

- When we install Python 3, we also get IDLE (Integrated Development Environment). IDLE includes a color syntax-highlighting editor, a debugger, the Python Shell, and a complete copy of Python 3's online documentation set.
- The steps are given below:

Step 1 : Press Start button and click on IDLE (Python 3.9, 64-bit) opti as shown in Fig. 1.10

Step 2 : We will see the Python interactive prompt i.e. interactive shell.

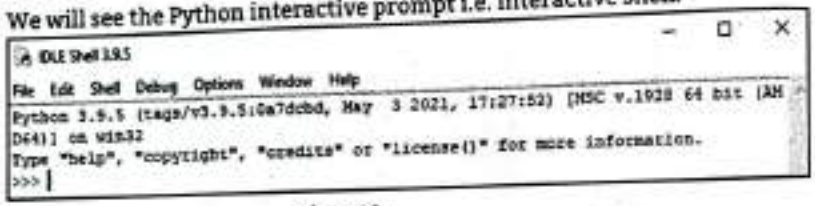


Fig. 1.10

Python interactive shell prompt contains opening message >>>, called shell prompt. A cursor is waiting for the command. A complete command is called a statement. When we write a command and press enter, the python interpreter will immediately display the result.

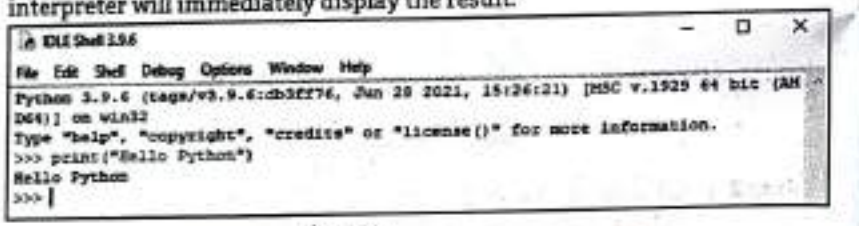


Fig. 1.11

Executing Python Programs Scripts:

- In Python IDLEs shell window, click on File, and select the New File or press Ctrl+N.

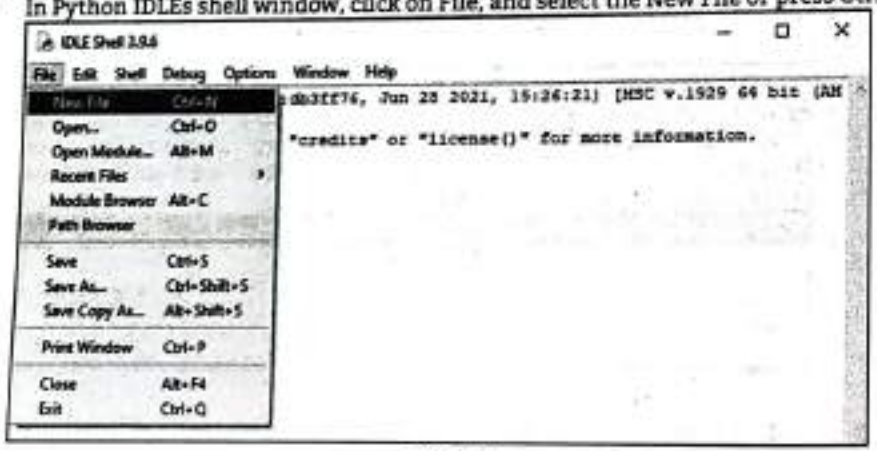


Fig. 1.12

- As soon as we click on New File, the window appears as shown in Fig. 1.13.

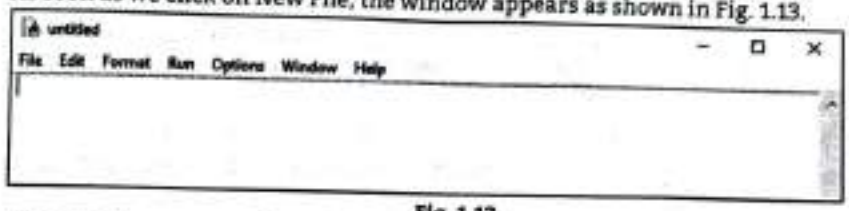


Fig. 1.13

- Write Python program in script mode, (See Fig. 1.14).

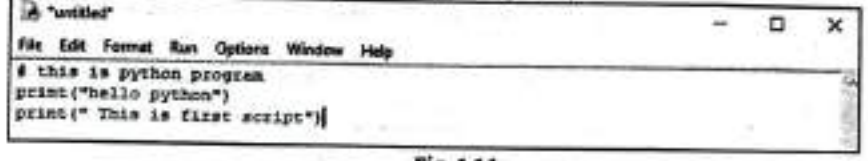


Fig. 1.14

- Save the above code with filename. By default, python interpreter will save it using the filename.py. Here, we save the script with file name test.py.

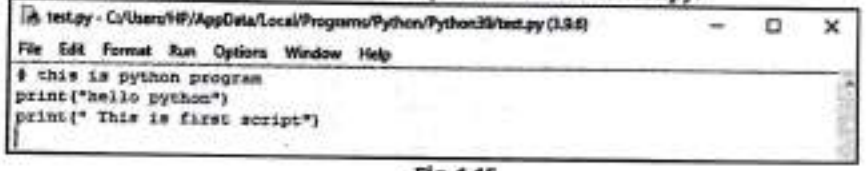


Fig. 1.15

- To run the Python program, click on Run and then Run Module option or we can press Ctrl+F5.

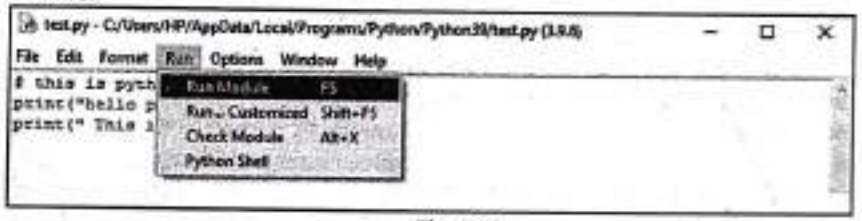


Fig. 1.16

- After clicking Run Module, we will get the output of program on Python shell.

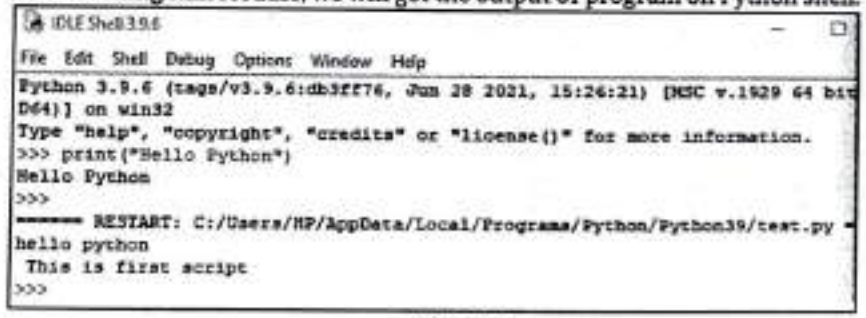


Fig. 1.17

1.1.5 Basic Syntax

- Python program is set of instructions. Python programs are typically organized with one statement per line. Before to write and run a Python program we understand structure of a Python program.
- Fig. 1.18 shows a typical program structure of Python programming.
- Python programming programs are structured as a sequence of statements. A Python statement is smallest program unit.
- Statements are the instructions that are written in a program to perform a specific task. A Python statement is a complete instruction executed by the Python interpreter.
- By default, the Python interpreter executes all statements sequentially, but we can change order of execution using control statements.

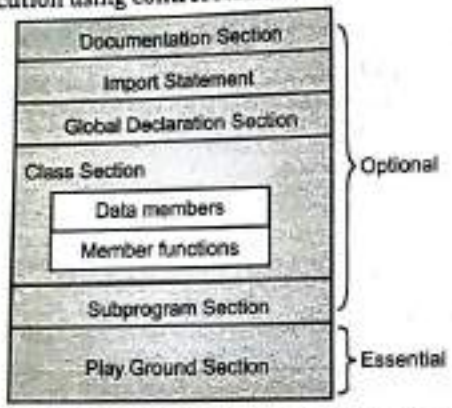


Fig. 1.18: Typical Program Structure of Python Programming

- Program structure of Python programming contains following sections:
  1. **Documentation Section** includes the comments that specify the purpose of the program. A comments that is a non-executable statement which is ignored by the compiler while program execution. Python comments are written anywhere in the program.
  2. **Import Section** is used includes different built in or user defined modules.
  3. **Global Declaration Section** is used to define the global variables for the programs.
  4. **Class Section** describes the information about the user defined classes in the Python program. A class is a collection of data members and member functions called method that operate on data members.
  5. **Sub Program Section** includes use defined functions. The functions include the set of statements that need to be executed when the function is called form anywhere.
  6. **Play Ground Section** is the main section of Python program and the main section starts where the function calling.

- Python has two basic modes namely, script and interactive.

1. **Interactive Mode:**

- The interactive mode is a command line shell which gives immediate feedback for each statement, while running previously fed statements in active memory.
- Interactive mode is used for quickly and conveniently running single line or blocks of code. Here's an example using the python shell that comes with a basic python installation.
- The ">>>" indicates that the shell is ready to accept interactive commands. For example, if we want to print the statement "Interactive Mode", simply type the appropriate code and hit enter.

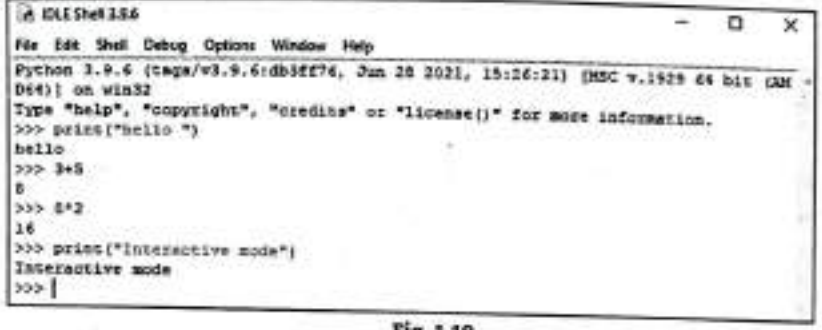


Fig. 1.19

2. **Script Mode:**

- The normal script mode is the mode where the scripted and finished .py files are run in the Python interpreter.
- In the standard Python shell we can go to "File" → "New File" (or just hit Ctrl + N) to pull up a blank script to write the code. Then save the script with a ".py" extension.
- We can save it anywhere we want for now, though we may want to make a folder somewhere to store the code as we test Python out. To run the script, either select "Run" → "Run Module" or press F5.
- We should see something like the following. (See Fig. 1.20 (a) and 1.20 (b)).

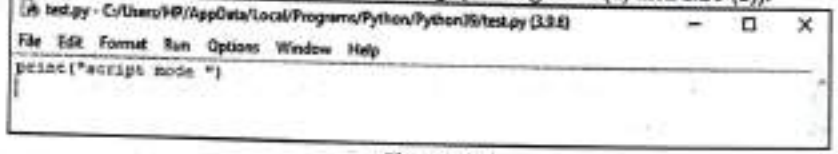


Fig. 1.20 (a)

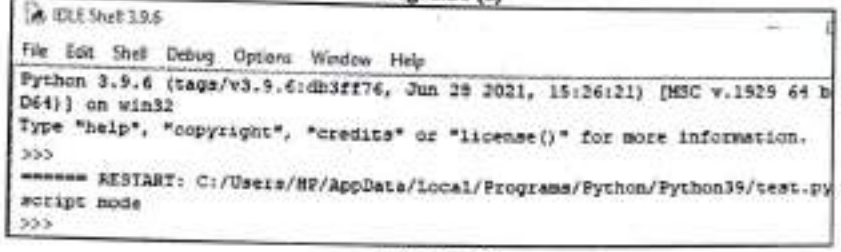


Fig. 1.20 (b)

- Following simple Python program shows structure of Python program:

Program 1.1: Calculate Area and Circumference of circle using class.

```
# Documentation section
import math      # import statement
radius=5         # global declaration Section
class Circle(): # class section
    def getArea(self):
        return math.pi*radius*radius
    def getCircumference(self):
        return radius*2*math.pi
    def showradius():   # sub Program section
        print("Radius =",radius)
showradius()        # Playground Section
c=Circle()
print("Area =",c.getArea()) print("Circumference=",c.getCircumference())
```

Output:

```
Radius = 5
Area = 78.53981633974483
Circumference = 31.41592653589793
```

1.2 BASICS OF PYTHON

- In this section we will study various basic concepts if Python programming such data types, variables, identifiers, control flow statements and so on.

1.2.1 Data Types

- The type of data value that can be stored in an identifier/variable is known as its data type.
- The data type determines how much memory is allocated to store data and what operations can be performed on it.
- The data stored in memory can be of many types and are used to define the operations possible on them and the storage method for each of them.
- Python handles several data types to facilitate the needs of programmers and application developers for workable data.
- The data stored in memory can be of many types. For example, a person's age is stored as a numeric value and his or her address is stored as alphanumeric characters.
- Python has various standard data types that are used to define the operations possible on them and the storage method for each of them.

- Data types in Python programming includes:

1. **Numbers:** Represents numeric data to perform mathematical operations. Numbers can be integers (like 1 and 2), floats (like 1.1 and 1.2), fractions (like 1/2 and 2/3), or even complex numbers.
2. **String:** Represents text characters, special symbols or alphanumeric data. String is sequence of Unicode characters.
3. **List:** Represents sequential data that the programmer wishes to sort, merge etc.
4. **Tuple:** Represents sequential data with a little difference from list.
5. **Dictionary:** Represents a collection of data that associate a unique key with each value.
6. **Boolean:** Represents truth values (true or false).
7. **None:** Python defines a special variable None denoting a "null object".

To determine a variable's type in Python programming we can use the type() function. The value of some objects can be changed. Objects whose value can be changed are called mutable and objects whose value is unchangeable (once they are created) are called immutable.

Let us see above data types in detail:

Boolean (bool Data Type):

The simplest build-in type in Python is the bool data type, it represents the two values, True and False. Internally the true value is represented as 1 and false is 0.

Example:

```
>>> size = 1
>>> size < 0
False
>>> size = 0
>>> size < 0
False
>>> size = -1
>>> size < 0
True
```

None Data Type:

Python defines a special variable None denoting a "null object", which is convenient to use when a variable is available but its value is considered undefined.

None in Python is used for defining null variables and objects. None is an instance of the NoneType class.

- None object is accessed through the built-in name None. It is a data type of the class NoneType object.

```
answer = None
    <may update answer from other data...>
if answer is None:
quit = True
elif answer == 'quit':
quit = True
else:
quite = False
```

- To check if a variable answer is None or not, always use if answer is None or if answer is not None. Testing just if not answer is dangerous, because the test is true if answer is an empty string. The difference between the is and == operators: is tests for object identity, while == tests if two objects have the same value (i.e., the same content).

3. Number Data Types:

- Number data types store numeric values. Integers, floating point numbers and complex numbers are falls under Python numbers category. They are defined as int, float and complex in Python.

Fig. 1.21: Types of Numbers Data Type

(i) Integers (int Data Type):

- An int data type represents an integer number. An integer number is a number without any decimal or fractional point.
- For example, a = 57, here a is called the int type variable and stores integer value 57.
- These represent numbers in the range – 2147483648 to 2147483647.
- An integer is a whole number that can be positive (+) or negative (–). Integer can be of any length, it is only limited by the memory available.

  Example: For number data types are integers.

```
>>> a=10
>>> a
10
```

- To determine the type of a variable type() function is used.

```
>>> type(a)
<class 'int'>
```

- In Python programming one can write integers in Hexadecimal (base 16), Octal (base 8) and Binary (base 2) formats by using one of the following prefixes to the integer.

Sr. No.	Prefix	Interpretation	Base
1.	'0b' or '0B'	Binary	2
2.	'0o' or '0O'	Octal	8
3.	'0x' or '0X'	Hexadecimal	16

Example: Integers in binary, octal and hexadecimal formats.

```
>>> print(0b10111011) # binary number
187
>>> print(0o10)       # octal number
8
>>> print(0xFF)       # hexadecimal number
255
```

(ii) Floating Point Numbers (Float Data Type):

- The float data type represents the floating point number. The floating point number is a number that contains a decimal point.
- Examples of floating point numbers, 0.5, –3.445, 330.44. For example, num = 2.345.
- Floating-point number or Float is a positive or negative number with a fractional part.
- A floating point number is accurate up to 15 decimal places. Integer and floating points are separated by decimal points. For example, 1 is integer, 1.0 is floating point number.
- One can append the character e or E followed by a positive or negative integer to specify scientific notation.

  Example: Floating point number.

```
>>> x = 10.1
>>> x
10.1
y = -10.5
>>> y
-10.5
>>> print(72e3)
72000.0
print(7.2e-3)
0.0072
```

(iii) Complex Numbers (Complex Data Type):

- A complex number is a number that is written in the form of a+bj. Here, a represents the real part of the number and b represents the imaginary part of the number.
- The suffix J or j after b represents the square root value of –1. The part a and b may contain the integers or floats. For example, 3+5j, 0.2+10.5j are complex numbers.
- Complex numbers are written in the form, x + yj, where x is the real part and y is the imaginary part.

Example: Complex number.

```
>>> x = 3+4j
>>> print(x.real)
3.0
>>> print(x.imag)
4.0
```

4. **String Data Type:**

- String is a collection of group of characters. Strings are identified as a contiguous set of characters enclosed in single quotes (' ') or double quotes (" ").
- Any letter, a number or a symbol could be a part of the string. Strings are unchangeable (immutable). Once a string is created, it cannot be modified.
- Strings in python support Unicode characters. The default encoding for Python source code is UTF-8. So, we can also say that String is a sequence of Unicode characters.
- Strings are ordered. Strings preserved the order of characters inserted.

Example: For string data type.

```
>>> s1="Hello"                  # string in double quotes
>>> s2='Hi'                     # string in single quotes
>>> s3="Don't open the door"    # single quote string in double quotes
>>> s4='I said "yipee"'         # double quote string in single quotes
>>> s1
'Hello'
>>> s2
'Hi'
>>> s3
"Don't open the door"
>>> s4
'I said "yipee"'
>>>
```

5. **List Data Type:**

- Lists are the most versatile of Python's compound data types. List is an ordered sequence of items. It is one of the most used data type in Python and is very flexible.
- List can contain heterogeneous values such as integers, floats, strings, tuples, lists and dictionaries but they are commonly used to store collections of homogeneous objects.
- The list data type in Python programming is just like an array that can store a group of elements and we can refer to these elements using a single name.
- Declaring a list is pretty straight forward. Items separated by co... ...sed within square brackets [ ].

Example: For list.

```
>>> first=[10, 20, 30]                  # homogenous values in list
>>> second=["One","Two","Three"]        # homogenous values in list
>>> first
[10, 20, 30]
>>> second
['One', 'Two', 'Three']
>>> third=[10,"one",20,"two"]           # heterogeneous values in list
>>> third
[10, 'one', 20, 'two']
>>> first + second                      # prints the concatenated lists
[10, 20, 30, 'One', 'Two', 'Three']
```

- Lists are mutable which means that value of elements of a list can be altered by using index.

Example: For list with updation/alteration/modification.

```
>>> first=[10, 20, 30]
>>> first[2]                            # print second value in the list
30
>>> first[2]=50                         # change second value in the list
>>> first
[10, 20, 50]
>>> first[2]                            # print second value in the list
50
>>> print (first*2)                     # prints the list two times
[10, 20, 30, 10, 20, 30]
```

List and Strings:

- A string is a sequence of characters and list is a sequence of values, but a list of characters is not same as string. We can convert string to a list of characters.

Example: For conversion of string to a list.

```
>>> p="Python"
>>> p
'Python'
>>> l=list(p)
>>> l
['P', 'y', 't', 'h', 'o', 'n']
```

6. **Tuple Data Type:**

- Tuple is an ordered sequence of items same as list. The only difference is that tuples are immutable. Tuples once created cannot be modified.
- Tuples are used to write-protect data and are usually faster than list as it cannot change dynamically. It is defined within parentheses ( ) where items are separated by commas (,).
- A tuple data type in python programming is similar to a list data type, which also contains heterogeneous items/elements.

Example:

```
tuple_obj=(70,2.50, "tybca")
```

- Tuples are immutable objects whose contents cannot be altered:

```
words = ('flower', 'homophone', 'flour')
words[1] = 'and' # illegal - Python issues an error message
```

7. **Dictionary Data Type:**

- Dictionary data type in Python is an unordered collection of key-value pairs. Dictionaries in Python work like associative arrays or hashes found in Perl and consist of key-value pairs.
- When we have the large amount of data, the dictionary data type is used. The dictionary data type is mutable in nature which means we can update modify/update any value in the dictionary.
- Items in dictionaries are enclosed in curly braces { } and separated by the comma (,). A colon (:) is used to separate key from value. Values can be assigned and accessed using square braces ([]).

Example: For dictionary data type.

```
>>> dic1={1:"First","Second":2}
>>> dic1
{1: 'First', 'Second': 2}
>>> type(dic1)
<class 'dict'>
>>> dic1[3]="Third"
>>> dic1
{1: 'First', 'Second': 2, 3: 'Third'}
>>> dic1.keys()
dict_keys([1, 'Second', 3])
>>> dic1.values()
dict_values(['First', 2, 'Third'])
>>>
```


1.2.2 Variables

- A variable is like a container that stores values that we can access or change. It is a way of pointing to a memory location used by a program.
- We can use variables to instruct the computer to save or retrieve data to and from this memory location.
- A variable is a name given to a location in the computer's memory location, where the value can be stored that can be used in the program.
- When we create a variable, some space in the memory is reserved or allocated for that variable to store a data value in it.
- The size of the memory reserved by the variable depends on the type of data it is going to hold. The period of time that a variable exists is called its lifetime.
- The variable is so called because its value may vary during the time of execution, but at a given instance only one value can be stored in it.

Variable Declaration:

- A variable is an identifier that holds a value. In programming, we say that we assign a value to a variable. Technically speaking, a variable is a reference to a computer memory, where the value is stored.
- Basic rules to declare variables in python programming language:
  1. Variables in Python can be created from alphanumeric characters and underscore(_) character.
  2. A variable cannot begin with a number.
  3. The variables are case sensitive. Means Amar is differ the 'AMAR' are two separate variables.
  4. Variable names should not be reserved word or keyword.
  5. No special characters are used except underscore (_) in variable declaration.
  6. Variables can be of unlimited length.
- Python variables do not have to be explicitly declared to reserve memory space. The variable is declared automatically when the variable is initialized, i.e., when we assign a value to the variable first time it is declared with the data type of the value assigned to it.
- This means we do not need to declare the variables. This is handled automatically according to the type of value assigned to the variable. The equal sign (=) i.e., the assignment operator is used to assign values to variables.
- The operand to the left of the = operator is the name of the variable and the operand to the right of the = operator is the literal value or any other variable value that is stored in the variable.

Syntax: `variable=value`

Example: For variable.

```
>>> a=10
>>> a
10
>>>
```

- Python language allows assigning a single value to several variables simultaneously.

  Example: `a=b=c=1`

  All above three variables are assigned to same memory location, when integer object is created with value 1.
- Multiple objects can also have assigned to multiple variables:

  Example: `a, b, c = 10,5.4,"hello"`

  In above example Integer object a assigned with value 10, float object b assigned with value 5.4 and string object c assigned with value "hello".

  Example: If x, y, z are defined as three variable in a program, then x = 10 will store the value 10 in the memory location named as x, y = 5 will store the value 5 in the memory location named as y and x + y will store the value 15 in the memory location named as z (as a result after computation of x + y).

```
>>> x = 10
>>> y = 5
>>> name = "Python"
>>> z = x + y
>>> print(x); print(y); print(name); Print (z)
10
5
Python
15
```

1.2.3 Input/Output Functions

- A python program needs to interact with the user to accomplish the desired task or result this can be achieved using Input-Output functions.
- The input() function helps to enter data at run time by the user and the output function print() is used to display the result of the program on the screen after execution.
- Input means the data entered by the user/programmer in the form of program. In python, the input() function is used to accept an input from a user.

Syntax:
```
variable_name=input()              # without any argument
variable_name=input('String')      # with argument
```

Example: For input in Python.

```
>>> input()
Hello python
'Hello python
>>> x= input ("Enter data:")
Enter data: 11.22
>>> print(x)
11.22
```


- Output means the data comes from computer after processing. In Python programming the print() function display the input value on screen.

  Syntax: `print(expression/constant/variable)`

  Example: For output in python.

```
>>> print ("Hello")
Hello
>>> a="Hello"
>>> b="Python"
>>> print(a+b)
```

  Output:

```
HelloPython
```

Formatting Output:

- Sometimes we would like to format our output to make it look attractive. This can be done by using the str.format() method. This method is visible to any string object.

```
x = 10; y = 20
print('The value of x is {} and y is {}'.format(x,y))
```

  Output: The value of x is 10 and y is 20

- Here the curly braces {} are used as placeholders. The output is printed according to the order given as follows:

```
print('I love {0} and {1}'.format('apple','milk'))
# Output: I love apple and milk
print('I love {1} and {0}'.format('apple','milk'))
# Output: I love milk and apple
```

1.2.4 Operators

- Python language supports the following types of operators.
  - Arithmetic Operators
  - Comparison (Relational) Operators
  - Assignment Operators
  - Logical Operators
  - Bitwise Operators
  - Membership Operators
  - Identity Operators
- Arithmetic, logical, Relational operators supported by Python language are same as other languages like C, C++.

1. Arithmetic Operators:

- Arithmetic operators are used to perform mathematical operations like addition, subtraction, multiplication etc.

Operator	Meaning	Example
+	Add two operands or unary plus	x + y + 2
-	Subtract right operand from the left or unary minus	x - y - 2
*	Multiply two operands	x * y
/	Divide left operand by the right one (always results into float)	x / y
%	Modulus - remainder of the division of left operand by the right	x % y (remainder of x/y)

- The new arithmetic operators in python are:
  (a) **** (Exponent)** - Performs exponential (power) calculation on operators
  Example: a**b = 10 to the power 20.
  (b) **// (Floor Division)** - The division of operands where the result is the quotient in which the digits after the decimal point are removed. But if one of the operands is negative, the result is floored, i.e., rounded away from zero (towards negative infinity)
  Example: 9//2 = 4 and 9.0//2.0 = 4.0, -11//3 = -4, -11.0//3 = -4.0

Program 1.2: For arithmetic operators we will take simple example of addition.

```
x = 15
y = 4
print('x + y =',x+y)
# Output: x + y = 19

print('x - y =',x-y)
# Output: x - y = 11

print('x * y =',x*y)
# Output: x * y = 60

print('x / y =',x/y)
# Output: x / y = 3.75

print('x // y =',x//y)
# Output: x // y = 3

print('x ** y =',x**y)
# Output: x ** y = 50625
```

2. Logical operators:

- The logical operators are used to perform logical operations like (and, or, not) and to combine two or more conditions to provide a specific result i.e. true or false.

Operator	Meaning	Example
and	True if both the operands are true	x and y
or	True if either of the operands is true	x or y
not	True if operand is false (complements the operand)	not x

Program 1.3: For Logical operators we will take simple example.

```
x = 4
print(x > 2 and x < 10)
print(x > 2 or x < 10)
print(not(x > 10))
```

Output:

```
True
True
True
```

3. Relational / Comparison operators:

- Comparison operators are used to compare values. It either returns True or False according to the condition.

Operator	Meaning	Example
==	Equal to - True if both operands are equal	x == y
!=	Not equal to - True if operands are not equal	x != y
<	Less than - True if left operand is less than the right	x < y
<=	Less than or equal to - True if left operand is less than or equal to the right	x <= y
>	Greater than - True if left operand is greater than the right	x > y
>=	Greater than or equal to - True if left operand is greater than or equal to the right	x >= y
<>	This is new Relational operator in Python. If values of two operands are not equal, then condition becomes true.	x <> y

Program 1.4: For Relational operators.

```
x = 4
y = 2
print(x == y)  # returns False because 4 is not equal to 2
print(x > y)   # returns True because 4 is greater that to 2
print(x >= y)  # returns True because 4 is greater that equal to 2
print(x < y)   # returns False because 4 is not less that to 2
print(x <= y)  # returns False because 4 is not less that equal to 2
print(x != y)  # returns True because 4 is not equal to 2
```

Output:

```
False
True
True
False
False
True
```

4. Assignment Operators:

- Assignment operators are used in Python to assign values to variables.
- a = 5 is a simple assignment operator that assigns the value 5 on the right to the variable a on the left.
- There are various compound operators in Python like a += 5 that adds to the variable and later assigns the same. It is equivalent to a = a + 5.
- The following are assignment operators in python.

Operator	Example	Same like
=	x=4	x=4
+=	x+=4	x=x+4
-=	x-=4	x=x-4
*=	x*=4	x=x*4
/=	x/=4	x=x/4
%=	x%=4	x=x%4
//=	x//=4	x=x//4
**=	x**=4	x=x**4
&=	x&=4	x=x&4
|=	x|=4	x=x|4
^=	x^=4	x=x^4

Program 1.5: For Assignment operators.

```
x = 4
x *= 2
print(x)

y = 10
y += 5
print(y)

z = 9
z /= 3
print(z)

a = 12
a //= 3
print(a)

b = 5
b **= 3
print(b)
```

Output:

```
8
15
3.0
4
125
```


5. Bitwise Operators:

- Bitwise operators act on operands. It operates bit by bit, hence the name is bitwise.
- For example, 2 is 10 in binary and 7 is 111.
- The following are bitwise operators in python which are same as in C, C++.

Operator	Meaning
&	bitwise AND: Sets each bit to 1 if both bits are 1
|	bitwise OR: Sets each bit to 1 if one of two bits is 1
^	bitwise XOR: Sets each bit to 1 if only one of two bits is 1
~	bitwise NOT: Inverts all the bits
<<	bitwise left shift: Shift left by pushing zeros in from the right and let the leftmost bits fall off
>>	bitwise right shift: Shift right by pushing copies of the leftmost bit in from the left, and let the rightmost bits fall off

Special Operators:

- Python language offers some special type of operators like the identity operator or the membership operator. They are described below with examples.

1. Identity operators

- **is** and **is not** are the identity operators in Python. They are used to check if two values (or variables) are located on the same part of the memory. Two variables that are equal does not imply that they are identical.

Operator	Meaning	Example
is	True if the operands are identical (refer to the same object)	x is True
is not	True if the operands are not identical (do not refer to the same object)	x is not True

Program 1.6: For Identity operators.

```
x1 = 5
y1 = 5
x2 = 'Hello'
y2 = 'Hello'
x3 = [1,2,3]
y3 = [1,2,3]
print(x1 is not y1)
# Output: False
print(x2 is y2)
# Output: True
print(x3 is y3)
# Output: False
```

- Here, we see that x1 and y1 are integers of same values, so they are equal as well as identical. Same is the case with x2 and y2 (strings).
- But x3 and y3 are list. They are equal but not identical. It is because interpreter locates them separately in memory although they are equal.

2. Membership operators

- **in** and **not in** are the membership operators; used to test whether a value or variable is in a sequence. (string, list, tuple, set and dictionary). In a dictionary we can only test for presence of key, not the value.

Operator	Meaning	Example
in	True if value/variable is found in the sequence	5 in x
not in	True if value/variable is not found in the sequence	5 not in x

Program 1.6: For Membership operators.

```
x = 'Hello world'
y = {1:'a',2:'b'}
print('H' in x)
# Output: True
print('hello' not in x)
# Output: True
print(1 in y)
# Output: True
print('a' in y)
# Output: False
```

- Here, 'H' is in x but 'hello' is not present in x (remember, Python is case sensitive). Similarly, 1 is key and 'a' is the value in dictionary y. Hence, 'a' in y returns False.

1.3 CONDITIONAL STATEMENTS-IF, IF-ELSE, NESTED IF-ELSE

- Python programming language provides following types of decision making statements.

(i) If statement:

- if statement is the most simple decision making statement. It is used to decide whether a certain statement or block of statements will be executed or not i.e. if a certain condition is true then a block of statement is executed otherwise not.

Syntax:

```
if expression:
statement(s)
```

- A few important things to note about if statements:
  1. The colon (:) is significant and required. It separates the header of the compound statement from the body.
  2. The line after the colon must be indented. It is standard in Python to use four spaces for indenting.

Program 1.7: Program for if statement.

```
a = 10
b = 20
if b > a:
 print("b is greater than a")
```

Output:

```
b is greater than a
```

(ii) If-ELSE Statements:

- Else is also called a two-way selection statement, because it leads the program to make a choice between two alternative courses of action.

Syntax:

```
if expression:
   statement(s)
else:
   statement(s)
```

Program 1.8: Program for else statement.

```
a = 30
b = 20
if b > a:
 print("b is greater than a")
else:
 print("a is greater than b")
```

Output:

```
a is greater than b
```

Program 1.9: Write a program to check whether the number N is even or odd.

```
N = 4
if N%2 == 0:
   print('Number is Even')
else:
   print(' Number is Odd')
```

Output:

```
Number is Even
```

(iii) IF...ELIF...ELSE Statements:

- When a program contains several testing conditions that involve more than two alternative course of action. In multi-way if statements program checks each condition until one evaluates to true or all evaluate to false. When a condition evaluates to True, the corresponding action of condition is took place. If no condition satisfy means evaluate to true then the corresponding action of trailing else is performed.

Syntax:

```
if expression:
   sequence of statements-1
elif condition-n:
   sequence of statements-n
else:
   default sequence of statements
```

Program 1.10: Program for IF...ELIF...ELSE Statements.

```
a = 4
b = 4
if b > a:
 print("b is greater than a")
elif a == b:
 print("a and b are equal")
else:
 print("a is greater than b")
```

Output:

```
a and b are equal
```

(iv) Nested IF-Else Statement:

- A nested if is an if statement that is the target of another if statement. Nested if statement means an if statement within another if statement.

Syntax:

```
if (<condition1>):
   statement(s)
   if (<condition2>):
      statement(s)
   else
else:
   if (<condition3>):
      statement(s)
   else
Statement(s)
```

Program 1.11: Program for Nested if-else Statements.

```
a = -10
if a > 0:
 print("Positive Number")
```



```
else:
 print("Negative Number")
 #nested if
if -10 <= a:
 print("Two digit Negative Number")
```

Output:

```
Negative Number
Two digit Negative Number
```

1.4 LOOPING-FOR,WHILE,NESTED LOOPS

- In a situation where you need to execute a block of code several number of times, Programming languages provide control statements with repetition statements known as loops, which repeat an action.

1. while loop:

- A while loop statement in Python programming language repeatedly executes a target statement as long as a given condition is true. In while loop, condition is checked first. The body of the loop is entered only if the condition evaluates to True. After one iteration, the condition is checked again. This process continues until the condition evaluates to False.

Syntax:

```
while condition:
   statement(s)
```

Program 1.12: Write a program to display numbers from 1 to 10.

```
i = 1
while i < 11:
   print(i)
   i += 1
```

Output:

```
1
2
3
4
5
6
7
8
9
10
```

2. for loop:

- The "for" loops in Python executes a certain block of code for a known number of iterations. The speciality of "for" loop in Python is that the block of code can be iterated for the number of items existing within a list, dictionary, string variable, over a particular range of numbers in counted number of steps, and for the number of items existing in a tuple. It has the ability to iterate over the items of any sequence, such as a list or a string.

  Syntax:

```
for iterating_var in sequence:
    statements(s)
```

- **Iterating by Sequence Index:** An alternative way of iterating through each item is by index offset into the sequence itself.
- **Syntax:**

```
for i in [Python Iterable]:
expression(i)
```

Example: Here "Python Iterable" can be a list, tuple or other advanced data structures.

```
for x in'Hi':
    print x
```

Program 1.13: Example for "for" statement.

```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
 print(x)
for letter in 'Python':
    print(letter)
    print("Good bye!")
```

Output:

```
apple
banana
cherry
P
Good bye!
y
Good bye!
t
Good bye!
h
Good bye!
o
Good bye!
n
Good bye!
```


Iterating by Subsequence Index:

- The function **range()** generates lists containing arithmetic progressions.
- We can generate a sequence of numbers using range() function. range(10) will generate numbers from 0 to 9 (10 numbers). We can also define the start, stop and step size as range(start,stop,step size). step size defaults to 1 if not provided.

  Syntax: `range(start-value, end-value, difference between the values)`
- This function does not store all the values in memory, it would be inefficient. So it remembers the start, stop, step size and generates the next number on the go.
- We can use the range() function in for loops to iterate through a sequence of numbers. It can be combined with the len() function to iterate though a sequence using indexing.

Program 1.14: Program to iterate through a list using indexing.

```
genre = ['Folkl', 'Classical', 'jazz']
# iterate over the list using index
for i in range(len(genre)):
    print("I like", genre[i])
```

Output:

```
I like Folk
I like Classical
I like jazz
```

- Python has another range function called **xrange()**.

  Syntax: `xrange(start-value, end-value, difference between the values).`
- The **xrange()** allowing the memory to be freed, when the range list is not in use because **xrange()** populates its range list whenever it is accessed by means of such as for loop.

3. Nested loops:

- When you have a block of code you want to run x number of times, then a block of code within that code which you want to run y number of times, you use what is known as a "nested loop". In Python, these are heavily used whenever someone has a list of lists - an iterable object within an iterable object.

Program 1.15: Program for Nested loops.

```
num_list = [1, 2, 3]
alpha_list = ['a', 'b', 'c']

for number in num_list:
    print(number)
    for letter in alpha_list:
    print(letter)
```

Output:

```
1
a
b
c
2
a
b
c
3
a
b
c
```

1.5 CONTROL STATEMENTS-BREAK, CONTINUE, PASS

- Python supports the following Loop Control Statements:

1. break Statement:

- Break statement is a jump statement which is used to transfer execution control. It breaks the current execution and in case of inner loop, inner loop terminates immediately and resumes execution at the next statement. The break statement can be used in both *while* and *for* loops.

Syntax:

```
break
```

Program 1.16: Example for break statement.

```
for letter in 'sybca':
 if letter == 'b':
    break
 print('Current Letter:', letter)
```

Output:

```
Current Letter: s
Current Letter: y
```

Program 1.17: Example for break statement.

```
var = 10
while var > 0:
 print('Current variable value:', var)
 var = var -1
 if var == 7:
 break
print('Out loop')
```


Output:

```
Current variable value: 10
Current variable value: 9
Current variable value: 8
Out loop
```

2. continue Statement:

- It returns the control to the beginning of the while loop. The continue statement rejects all the remaining statements in the current iteration of the loop and moves the control back to the top of the loop. The continue statement can be used in both *while* and *for* loops.

Syntax: `Continue`

Program 1.18: Example for continue statement.

```
for letter in 'Sybca':
 if letter == 'b':
 continue
 print('Current Letter:', letter)
```

Output:

```
Current Letter: S
Current Letter: y
Current Letter: c
Current Letter: a
```

Program 1.19: Example for continue statement.

```
var = 5
while var > 0:
 var = var -1
 if var == 3:
 continue
 print('Current variable value:', var)
print('In loop')
```

Output:

```
Current variable value: 4
In loop
Current variable value: 2
In loop
Current variable value: 1
In loop
Current variable value: 0
In loop
```

3. **pass Statement:**

- It is used when a statement is required syntactically but you do not want any command or code to execute. The *pass* statement is a *null* operation; nothing happens when it executes.

Syntax:

```
pass
```

Program 1.20: Example for pass statement.

```
for letter in 'Sybca':
 if letter == 'b':
 pass
 print('This is pass block')
 print('Current Letter:', letter)
```

Output:

```
This is pass block
Current Letter: S
This is pass block
Current Letter: y
This is pass block
Current Letter: b
This is pass block
Current Letter: c
This is pass block
Current Letter: a
```

1.6 STRING MANIPULATION-ACCESSING STRING, BASIC OPERATIONS, STRING SLICES, FUNCTION AND METHODS, EXAMPLES

- A Python string is a sequence of characters. Python Strings are immutable (unchanging) sequences of Unicode points.

```
# all of the following are equivalent Examples
my_string = 'Hello'
print(my_string)

my_string = "Hello"
print(my_string)

my_string = '''Hello'''
print(my_string)
```



```
# triple quotes string can extend multiple lines
my_string = """Hello, welcome to
 the world of Python"""
print(my_string)
```

When you run the program, the output will be:

```
Hello
Hello
Hello
Hello, welcome to
 the world of Python
```

String Representation

- In Python, Strings are stored as individual characters in a contiguous memory location.
- The benefit of using String is that it can be accessed from both the directions (forward and backward).
- Both forward as well as backward indexing are provided using Strings in Python.
  - Forward indexing starts with 0,1,2,3,....
  - Backward indexing starts with -1,-2,-3,-4,....

Example:

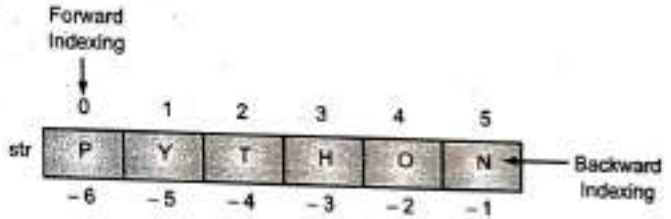


Fig. 1.22

str[0] = 'P' = str[-6], str[1] = 'Y' = str[-5], str[2] = 'T' = str[-4], str[3] = 'H' = str[-3], str[4] = 'O' = str[-2], str[5] = 'N' = str[-1].

1.6.1 Accessing String

- Python does not support a character type; these are treated as strings of length one, thus also considered a substring. Square brackets can be used to access elements of the string.
- We can access individual characters using indexing and a range of characters using slicing. Index starts from 0. Trying to access a character out of index range will raise an IndexError. The index must be an integer. We can't use floats or other types, this will result into TypeError.
- Python allows negative indexing for its sequences.
- The index of -1 refers to the last item, -2 to the second last item and so on.

Program 1.21: Accessing values in strings.

```
msg1 = 'Hello World!'
#first character
print("msg1[0]= ", msg1[0])
#last character
print("msg1[-1] = ", msg1[-1])
```

Output:

```
msg1[0]= H
msg1[-1]= !
```

1.6.2 Basic Operations

- Assume string variable **a** holds 'Hello' and variable **b** holds 'Python', then:

Operator	Description	Example
+	Concatenation - Adds values on either side of the operator	a + b give output as HelloPython
*	Repetition - Creates new strings, concatenating multiple copies of the same string	a*2 give output as -HelloHello
[]	Slice - Gives the character from the given Index	a[1] give output as e
[:]	Range of Slice - Gives the characters from the given range	a[1:4] give output as ell
in	Membership - Returns true if a character exists in the given string	H in a given output as 1
not in	Membership - Returns true if a character does not exist in the given string	M not in a given output as 1
%	Format - Performs String formatting	

Program 1.22: Python program for concatenation of two strings.

```
str1 = 'Hello'
str2 ='World!'
# using + Operator
print('str1 + str2 = ', str1 + str2)
```

Output:

```
str1 + str2 = HelloWorld!
```

1.6.3 String Slices

- The "slice" syntax is a handy way to refer to sub-parts of sequences -- typically strings and lists. The slice s[start:end] is the elements beginning at start and extending up to but not including end.
- We can access a range of items in a string by using the slicing operator :(colon).

- As an alternative, Python uses negative numbers to give easy access to the chars at the end of the string: s[-1] is the last char 'o', s[-2] is 'l' the next-to-last char, and so on.
- **Example:** Suppose we have s = "Hello"

  s[1:4] is 'ell' -- chars starting at index 1 and extending up to but not including index 4

  s[1:] is 'ello' -- omitting either index defaults to the start or end of the string

  s[:] is 'Hello' -- omitting both always gives us a copy of the whole thing (this is the pythonic way to copy a sequence like a string or list)

  s[-1] is 'o' -- last char (1st from the end)

  s[-4] is 'e' -- 4th from the end

Program 1.23: Program for array slicing.

```
msg1 = 'Python'
print("msg1[1:5] = ", msg1[1:5])
```

Output:

```
msg1[1:5] = ytho
```

1.6.4 Function and Methods

- Python has several built-in functions and methods associated with the string data type. In Python, the following built-in functions and methods used to manipulate and modify strings:

1. **Capitalize():** Capitalizes first letter of string
2. **count(str, beg= 0, end=len(string)):** Counts how many times str occurs in string or in a substring of string if starting index beg and ending index end are given.
3. **endswith(suffix, beg=0, end=len(string)):** Determines if string or a substring of string (if starting index beg and ending index end are given) ends with suffix; returns true if so and false otherwise.
4. **isalnum():** Returns true if string has at least 1 character and all characters are alphanumeric and false otherwise.
5. **isalpha():** Returns true if string has at least 1 character and all characters are alphabetic and false otherwise.
6. **isdigit():** Returns true if string contains only digits and false otherwise.
7. **islower():** Returns true if string has at least 1 cased character and all cased characters are in lowercase and false otherwise.
8. **isnumeric():** Returns true if a unicode string contains only numeric characters and false otherwise.
9. **isspace():** Returns true if string contains only whitespace characters and false otherwise.
10. **istitle():** Returns true if string is properly "titlecased" and false otherwise.
11. **isupper():** Returns true if string has at least one cased character and all cased characters are in uppercase and false otherwise.

12. **join(seq)**: Merges (concatenates) the string representations of elements in sequence seq into a string, with separator string.
13. **len(string)**: Returns the length of the string.
14. **ljust(width[, fillchar])**: Returns a space-padded string with the original string left-justified to a total of width columns.
15. **lower()**: Converts all uppercase letters in string to lowercase.
16. **lstrip()**: Removes all leading whitespace in string.
17. **maketrans()**: Returns a translation table to be used in translate function.
18. **max(str)**: Returns the max alphabetical character from the string str.
19. **min(str)**: Returns the min alphabetical character from the string str.
20. **replace(old, new [, max])**: Replaces all occurrences of old in string with new or at most max occurrences if max given.
21. **rfind(str, beg=0,end=len(string))**: Same as find(), but search backwards in string.
22. **rjust(width,[, fillchar])**: Returns a space-padded string with the original string right-justified to a total of width columns.
23. **rstrip()**: Removes all trailing whitespace of string.
24. **split(str="", num=string.count(str))**: Splits string according to delimiter str (space if not provided) and returns list of substrings; split into at most num substrings if given.
25. **splitlines(num=string.count('\n'))**: Splits string at all (or num) NEWLINEs and returns a list of each line with NEWLINEs removed.
26. **swapcase()**: Inverts case for all letters in string.
27. **title()**: Returns "titlecased" version of string, that is, all words begin with uppercase and the rest are lowercase.
28. **translate(table, deletechars="")**: Translates string according to translation table str(256 chars), removing those in the del string.
29. **upper()**: Converts lowercase letters in string to uppercase. E.g print s.upper() #s is string.
30. **zfill(width)**: Returns original string leftpadded with zeros to a total of width characters; intended for numbers, zfill() retains any sign given (less one zero).
31. **isdecimal()**: Returns true if a unicode string contains only decimal characters and false otherwise.
32. **center(width, fillchar)**: Returns a space-padded string with the original string centered to a total of width columns.
33. **encode()**: Returns encoded string version of string; on error, default is to raise a ValueError unless errors is given with 'ignore' or 'replace'.
34. **index()**: Returns Index of Substring
35. **find(str, beg=0 end=len(string))**: Determine if str occurs in string or in a substring of string if starting index beg and ending index end are given returns index if found and -1 otherwise.

Program 1.24: Write a Python program to get a string made of the first 2 and the last 2 chars from a given a string. If the string length is less than 2, return instead of the empty string.

```
string=raw_input("Enter string:")
count=0
for i in string:
   count=count+1
new=string[0:2]+string[count-2:count]
print("Newly formed string is:")
print(new)
```

Output:

```
Enter string: Hello world
Newly formed string is: Held
```

Examples:

Program 1.25: Write a Python program to get a string from a given string where all occurrences of its first char have been changed to '$'.

```
str1input:{"Enter a String:",}
print("Original String:",str1}
char = str1[0]
str1 = str1.replace(char, '$')
str1 = char + str1[1:]
print("Replaced String:",str1}
```

Output:

```
Enter  a String: onion
Original String: onion
Replaced String: oni$n
```

Program 1.26: Python Program to Replace all Occurrences of 'a' with $ in a String.

```
string=raw_input("Enter string:")
string=string.replace('a','$')
string=string.replace('A','$')
print("Modified string:")
print(string)
```

Output:

```
Enter string:Asia
Modified string:
$si$
```

Program 1.27: Write a python program to count repeated characters in a string.

Sample string: 'thequickbrownfoxjumpsoverthelazydog'

```
import collections
str1 = 'thequickbrownfoxjumpsoverthelazydog'
d = collections.defaultdict(int)
for c in str1:
    d[c] += 1
for c in sorted(d, key.d.get, reverse.True):
    if d[c] > 1:
        print('%s %d' % (c, d[c]))
```

Output:

```
20
o 4
e 3
u 2
h 2
r 2
t 2
```

1.7 LISTS-INTRODUCTION, ACCESSING LIST, OPERATIONS, WORKING WITH LISTS, FUNCTION & METHODS

- A Python list is a mutable sequence of data values called items or elements. An item can be of any type. List is one of the most frequently used and very versatile datatype used in Python.

Creating Elements in a List:

- In Python programming, a list is created by placing all the items (elements) inside a square bracket [ ], separated by commas.
- It can have any number of items and they may be of different types (integer, float, string etc.).

Syntax: `<list_name>=[value1,value2,value3,...,valuen]`

Empty List:

```
list1 = []
```

- The empty Python list is written as two square brackets containing nothing.

Examples of different lists:

```
list1 = ['C', 'java', 2016, 2018]   # the items in a list need not be of the same type.
list2 = [1, 2, 3, 4, 5 ]
list3 = ["a", "b", "c", "d"]
```


Example:

```
list1 = ["apple", "banana", "cherry"]
print(list1)
```

Output:

```
["apple", "banana", "cherry"]
```

1.7.1 Accessing List

- There are various ways in which we can access the elements of a list.

List Index:

- We can use the index operator[ ] to access an item in a list. Index starts from 0. So, a list having 5 elements will have index from 0 to 4.
- Trying to access an element other that this will raise an IndexError. The index must be an integer. We can't use float or other types, this will result into TypeError.

Negative Indexing:

- Python allows negative indexing for its sequences. The index of -1 refers to the last item, -2 to the second last item and so on.

Slicing List:

- We can access a range of items in a list by using the slicing operator: (colon).
- Slicing can be best visualized by considering the index to be between the elements. So if we want to access a range, we need two index that will slice that portion from the list.
- Basic Syntax to Access Python List is:

  `<list_name>[index]`

Examples:

```
list1=[1,2,3,4]
list2=['a','b','c']
print(list1[0])
print(list1[0:2])
print(list2[-3:-1])
print(list1[0:])
print(list2[:2])
```

Output:

```
1
[1, 2]
['a', 'b']
[1, 2, 3, 4]
['a', 'b']
```

1.7.2 List Operation

- Lists respond to the + and * operators much like strings; they mean concatenation and repetition here too, except that the result is a new list, not a string.
- In fact, lists respond to all of the general sequence operations we used on strings in the previous section.

Python Expression	Results	Description
len([4, 5, 6, 7])	4	Length
[1, 2, 3] + [4, 5, 6]	[1, 2, 3, 4, 5, 6]	Concatenation
['Hi!'] * 3	['Hi!', 'Hi!', 'Hi!']	Repetition
3 in [1, 2, 3]	True	Membership
for i in [1, 2, 3]: print i,	1 2 3	Iteration

1.7.3 Working with Lists

1.7.3.1 Updating List

- The single or multiple elements of lists can be updated by giving the slice on the left-hand side of the assignment operator. i.e. We can use assignment operator (=) to change an item or a range of items.
- We can add one item to a list using append() method or add several items using extend() method.
- We can also use + operator to combine two lists. This is also called concatenation.
- The * operator repeats a list for the given number of times.

Example:

```
list1 = ['C', 'java', 2016, 2018];
print("Value available at index 2: ")
print(list1[2])
list1[2]= 2000
print(list1)
```

Output:

```
Value available at index 2:
2016
['C', 'java', 2000, 2018]
```

Example:

```
list2 = ["apple", "banana", "cherry"];
list2.append("orange");
print(list2)
```

Output:

```
["apple", "banana", "cherry", "orange"]
```


1.7.3.2 Delete List

- We can delete one or more items from a list using the keyword del. It can even delete the list entirely. We can use del statement(if you know exactly which elements you are deleting).
- We can use remove() method to remove the given item or pop() method to remove an item at the given index. The remove() method is used to delete the elements from a list if we do not know which element we are deleting.
- The pop() method removes and returns the last item if index is not provided. This helps us implement lists as stacks (first in, last out data structure).
- We can also use the clear() method to empty a list.

Example:

```
list1 = ['C', 'java', 2016, 2018];
print list1
del list1[2];
print "After deleting:", list1
list1.remove('java');
print list1
```

Output:

```
['C', 'java', 2016, 2018]
After deleting: ['C', 'java', 2018]
['C', 2018]
```

1.7.4 Functions and Methods

Python includes the following list functions:

- **all()**: Return True if all elements of the list are true (or if the list is empty).
- **any()**: Return True if any element of the list is true. If the list is empty, return False.
- **enumerate()**: Return an enumerate object. It contains the index and value of all the items of list as a tuple.
- **len()**: Return the length (the number of items) in the list.
- **list()**: Convert an iterable (tuple, string, set, dictionary) to a list.
- **max()**: Return the largest item in the list.
- **min()**: Return the smallest item in the list.
- **sorted()**: Return a new sorted list (does not sort the list itself).
- **sum()**: Return the sum of all elements in the list.

Python includes following list methods:

- **list.append(obj)**: Appends object obj to list.
- **list.count(obj)**: Returns count of how many times obj occurs in list.

- **list.extend(seq)**: Appends the contents of seq to list.
- **list.index(obj)**: Returns the lowest index in list that obj appears.
- **list.insert(index, obj)**: Inserts object obj into list at offset index.
- **list.pop(obj=list[-1])**: Removes and returns last object or obj from list.
- **list.remove(obj)**: Removes object obj from list.
- **list.reverse()**: Reverses objects of list in place.
- **list.sort([func])**: Sorts objects of list, use compare func if given.
- **clear()**: Removes all items from the list.
- **copy()**: Returns a shallow copy of the list.

Program 1.28: For list methods.

```
lst = [123, 'xyz', 'pqr', 'abc', 'xyz'];
lst.reverse();
print("List: ", lst)
print("Count for 123: ", lst.count(123))
print("Count for xyz: ", lst.count('xyz'))
lst.remove('xyz')
print("List: ",lst)
lst.remove('abc')
print("List: ", lst)
lst.pop()
print(lst)
```

Output:

```
List: ['xyz', 'abc', 'pqr', 'xyz', 123]
Count for 123: 1
Count for xyz: 2
List: ['abc', 'pqr', 'xyz', 123]
List: ['pqr', 'xyz', 123]
['pqr', 'xyz']
```

1.8 TUPLE-INTRODUCTION, ACCESSING TUPLES, OPERATIONS WORKING, FUNCTION AND METHODS, EXAMPLES

- A tuple is a collection which is ordered and **unchangeable**. In Python tuples are written with round brackets. This Python Data Structure is like a list in Python, is a heterogeneous container for objects.
- The differences between tuples and lists are, we cannot change the elements of a tuple once it is assigned whereas in a list, elements can be changed. This means that while you can reassign or delete an entire tuple, you cannot do the same to a single item or a slice. Tuples use parentheses, whereas lists use square brackets.

Advantages of Tuple:

1. Processing of Tuples are faster than Lists.
2. It makes the data safe because tuples are immutable and hence cannot be changed.
3. Generally tuples are used for heterogeneous (different) datatypes and list for homogeneous (similar) datatypes.
4. Tuples are used for String formatting.

Difference between Tuples and Lists:

1. The syntax of tuples is shown by parenthesis() whereas the syntax of lists is shown by square brackets[ ].
2. List has variable length, tuple has fixed length.
3. List has mustable nature, tuple has immutable nature.
4. List has more functionality than the tuple.
5. Typles are heterogeneous while lists are homogeneous. One has to deal individually with the items.
6. Tuples show structure whereas lists show order.

Creating Tuple:

- A tuple is created by placing all the items (elements) inside a parentheses (), separated by comma. The parentheses are optional but is a good practice to write it. A tuple can have any number of items and they may be of different types (integer, float, list, string etc.).
- For example:

```
tup1 = ("apple", "orange", 2018);  tup2 = (1, 2, 3, 4, 5);
tup3 = "a", "b", "c", "d";
```

- The empty tuple is written as two parentheses containing nothing.

```
tup1 = ();
```

- To write a tuple containing a single value you have to include a comma, even though there is only one value:

```
tup1 = (50,);
```

- Like string indices, tuple indices start at 0, and they can be sliced, concatenated, and so on.

Program 1.29: Python program for creating Tuples.

```
# empty tuple
my_tuple = ()
print(my_tuple)

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

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

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

# tuple can be created without parentheses also called tuple packing
my_tuple = 3, 4.6, "tybca"
print(my_tuple)

# tuple unpacking is also possible
a, b, c = my_tuple
print(a)
print(b)
print(c)
```

Output:

```
()
(1, 2, 3)
(1, 'Hello', 3.4)
('Hello', [8, 4, 6], (1, 2, 3))
(3, 4.6, 'tybca')
3
4.6
tybca
```

1.8.1 Accessing Tuples

- To access values in tuple, use the square brackets for slicing along with the index o indices to obtain value available at that index.

For example:

```
tup1 = ("apple", "orange", 2018);
tup2 = (1, 2, 3, 4, 5, 6, 7 )
print "tup1[0]: ", tup1[0]
print "tup2[1:6]: ", tup2[1:6]
```

When the above code is executed, it produces the following result:

```
tup1[0]: apple
tup2[1:6]: (2, 3, 4, 5, 6)
```


- **Python Tuple Packing**

  Python Tuple packing is the term for packing a sequence of values into a tuple without using parentheses.

```
>>> mytuple=1,2,3 #Or it could have been mytuple=1,2,3
>>> mytuple
```

- **Python Tuple Unpacking**

  The opposite of tuple packing, unpacking allots the values from a tuple into a sequence of variables.

```
>>> a,b,c = mytuple
>>> print(a,b,c)
1 2 3
```

1.8.2 Tuple Operations

- Tuples respond to the + and * operators much like strings; they mean concatenation and repetition here too, except that the result is a new tuple, not a string. The various operations that we can perform on tuples are very similar to lists.

1. **Concatenation:**

- Using the addition operator +, with two or more tuples, adds up all the elements into a new tuple.

  Example: (1, 2, 3) + (4, 5, 6)

  Output: (1, 2, 3, 4, 5, 6)

2. **Repetition:**

- Multiplying a tuple by any integer, x will simply create another tuple with all the elements from the first tuple being repeated x number of times. For example, t*3 means, elements of tuple t will be repeated 3 times.

  Example: `('Hi!',) * 4`

  Output: `('Hi!', 'Hi!', 'Hi!', 'Hi!')`

3. **Tuple Membership Test using in keyword:**

- In keyword, can not only be used with tuples, but also with strings and lists too. It is used to check, if any element is present in the sequence or not. It returns True if the element is found, otherwise False.

  Example:

```
#in operation
tup = ('a','p','p','l','e',)
print('a' in tup)          # Output: True
print('b' in tup)          # Output: False
# Not in operation
print('g' not in tup)      # Output: True
```

4. **Iterating Through a Tuple:**

Using a for loop we can iterate though each item in a tuple.

Example:

```
for name in ('mom','dad'):
 print("Hello",name)
```

Output:

```
# Hello mom
# Hello dad
```

Python Expression	Results	Description Length
(1, 2, 3) + (4, 5, 6)	(1, 2, 3, 4, 5, 6)	Concatenation
('Hi!',) * 4	('Hi!', 'Hi!', 'Hi!', 'Hi!')	Repetition
3 in (1, 2, 3)	True	Membership
for x in (1, 2, 3): print x,	1 2 3	Iteration

1.8.3 Working of Tuple

1.8.3.1 Updating Tuple

- Tuples are immutable which means you cannot update or change the values of tuple elements. You are able to take portions of existing tuples to create new tuples as the following example demonstrates:

```
tup1 = (10, 32.76)
tup2 = ('abc', 'xyz')
# tup1[0] = 100 is not valid
# So create a new tuple as follows:
tup3 = tup1 + tup2
print tup3
```

When the above code is executed, it produces the following result:

```
(10, 32.76, 'abc', 'xyz')
```

- We can use + operator to combine two tuples. This is also called **concatenation.**
- We can also **repeat** the elements in a tuple for a given number of times using the * operator.

```
>>> (1,) * 5
(1, 1, 1, 1, 1)
```

- Both + and * operations result into a new tuple.

1.8.3.2 Deleting Tuples

Deleting Tuple:

- Removing individual tuple elements is not possible. There is, of course, nothing wrong with putting together another tuple with the undesired elements discarded.

- To explicitly remove an entire tuple, just use the **del** statement.

Example:

```
tup = ("apple", "orange", 2018);
print tup
del tup
print "After deleting tup: "
print tup
```

Output:

```
('apple', 'orange', 2018);
   After deleting tup:
      Traceback (most recent call last):
       File "test.py", line 5, in <module>
       print tup;
      NameError: name 'tup' is not defined
```

- This produces an exception, this is because after **del tup** tuple does not exist any more.

Deleting elements of tuple:

- One way to delete the elements from the tuple , we have to convert into list and then perform deletion.

Program 1.30: Write Python program for deleting elements from Tuple.

```
# Python program to remove an item from a tuple.
#create a tuple
tup = "a", "b", "c", "d", "e", "f", "g", "h", 10
print(tup)
#tuples are immutable, so you can not remove elements
#using merge of tuples with the + operator you can remove an item and it
will create a new tuple
tup = tup[:2] + tup[3:]
print(tup)
#converting the tuple to list
lst = list(tup)
#use different ways to remove an item of the list
lst.remove("h")
#converting the tuple to list
tup = tuple(lst)
print(tup)
```

Output:

```
('a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 10)
('a', 'b', 'd', 'e', 'f', 'g', 'h', 10)
('a', 'b', 'd', 'e', 'f', 'g', 10)
```

1.8.4 Function and Methods

Functions:

- Python includes the following tuple functions:
  1. **cmp(tuple1, tuple2):**
     - Compares elements of both tuples.
     - This is used to compare two tuples. It will return either 1, 0 or -1, depending upon whether the two tuples being compared are similar or not.
     - The cmp() function takes two tuples as arguments, where both of them are compared. If **T1** is the first tuple and **T2** is the second tuple, then:
       - if **T1 > T2**, then cmp(T1, T2) returns **1**
       - if **T1 = T2**, then cmp(T1, T2) returns **0**
       - if **T1 > T2**, then cmp(T1, T2) returns **-1**
  2. **len(tuple):**
     - This function is used to get the number of elements inside any tuple.

     Example: `len((1, 2, 3)) output: 3`
  3. **max(tuple):** Returns item from the tuple with max value.
  4. **min(tuple):** Returns item from the tuple with min value.
  5. **tuple(seq):** This method converts a list of items into tuples.

Program 1.31: Python program for built in Tuple function.

```
aList = ('xyz', 'pqr', 'abc');
aTuple = tuple(aList)
print("Tuple elements: ", aTuple)
# use of max() method:
print("Max value element: ", max(aTuple))
# use of min() method:
print("Max value element: ", min(aTuple))
# use of len() method:
tuple1, tuple2 = ('xyz', 'pqr','spq'), (450, 'abc')
print("First tuple length: ", len(tuple1))
print("Second tuple length: ", len(tuple2))
```

Output:

```
Tuple elements: ('xyz', 'pqr', 'abc')
Max value element: xyz
Max value element: abc
First tuple length: 3
Second tuple length: 2
```


Methods:

Python Tuple Methods:

- A method is a sequence of instructions to perform on something. Unlike a function, it does modify the construct on which it is called. You call a method using the dot operator in python. Python has two built-in methods that you can use on tuples.

1. **index(x):** Return index of first item that is equal to x.

   Example:

   ```
   >>> a=(1,2,3,2,4,5,2)
   >>> a.index(2)
   ```

   Output: 1

   As you can see, we have 2s at indices 1, 3, and 6. But it returns only the first index.
2. **Count(x):** Return the number of items is equal to x.

   Example:

   ```
   >>> a=(1,2,3,2,4,5,2)
   >>> a.count(2)
   ```

   Output: 3

Program 1.32: Write a Python program to create a list of tuples with the first element as the number and second element as the square of the number.

```
l_range=int(input("Enter the lower range:"))
u_range=int(input("Enter the upper range:"))
a=[(x,x**2) for x in range(l_range,u_range+1)]
print(a)
```

Output:

```
Enter the lower range:1
Enter the upper range:4
[(1, 1), (2, 4), (3, 9), (4, 16)]
```

Examples:

Program 1.33: Write a Python program to add an item in a tuple.

```
#create a tuple
tuplex = (4, 6, 2, 8, 3, 1)
print(tuplex)
#tuples are immutable, so you can not add new elements
#using merge of tuples with the + operator you can add an element and it
will create a new tuple
tuplex = tuplex + (9,)
print(tuplex)
#adding items in a specific index
```

```
tuplex = tuplex[:5] + (15, 20, 25) + tuplex[:5]
print(tuplex)
#converting the tuple to list
listx = list(tuplex)
#use different ways to add items in list
listx.append(30)
tuplex = tuple(listx)
print(tuplex)
```

Output:

```
(4, 6, 2, 8, 3, 1)
(4, 6, 2, 8, 3, 1, 9)
(4, 6, 2, 8, 3, 15, 20, 25, 4, 6, 2, 8, 3)
(4, 6, 2, 8, 3, 15, 20, 25, 4, 6, 2, 8, 3, 30)
```

Program 1.34: Write a Python program to convert a tuple to a string.

```
tup = ('e', 'x', 'e', 'r', 'c', 'i', 's', 'e', 's')
str = ''.join(tup)
print(str)
```

Output:

```
exercises
```

Program 1.35: Write a Python program to get the 4th element from front and 4th element from last of a tuple.

```
#Get an item of the tuple
tuplex = ("p", "y", "t", "h", "o", "n", "b", "o", "o", "k")
print(tuplex)
#Get item (4th element)of the tuple by index
item = tuplex[3]
print(item)
#Get item (4th element from last)by index negative
item1 = tuplex[-4]
print(item1)
```

Output:

```
('p', 'y', 't', 'h', 'o', 'n', 'b', 'o', 'o', 'k')
h
b
```


Program 1.36: Write a Python program to find the repeated items of a tuple.

```
#create a tuple
tuplex = 2, 4, 5, 6, 2, 3, 4, 4, 7
print(tuplex)
#return the number of times it appears in the tuple.
count = tuplex.count(4)
print(count)
```

Output:

```
(2, 4, 5, 6, 2, 3, 4, 4, 7)
3
```

Program 1.37: Write a Python program to check whether an element exists within a tuple.

```
tuplex = ("w", 3, "r", "e", "s", "o", "u", "r", "c", "e")
print("r" in tuplex)
print(5 in tuplex)
```

Output:

```
True
False
```

1.9 DICTIONARIES-INTRODUCTION, ACCESSING VALUES IN DICTIONARIES, WORKING WITH DICTIONARIES, PROPERTIES, FUNCTION, EXAMPLES

- A dictionary is a collection which is unordered, changeable and indexed. In Python dictionaries are written with curly brackets, and they have keys and values.
- Dictionaries are optimized to retrieve values when the key is known.
- Dictionaries are Python's implementation of a data structure that is more generally known as an associative array. A dictionary consists of a collection of key-value pairs. Each key-value pair maps the key to its associated value.

Creation of Dictionary:

- Creating a dictionary is as simple as placing items inside curly braces {} separated by comma. An item has a key and the corresponding value expressed as a pair, key: value.
- Each key is separated from its value by a colon (:), the items are separated by commas, and the whole thing is enclosed in curly braces. An empty dictionary without any items is written with just two curly braces, like this: {}.
- Keys are unique within a dictionary while values may not be. The values of a dictionary can be of any type, but the keys must be of an immutable data type such as strings, numbers, or tuples.

Syntax:

```
dictionary_name = {
 <key>: <value>,
 <key>: <value>,
 .
 .
 .
 <key>: <value>
}
```

Or

```
dictionary_name = dict([
 (<key>, <value>),
 (<key>, <value),
 .
 .
 .
 (<key>, <value>)
])
```

Where dict is built-in function.

Example:

```
# empty dictionary with name dict
dict = {}

# dictionary with integer keys
dict = {1: 'apple', 2: 'orange'}

# dictionary with mixed keys
dict = {'name': 'abc', 1: [2, 4, 3]}

# using dict()
dict = dict({1:'apple', 2:'orange'})

# from sequence having each item as a pair
dict = dict([(1,'apple'), (2,'orange')])
```

1.9.1 Accessing Values in Dictionary

- To access dictionary elements, square brackets are used along with the key to obtain its value.

To access dictionary elements we need to pass key, associated to the value.

Syntax:

```
<dictionary_name>[key]
```

Example:

```
dict = {'Name': 'abc', 'Age': 20, 'Class': 'tybca'}
print "dict['Name']: ", dict['Name']
print "dict['Age']: ", dict['Age']
```

Output:

```
dict['Name']: abc
dict['Age']: 20
```

- If we attempt to access a data item with a key, which is not part of the dictionary, we get an error.

```
dict = {'Name': 'abc', 'Age': 20, 'Class': 'tybca'}
print "dict['xyz']: ", dict['xyz']
```

Output:

```
dict['xyz']:
Traceback (most recent call last):
 File "test.py", line 2, in <module>
 print "dict['xyz']: ", dict['xyz'];
KeyError: 'xyz'
```

1.9.2 Working with Dictionaries

1.9.2.1 Updating Dictionary

- A dictionary can be updated by adding a new entry or a key-value pair, modifying an existing entry, or deleting an existing entry as shown below:

Program 1.38:

```
dict = {'Name': 'abc', 'Age': 20, 'Class': 'tybca'}
dict['Age'] = 22 ; # update existing entry
dict['College'] = "College"; # Add new entry

print("dict['Age']: ", dict['Age'])
print("dict['College']: ", dict['College'])
```

Output:

```
dict['Age']: 22
dict['College']: college
```

Example: Python program for updating Dictionary.

```
dict1 = {'name':'Omkar', 'age': 20}
# update value
dict1['age'] = 25
#Output: {'age': 25, 'name': 'Omkar'}
print(dict1)
# add item
dict1['address'] = 'pune'
# Output: {'address': 'pune', 'age': 25, 'name': 'Omkar'}
print(dict1)
```

Program 1.39: Print all key names in the dictionary, one by one.

```
dict1 = {'name':'Omkar', 'address': 'pune', 'phone':9897965867}
for x in dict1: print(x)
```

Output:

```
phone
name
address
```

Program 1.40: Program to print all values of dictionary.

```
dict1 = {'name':'Omkar', 'address': 'pune', 'phone':9897965867}
for x in dict1.values():
 print(x)
```

Output:

```
Omkar
9897965867
pune
```

Program 1.41: Program to print all keys and values of dictionary.

```
dict1 = {'name':'Omkar', 'address': 'pune', 'phone':9897965867}
for x, y in dict1.items():
 print(x, y)
```

Output:

```
address pune
phone 9897965867
name Omkar
```


1.9.2.2 Delete Dictionary Elements

- A dictionary element can be removed one at a time or can clear the entire contents of a dictionary. To explicitly remove an entire dictionary, the **del** statement is used. The del keyword removes the item with the specified key name:

1. **Example using del:**

```
dict1 = {'Name': 'abc', 'Age': 20, 'Class': 'tybca'}
del dict1['Name']; # remove entry with key 'Name'
dict1.clear(); # remove all entries in dict
del dict1 ; # delete entire dictionary
print "dict1['Age']: ", dict1['Age']
```

This produces the following result. Note that an exception is raised because after **del dict** dictionary does not exist any more.

```
dict['Age']:
Traceback (most recent call last):
 File "test.py", line 5, in <module>
 print "dict['Age']: ", dict['Age'];
TypeError: 'type' object is unsubscriptable
```

2. **The clear() method empties the dictionary:**

Example using clear():

```
dict1 = {'name':'Omkar', 'address': 'pune', 'phone':9897965867}
dict1.clear()
print(dict1)
# Output: {}
```

3. **The pop() method removes the item with the specified key name:**

Example using pop():

```
dict1 = {'name':'Omkar', 'address': 'pune', 'phone':9897965867}
dict1.pop('address')
print(dict1)
# Output: {'name':'Omkar', 'phone':9897965867}
```

4. **The popitem() method removes the last inserted item.**

Example using popitem():

```
dict1 = {'name':'Omkar', 'address': 'pune', 'phone':9897965867}
dict1.popitem( )
print(dict1)
# Output: {'name':'Omkar', 'address': 'pune'}
```

Program 1.42: To delete and remove elements from dictionary:

```
squares = {1:1, 2:4, 3:9, 4:16, 5:25}
# remove a particular item
print(squares.pop(4))
print(squares)
# remove an arbitrary item
print(squares.popitem())
print(squares)
# delete a particular item
del squares[3]
print(squares)
# remove all items
squares.clear()
print(squares)
# delete the dictionary itself
del squares
print(squares)
```

Output:

```
16
{1: 1, 2: 4, 3: 9, 5: 25}
(1, 1)
{2: 4, 3: 9, 5: 25}
{2: 4, 5: 25}
{}
NameError: name 'squares' is not defined
```

1.9.3 Properties

- Dictionary values have no restrictions. They can be any arbitrary Python object, either standard objects or user-defined objects. However, same is not true for the keys.
- There are two important points to remember about dictionary keys:
  (a) More than one entry per key not allowed. Which means no duplicate key is allowed. When duplicate keys encountered during assignment, the last assignment wins.
  (b) Keys must be immutable. Which means you can use strings, numbers or tuples as dictionary keys but something like ['key'] is not allowed.
- In a dictionary, we can add any number of values for one key. Python object, either standard objects or user-defined objects.

Example:

```
>>> d = {0: 'a', 1: 'a', 2: 'a', 3: 'a'}
>>> d
{0: 'a', 1: 'a', 2: 'a', 3: 'a'}
>>> d[0] == d[1] == d[2]
True
```

- More than one entry per key not allowed. Which means no duplicate key is allowed. When duplicate keys encountered during assignment, the last assignment wins, means the second occurrence will override the first.

Example:

```
dict = {'Name': 'Omkar', 'Age': 20, 'Name': 'Mani'}
print "dict['Name']: ", dict['Name']
```

Output:

```
dict['Name']: Mani
```

- Keys must be immutable. Which means you can use strings, numbers or tuples as dictionary keys but something like ['key'] is not allowed.

Example:

```
dict = {['Name']: 'Omkar', 'Age': 20}
print "dict['Name']: ", dict['Name']
# Output: error
```

Program having a tuple as a dictionary key, because tuples are immutable:

```
>>> d = {(1, 1): 'a', (1, 2): 'b', (2, 1): 'c', (2, 2):'d'}
>>> d[(1,1)]
'a'
>>> d[(2,1)]
'c'
```

1.9.4 Function and Methods

Python includes the following dictionary functions:

1. **cmp(dict1, dict2):** Compares elements of both dict.
2. **len(dict):** Gives the total length of the dictionary. This would be equal to the number of items in the dictionary.
3. **str(dict):** Produces a printable string representation of a dictionary.
4. **type(variable):** Returns the type of the passed variable. If passed variable is dictionary, then it would return a dictionary type.

Example:

```
dict = {'Name': 'abc', 'Age':20};
#usage of len() method
print "Length: %d" % len (dict)  #Output:Length: 2
# usage of type() method
```

```
print "Variable Type: %s" % type(dict)
#Output:Variable Type: <type 'dict'>
#usage of str() method.
print "Equivalent String: %s" % str (dict)
#Output: Equivalent String: {'Age': 20, 'Name': 'abc'}
```

Python includes following dictionary methods:

1. **dict.clear()**: Removes all elements of dictionary *dict*.
2. **dict.copy()**: Returns a shallow copy of dictionary *dict*.
3. **dict.fromkeys()**: Create a new dictionary with keys from seq and values set to *value*.
4. **dict.get(key, default=None)**: For *key* key, returns value or default if key not in dictionary.
5. **dict.has_key(key)**: Returns *true* if key in dictionary *dict*, *false* otherwise.
6. **dict.items()**: Returns a list of *dict*'s (key, value) tuple pairs.
7. **dict.keys()**: Returns list of dictionary dict's keys.
8. **dict.setdefault(key, default=None)**: Similar to get(), but will set dict[key]=default if *key* is not already in dict.
9. **dict.update(dict2)**: Adds dictionary *dict2*'s key-values pairs to *dict*.
10. **dict.values()**: Returns list of dictionary *dict*'s values.

Program 1.43: To show usage of all methods of dictionary.

```
dict = {'Name': 'abc', 'Age': 20}
# usage of items() method.
print("Value: %s" % dict.items())

# usage of update() method.
dict = {'Name': 'abc', 'Age': 20}
dict1 = {'Sex': 'female' }
dict.update(dict1)
print("Value: %s" % dict)

# usage of keys() method.
print("Value: %s" % dict.keys())

# usage of values() method.
print("Value: %s" % dict.values())

# usage of copy() method.
dict2 = dict.copy()
print("New Dictionary: %s" % str(dict2))

# clear() method.
print("Start Len: %d" % len(dict))

dict.clear()
print("End Len: %d" % len(dict))
```


Output:

```
Value: dict_items([('Name', 'abc'), ('Age', 20)])
Value: {'Name': 'abc', 'Age': 20, 'Sex': 'female'}
Value: dict_keys(['Name', 'Age', 'Sex'])
Value: dict_values(['abc', 20, 'female'])
New Dictionary: {'Name': 'abc', 'Age': 20, 'Sex': 'female'}
Start Len: 3
End Len: 0
```

Examples:

Program 1.44: Write a Python script to sort (ascending and descending) a dictionary by value.

```
import operator
d = {1: 2, 3: 4, 4: 3, 2: 1, 0: 0}
print('Original dictionary : ',d)
sorted_d = dict(sorted(d.items(), key=operator.itemgetter(1)))
print('Dictionary in ascending order by value : ',sorted_d)
sorted_d = dict(sorted(d.items(), key=operator.itemgetter(1),reverse=True))
print('Dictionary in descending order by value : ',sorted_d)
```

Output:

```
Original dictionary : {1: 2, 3: 4, 4: 3, 2: 1, 0: 0}
Dictionary in ascending order by value : {0: 0, 2: 1, 1: 2, 4: 3, 3: 4}
Dictionary in descending order by value : {3: 4, 4: 3, 1: 2, 2: 1, 0: 0}
```

Program 1.45: Write a Python script to add a key to a dictionary.

Sample Dictionary : {0: 10, 1: 20} Expected Result : {0: 10, 1: 20, 2: 30}.

```
d = {0:10, 1:20}
print(d)
d.update({2:30})
print(d)
```

Write a Python script to concatenate following dictionaries to create a new one.

Sample Dictionary:

```
dic1={1:10, 2:20}
dic2={3:30, 4:40}
dic3={5:50,6:60}
Expected Result: {1: 10, 2: 20, 3: 30, 4: 40, 5: 50, 6: 60}
dic1={1:10, 2:20}
dic2={3:30, 4:40}
dic3={5:50,6:60}
dic4 = {}
for d in (dic1, dic2, dic3): dic4.update(d)
print(dic4)
```

Program 1.46: Write a Python script to check if a given key already exists in a dictionary

```
d = {1: 10, 2: 20, 3: 30, 4: 40, 5: 50, 6: 60}
def is_key_present(x):
 if x in d:
 print('Key is present in the dictionary')
 else:
 print('Key is not present in the dictionary')
is_key_present(5)
is_key_present(9)
```

Output:

```
Key is present in the dictionary
Key is not present in the dictionary
```

Program 1.47: Write a Python program to iterate over dictionaries using for loops.

```
d = {'x': 10, 'y': 20, 'z': 30}
for dict_key, dict_value in d.items():
 print(dict_key,'->',dict_value)
```

Output:

```
y -> 20
z -> 30
x -> 10
```

Program 1.48: Write a Python script to generate and print a dictionary that contains a number (between 1 and n) in the form (x, x*x). Sample Dictionary (n = 5) : Expected Output : {1: 1, 2: 4, 3: 9, 4: 16, 5: 25}

```
n=int(input("Input a number "))
d = dict()

for x in range(1,n+1):
 d[x]=x*x

print(d)
```

Output:

```
10
{1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36, 7: 49, 8: 64, 9: 81, 10: 100}
```

Program 1.49: Write a Python script to print a dictionary where the keys are numbers between 1 and 15 (both included) and the values are square of keys. d=dict()

```
for x in range(1,16):
 d[x]=x**2
print(d)
```

Output:

```
{1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36, 7: 49, 8: 64, 9: 81, 10: 100, 11:
121, 12: 144, 13: 169, 14: 196, 15: 225}
```


Program 1.50: Write a Python program to combine two dictionary adding values for common keys.

```
d1 = {'a': 100, 'b': 200, 'c':300}
d2 = {'a': 300, 'b': 200, 'd':400}
Sample output: Counter({'a': 400, 'b': 400, 'd': 400, 'c': 300})
from collections import Counter
d1 = {'a': 100, 'b': 200, 'c':300}
d2 = {'a': 300, 'b': 200, 'd':400}
d = Counter(d1) + Counter(d2)
print(d)
```

Program 1.51: Write a Python program to create and display all combinations of letters, selecting each letter from a different key in a dictionary. Sample data : {'1':['a','b'], '2':['c','d']} Expected Output:

```
ac
ad
bc
bd
import itertools
d ={'1':['a','b'], '2':['c','d']}
for combo in itertools.product(*[d[k] for k in sorted(d.keys())]):
 print(''.join(combo))
```

Program 1.52: Write a Python program to create a dictionary from two lists without losing duplicate values.

```
Sample lists: ['Class-V', 'Class-VI', 'Class-VII', 'Class-VIII'], [1, 2, 2, 3]
Expected Output: defaultdict(<class 'set'>, {'Class-VII': {2}, 'Class-VI':
{2}, 'Class-VIII': {3}, 'Class-V': {1}})
from collections import defaultdict
class_list = ['Class-V', 'Class-VI', 'Class-VII', 'Class-VIII']
id_list = [1, 2, 2, 3]
temp = defaultdict(set)
for c, i in zip(class_list, id_list):
 temp[c].add(i)
print(temp)
```

Program 1.53: Write a Python program to match key values in two dictionaries. Sample dictionary: {'key1': 1, 'key2': 3, 'key3': 2}, {'key1': 1, 'key2': 2} Expected output: key1: 1 is present in both x and y.

```
x = {'key1': 1, 'key2': 3, 'key3': 2}
y = {'key1': 1, 'key2': 2}
for (key, value) in set(x.items()) & set(y.items()):
 print('%s: %s is present in both x and y' % (key, value))
```

1.10 FUNCTIONS-DEFINING A FUNCTION, CALLING A FUNCTION, TYPES OF FUNCTION, FUNCTION ARGUMENTS, ANONYMOUS FUNCTION, GLOBAL AND LOCAL VARIABLE, EXAMPLES

- A function is defined as a block of organized, reusable code used to perform a single, related action.
- To provide a high degree of modularity Functions are used.

There are two types of Functions:

(a) **Built-in Functions:** Functions that are predefined and organized into a library. We have used many predefined functions in Python.

(b) **User-Defined:** Functions that are created by the programmer to meet the requirements.

Advantages of Functions:

1. To makes the code easier to manage, debug, and scale, functions are used.
2. Every time you need to execute a sequence of statements, all we need to do is to call the function, so that we can reuse the code.
3. Functions allow us to change functionality easily, and different programmers can work on different functions.

1.10.1 Defining a Function

- Following are the rules to define a function in Python:
  - Function blocks begin with the keyword **def** followed by the function name and parentheses ( ).
  - The input parameters or arguments should be placed within these parentheses.
  - The first statement of a function can be an optional statement - the documentation string of the function or *docstring* or documentation string. It is used to explain in brief, what a function does.
  - The code block within every function starts with a colon (:) and is indented.
  - The return statement is used to exit a function. A return statement with no arguments is the same as return None.

Syntax:

```
def function_name( param1, param2,.. ):
 """function_docstring"""
function_body
return [expression]
```


- By default, parameters have a positional behavior and you need to inform them in the same order that they were defined. Every function must be defined before it is called.
- How function works in Python:

Fig. 1.23

Example:

```
# function without parameters
def my_function():
 print("Hello from a function")
# A string as input parameter and prints it on standard screen.
def fun( str ):
 "This prints a passed string into this function"
 print str
 return
```

1.10.2 Calling a Function

- Defining a function only gives it a name, specifies the parameters that are to be included in the function and structures the blocks of code.
- Once the basic structure of a function is finalized, you can execute it by calling it from another function or directly from the Python prompt.
- To call a function means that you are telling the program to execute the function. If there is a return value defined, the function would return the value, else the function would return None.
- To call a function, use the function name followed by parenthesis, if you need to pass parameters/arguments to the function, you write them inside the parentheses.

Syntax: `function_name(arg1, arg2)`

Example:

1. **Call a function that performs a task and has no return value.**

```
def my_function():
   print("Hello from a function")
my_function()
```

Output: 'Hello from a function'

2. **Call a function with a return value.**

```
def my_function(name):
print("Hello" + name)
my_function("Om")
my_function("Shanti")
my_function("Ram")
```

Output:

```
Hello Om
Hello Shanti
Hello Ram
```

3. **To call a function with arguments.**

```
def avg_number(x, y):
    print("Average of ",x," and ",y, " is ",(x+y)/2)
avg_number(3, 4)
```

Output: Average of 3 and 4 is 3.5

Kinds of Functions:

- Looking from a particular point of view, we can discern three kinds of functions – functions that are always available for usage, functions that are contained within external modules, which must be imported and functions defined by a programmer with def keyword.

For example:

(a)
```
>>>from math import sqrt
>>> print sqrt(81)
9.0
```

The sqrt() function is imported from the math module.

(b)
```
>>> def cube(x);
    return x * x * x
>>> print cube(9)
729
```

The cube() function is a custom defined function.

(c)
```
>>> print abs(- 1)
1
```

The abs() function is a built-in function readily accessible. It is part of the core of the language.

Program 1.54: Python Program to Find HCF or GCD.

```
# define a function
def calhcf(x, y):
# choose the smaller number
 if x > y:
 smaller = y
 else:
 smaller = x
 for i in range(1, smaller+1):
 if((x % i == 0) and (y % i == 0)):
 hcf = i
 return hcf
num1 = 54
num2 = 24
# take input from the user
# num1 = int(input("Enter first number: "))
# num2 = int(input("Enter second number: "))
print("The H.C.F. of", num1,"and", num2,"is", calhcf(num1, num2))
```

Output:

```
The H.C.F. of 54 and 24 is 6
```

Program 1.55: To Find the Largest Among Three Numbers.

```
x = 12
y = 17
z = 9
# uncomment following lines to take three numbers from user
#x = float(input("Enter first number: "))
#y = float(input("Enter second number: "))
#z = float(input("Enter third number: "))
if (x >= y) and (x >= z):
 largest = x
elif (y >= x) and (y >= z):
 largest = y
else:
 largest = z
print("The largest number between",x,",",y,"and",z,"is",largest)
```

Output:

```
The largest number between 12, 17 and 9 is 17.
```

1.10.3 Types of Function

- There are three types of functions in Python:
  - Built-in functions, such as min() and max() to get the minimum and maximum values, print() to print an object to the terminal.
  - User-Defined Functions (UDFs), which are functions that users will write their own.
  - Anonymous functions, which are also called lambda functions because they are not declared with the standard def keyword.

1.10.4 Function Arguments

- Function **arguments** are the real **values** passed to (and received by) the function.
- The arguments in the function definition bind the arguments passed at function invocation (i.e. when the function is called), which are called actual parameters, to the names given when the function is defined, which are called formal parameters.
- **Actual arguments**, or simply "arguments," are the values passed to functions to be operated on. **Formal parameters**, or simply "parameters," are the "placeholder" names for the arguments passed.

1. Formal Arguments:

- A function can be called by using the following types of formal arguments:
  - Keyword arguments
  - Default arguments
  - Required arguments/ Positional argument
  - Variable-length arguments
- Let us see details of each argument.

2. Keyword Arguments:

- Keyword arguments are related to the function calls. When you use keyword arguments in a function call, the caller identifies the arguments by the parameter name. The Python interpreter is able to use the keywords provided to match the values with parameters.
- The order of parameters does not matter.

Example:

```
# Function definition is here
def func( str ):
 "This prints a string passed into this function"
 print str
 return;
# calling func function
func( str = "TYBBA")
```

Output:

```
TYBBA
```


Example:

```
# Function definition is here
def mydata( name, age ):
 "This prints a passed info into this function"
 print "Name: ", name
 print "Age: ", age
 return;
# calling the function
mydata( age=50, name="manisha" )
```

Output:

```
Name: manisha
Age: 50
```

3. Default arguments:

- A default argument is an argument that assumes a default value if a value is not provided in the function call for that argument.

Example:

```
# Function definition is here
def mydata( name, age = 40 ):
 "This prints a passed info into this function"
 print "Name: ", name
 print "Age: ", age
 return;
# calling mydata function
mydata( age=50, name="manisha" )
mydata( name="manisha" )
```

Output:

```
Name:  manisha
Age:   50
Name:  manisha
Age:   40
```

Example:

```
# function to calculate the square of the sum of two numbers
def nsquare(x, y = 2):
 return (x*x + 2*x*y + y*y)
print("The square of the sum of 2 and 2 is: ", nsquare(2))
print("The square of the sum of 2 and 3 is: ", nsquare(2,4))
```

Output:

```
The square of the sum of 2 and 2 is: 16
The square of the sum of 2 and 4 is: 36
```

Required Arguments:

- Required arguments are the arguments passed to a function in correct positional order.
- The number of arguments in the function call should match exactly with the function definition.

```
# Function definition is here
def printme( str ):
 "This prints a passed string into this function"
 print str
 return;
# calling function
printme()
```

Output:

```
Traceback (most recent call last):
 File "test.py", line 11, in <module>
 printme();
TypeError: printme() takes exactly 1 argument (0 given)
```

Variable-length Arguments:

- The function can have more arguments than the arguments defining the function.
- These arguments are called *variable-length* arguments and are not named in the function definition, unlike required and default arguments.

Syntax:

```
def functionname([formal_args,] *var_args_tuple ):
 "function_docstring"
 function_suite
 return [expression]
```

- An asterisk (*) is placed before the variable name that holds the values of all non-keyword variable arguments. This tuple remains empty if no additional arguments are specified during the function call.

Example:

```
# Function definition is here
def printinfo( arg1, *vartuple ):
 "This prints a variable passed arguments"
 print "Output is: "
 print arg1
 for var in vartuple:
 print var
 return;
```



```
# call printinfo function
printinfo( 10 )
printinfo( 70, 60, 50 )
```

Output:

```
Output is:
10
Output is:
70
60
50
```

return Statement:

- The statement return [expression] exits a function, optionally passing back an expression to the caller. A return statement with no arguments is the same as return None.
- def function_name(argument1, argument2, ...):

```
statement_1
statement_2
....
return expression
function_name(arg1, arg2)
```

Example: The following function returns the square of the sum of two numbers.

```
def nsquare(x, y):
 return (x*x + 2*x*y + y*y)
print("The square of the sum of 2 and 3 is: ", nsquare(2, 3))
```

Output:

```
The square of the sum of 2 and 3 is: 25
```

1.10.5 Anonymous Function

- Anonymous Functions are the functions that are not bond to name. It means anonymous function does not have a name.
- While normal functions are defined using the def keyword, in Python anonymous functions are defined using the lambda keyword. Hence, anonymous functions are also called lambda functions.
- A lambda function can take any number of arguments, but can only have one expression. Lambda is created without using the def keyword.

Syntax of a lambda expression in Python:

```
lambda [arg1,arg2,..]:[expression]
```

Where you place lambda expression in Python, it returns the value of the expression.

Example:

```
#Function Definition
square=lambda x1: x1*x1
#Calling square as a function
print "Square of number is",square(10)
```

Output:

```
>>>
Square of number is 100
>>>
```

Example: Program To Display Powers of 2 Using Anonymous Function.

```
terms = int(input("How many terms? "))
# use anonymous function
result = list(map(lambda x: 2 ** x, range(terms)))
# display the result
print("The total terms is:",terms)
for i in range(terms):
 print("2 raised to power",i,"is",result[i])
```

Output:

```
The total terms is: 10
2 raised to power 0 is 1
2 raised to power 1 is 2
2 raised to power 2 is 4
2 raised to power 3 is 8
2 raised to power 4 is 16
2 raised to power 5 is 32
2 raised to power 6 is 64
2 raised to power 7 is 128
2 raised to power 8 is 256
2 raised to power 9 is 512
```

1.10.6 Global and Local Variable, Examples

- A variable's scope tells us where in the program it is visible. A variable may have local or global scope. A variable that's declared inside a function has a local scope. When you declare a variable outside python functions, or anything else, it has global scope.
- A variable's lifetime is the period of time for which it resides in the memory. A variable that's declared inside python functions is destroyed after the function stops executing. So the next time the function is called, it does not remember the previous value of that variable.

- All variables in a program may not be accessible at all locations in that program. This depends on where you have declared a variable.
- There are two basic scopes of variables in Python:
  - **Local variables:** Variables declared inside a function body is known as Local Variable.
  - **Global variables:** Variable defined outside the function is called Global Variable.

Global Vs. Local Variables:

- Variables that are defined inside a function body have a local scope, and those defined outside have a global scope.
- The local variables can be accessed only inside the function in which they are declared, whereas global variables can be accessed throughout the program body by all functions.
- When you call a function, the variables declared inside it are brought into scope.

Program 1.56:

```
sum = 0; # This is global variable.
# Function definition
def sumfunc( arg1, arg2 ):
# add both the parameters and return them."
sum = arg1 + arg2; # Here sum is local variable.
print "Inside the function local sum: ", sum
return sum;
# calling sumfunc function
sumfunc( 100, 200 );
print "Outside the function global sum: ", sum
```

Output:

```
Inside the function local sum: 300
Outside the function global sum: 0
```

Nonlocal Variables:

- Nonlocal variable are used in nested function whose local scope is not defined. This means, the variable can be neither in the local nor the global scope.
- In the following code , we use nonlocal keyword to create nonlocal variable.

```
def outer():
 x = "local"
 def inner():
 nonlocal x
 x = "nonlocal"
 print("inner:", x)
 inner()
 print("outer:", x)
outer()
```

- When we run the code, the will output be:

```
inner: nonlocal
outer: nonlocal
```

- In the above code, there is a nested function inner(). We use nonlocal keyword to create nonlocal variable. The inner() function is defined in the scope of another function outer().

Program 1.57: To Check Prime Number.

```
num = int(input("Enter a number: "))
# prime numbers are greater than 1
if num > 1:
 # check for factors
 for i in range(2,num):
 if (num % i) == 0:
 print(num,"is not a prime number")
 break
 else:
 print(num,"is a prime number")
# if input number is less than or equal to 1, it is not prime
else:
 print(num,"is not a prime number")
```

Output:

```
Enter a number: 56
56 is not a prime number
```

Program 1.58: To Find Factors of Number.

```
# define a function
def factors(x):
 print("The factors of",x,"are:")
 for i in range(1, x + 1):
 if x % i == 0:
 print(i)
num = int(input("Enter a number: "))
print_factors(num)
```

Output:

```
Enter a number: 50
The factors of 50 are:
1
2
5
25
50
```


Program 1.59: To Find the Factorial of a Number.

```
# change the value for a different result
num = int(input("Enter a number: "))
factorial = 1
# check if the number is negative, positive or zero
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)
```

Output:

```
Enter a number
5
The factorial of 5 is 120
```

Program 1.60: To Find the Sum of Natural Numbers.

```
num = 10
# uncomment to take input from the user
#num = int(input("Enter a number: "))
if num < 0:
 print("Enter a positive number")
else:
 sum = 0
 # use while loop to iterate un till zero
 while(num > 0):
 sum += num
 num -= 1
 print("The sum is",sum)
```

Output:

```
The sum is 55
```

Program 1.61: Write a Python Program to Calculate the Average of Numbers in a Given List.

```
ele=[2,4,6,7,8]
def avg(ele):
 return sum(ele)/len(ele)
print("average : ",round(avg(ele),1))
```

Output:

```
average : 5.4
```

Program 1.62: Write a recursive function which print string in reverse order.

```
def rec(s):
    if (len(s)==0):
        return s
    else:
        return rec(s[1:])+s[0]

s=input("enter the string:=")
if(s==""):
    print("String should not be null")
else:
    rev=rec(s)
    print(rev)
```

Output:

```
enter the string:=Python programming
gnimmargorp nohtyP
```

Program 1.63: Write an anonymous function to find area of circle.

```
rad=lambda a:3.14*a*a
a=int(input("enter radius of circle to calculate area="))
print(rad(a))
```

Output:

```
enter radius of circle to calculate area=15
706.5
```

Program 1.64: Write a function which prints a dictionary where the keys are numbers between 1 and 20 (both included) and the values are square of keys.

```
d={}
def createdict(d):
    for i in range(1,21):
        d[i]=i**2
    return d

print ("Dictionary is:",createdict(d))
```

Output

```
Dictionary is: {1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36, 7: 49, 8: 64, 9: 81,
10: 100, 11: 121, 12: 144, 13: 169, 14: 196, 15: 225, 16. 256, 17: 289
324, 19: 361, 20: 400}
```


Program 1.65: Write a program to find gcd of number use recursion.

```
def getgcd(a,b):
    if(b==0):
        return a
    else:
        return getgcd(b,a%b)

a=int(input("enter value for 1 st number"))
b=int(input("enter value for 2 nd number"))
gcd=getgcd(a,b)
print("GCD is:=",gcd)
```

Output:

```
enter value for 1 st number35
enter value for 2 nd number56
GCD is:= 7
```

Additional Programs

Program 1.66: Program to find the square root of a number.

```
x=int(input("Enter an integer number:"))
ans=x**0.5
print("Square root= ", ans)
```

Output:

```
Enter an integer number: 144
Square root= 12.0
```

Program 1.67: Program to find the area of Rectangle.

```
l=float(input("Enter length of the rectangle: "))
b=float(input("Enter breadth of the rectangle: "))
area=l*b
print("Area of Rectangle= ",area)
```

Output:

```
Enter length of the rectangle: 5
Enter breadth of the rectangle: 6
Area of Rectangle= 30.0
```

Program 1.68: Write a program to display following pattern.

Code:

```
1 2 3 4
1 2 3
1 2
1
a=[1,2,3,4]
for i in range(4+1):
 for j in range(4-i):
 print(a[j],end=" ")
 print(" ")
```

Program 1.69: Program to calculate surface volume and area of a cylinder.

```
pi=22/7
height = float(input('Height of cylinder: '))
radian = float(input('Radius of cylinder: '))
volume = pi * radian * radian * height
sur_area = ((2*pi*radian) * height) + ((pi*radian**2)*2)
   print("Volume is: ", volume)
   print("Surface Area is: ", sur_area)
```

Output:

```
Height of cylinder: 4
Radius of cylinder: 6
Volume is: 452.57142857142856
Surface Area is: 377.1428571428571
```

Program 1.70: Program to swap the value of two variables.

```
num1=input("Enter first value: ")
num2=input("Enter second value: ")
   print("Numbers before swapping")
   print("num1= ",num1)
   print("num2= ",num2)
temp=num1
num1=num2
num2=temp
   print("Numbers after swapping")
   print("num1= ",num1)
   print("num2= ",num2)
```

Output:

```
Enter first value: 10
Enter second value: 20
Numbers before swapping
num1= 10
num2= 20
Numbers after swapping
num1= 20
num2= 10
```


Program 1.71: Write a program which finds sum of digits of a number.

```
n=int(input("Enter a number:"))
tot=0
while(n>0):
 dig=n%10
 tot=tot+dig
 n=n//10
print("The total sum of digits is:",tot)
```

Output:

```
Enter a number:156
The total sum of digits is: 12
```

Program 1.72: Write a program which prints Fibonacci series of a number.

```
nterms = 10
# uncomment to take input from the user
#nterms = int(input("How many terms? "))
# first two terms
n1 = 0
n2 = 1
count = 0
# check if the number of terms is valid
if nterms <= 0:
 print("Please enter a positive integer")
elif nterms == 1:
 print("Fibonacci sequence upto",nterms,":")
 print(n1)
else:
 print("Fibonacci sequence upto",nterms,":")
 while count < nterms:
 print(n1,end=' , ')
 nth = n1 + n2
 # update values
 n1 = n2
 n2 = nth
 count += 1
```

Output:

```
Fibonacci sequence upto 10:
0 , 1 , 1 , 2 , 3 , 5 , 8 , 13 , 21 , 34 ,
```

Program 1.73: Write a program which accept an integer value 'n' and display all prime numbers till 'n'.

```
# Python program to ask the user for a range and display all the prime
  numbers in that interval
  # uncomment to take input from the user
  # lower = int(input("Enter lower range: "))
  # upper = int(input("Enter upper range: "))
  lower = 10
  upper = 20

  for num in range(lower,upper + 1):
   # prime numbers are greater than 1
   if num > 1:
   for i in range(2,num):
   if (num % i) == 0:
   break
   else:
   print(num)
```

Output:

```
11
13
17
19
```

Program 1.74: Write a program to display following pattern.

```
*****
 ****
 ***
 **
 *
n=int(input("Enter number of rows: "))
for i in range (n,0,-1):
 print((n-i) * ' ' + i * '*')
```

Output:

```
Enter number of rows: 5
*****
 ****
 ***
 **
 *
```


Program 1.75: Write a program to reverse a given number.

Code:

```
n=int(input("enter any number"))
sum=0
while(n!=0):
 r=n%10
 sum=(sum*10)+r
 n=n//10
print("reverse number :",sum)
```

Output:

```
enter any number21
reverse number : 12
```

Summary

- Python is an open source, object-oriented, high-level powerful programming language.
- A data type is a set of values, and a set of operators that may be applied to those values. Python supports five basic numerical types namely Numbers, String, List, Tuple and Dictionary.
- String in Python are surrounded by either single quotation marks, or double quotation marks such as 'hello' is the same as "hello".
- String is a sequence of characters. Strings in Python are arrays of bytes representing unicode characters.
- Variable is a name which is used to refer memory location. Variable also known as identifier and used to hold value. The period of time that a variable exists is called its lifetime.
- Python variables do not need explicit declaration to reserve memory space. The declaration happens automatically when you assign a value to a variable. The equal sign (=) is used to assign values to variables.
- A local variable is a variable that is only accessible from within a given function. Such variables are said to have local scope. In Python, any variable assigned a value in a function becomes a local variable of the function.
- A global variable is a variable that is defined outside of any function definition. Such variables are said to have global scope.
- Tuple is another data type which is a sequence of data similar to list.
- Python Dictionary is an unordered sequence of data of key-value pair form.
- Python operators can be classified into several categories such as arithmetic, logical, comparison, bitwise and assignment.

- A control statement is a statement that determines the control flow of a set of instructions. A control structure is a set of instructions and the control statements controlling their execution. Three fundamental forms of control in programming are sequential, selection, and iterative control.
- Python list is enclosed between square ([]) brackets and elements are stored in the index basis with starting index 0.
- A list is a container which holds comma-separated values (items or elements) between square brackets where items or elements need not all have the same type.
- A **tuple** in Python is an *immutable* linear data structure, denoted by a comma-separated list of elements within parentheses, allowing mixed-type elements.
- A tuple is a sequence of immutable objects, therefore tuple cannot be changed. It can be used to collect different types of object.
- Dictionary is an unordered set of key and value pair. It is a container that contains data, enclosed within curly braces. The pair i.e., key and value is known as item.
- Functions are a construct to structure programs. They are known in most programming languages, sometimes also called subroutines or procedures. Functions are used to utilize code in more than one place in a program.
- Function in Python is defined by the "def" statement followed by the function name and parentheses (()).
- Inside the function, the arguments are assigned to variables called parameters.

Check Your Understanding

1. Which of the following is incorrect variable name in Python?
   (a) variable_1 (b) variable1
   (c) 1variable (d) _variable
2. Which statement is correct?
   (a) List is immutable && Tuple is mutable
   (b) List is mutable && Tuple is immutable
   (c) Both are Mutable.
   (d) Both are Immutable
3. What type of data is: arr = [(1,1),(2,2),(3,3)]?
   (a) Array of tuples (b) Tuples of lists
   (c) List of tuples (d) Invalid type
4. Assignment Operators are used to.....
   (a) to assign values to variables (b) to compare values
   (c) to sort values (d) all of the above
5. 'in' operator is .....
   (a) Identity operator (b) Membership operator
   (c) Arithmetic operator (d) Assignment operator

6. A while loop in Python is used for what type of iteration?
   (a) indefinite (b) discriminate
   (c) definite (d) indeterminate
7. In a string range of slice shown by character _____
   (a) [--] (b) [:]
   (c) [--] (d) **
8. Tuples are shown by ___
   (a) { } (b) [ ]
   (c) ( ) (d) < >
9. A dictionary consists of a collection of -------- pairs
   (a) Key-value (b) Value-key
   (c) Key: Value (d) value: Key
10. Which brackets are used to access dictionary elements?
    (a) [] (b) {}
    (c) () (d) <>
11. Which of the following keywords marks the beginning of function block?
    (a) fun (b) define
    (c) def (d) function

Answers

1. (c)	2. (b)	3. (c)	4. (a)	5. (b)	6. (a)	7. (b)	8. (c)	9. (c)	10. (a)
11. (c)									

Practice Questions

Q.I Answer the following Questions in short:

1. Which are data types in Python?
2. What are the types of variables?
3. Which are the different conditional statements?
4. What is looping?
5. What is operator?
6. List out python basic operators.
7. List out loop control statements.
8. Which are special operators in Python?
9. What are the different types of function?
10. How to call function?
11. What is string slice?
12. How to select an element from List?
13. What is the difference between Xrange and range?

14. How to delete tuple?
15. Explain any two tuple operations with an example.
16. List out different dictionary function.

Q.II Answer the following Questions:

1. Explain Loop control statements used in Python?
2. Write a python program to display fibonnacci series.
3. Write python program to check whether number is even or odd.
4. Write a python program to display Prime numbers between given range.
5. Write program to find factorial of given number.
6. Write a program which finds sum of digits of a number.
7. Explain any five built-in List functions.
8. Which are special operations in string?
9. How to create and access elements in list.
10. Which are basic tuple operations? Explain with example.
11. How to access values in tuples?
12. What is the use of + and * operators on tuples?
13. Write a Python program to add an item in a tuple.
14. Write a Python program to check whether an element exists within a tuple.
15. What are built-in dictionary functions? Explain.
16. How to access dictionary elements?
17. Explain how to delete elements in Dictionary.
18. Explain about functions with suitable examples.
19. Write about the concept of scope of a variable in a function.
20. Write in brief about anonymous functions.
21. Write a program to calculate simple interest.
22. Write program to find area of circle.
23. Write program to swap numbers.
24. Write short note on identifier and keyword.
25. Explain different data types in python.
26. What are the features of Python?
27. Explain any three built-in dictionary functions.

Q.III Define the terms:

1. Data type
2. Tuple
3. List
4. Variable
5. Slicing Dictionaries
6. functions
7. anonymous function
8. global variable
9. local variable

■■■

2... Modules and Packages

Learning Objectives ...

- To learn Concepts of Files.
- To study Modules.
- To understand Packages.

2.1 INTRODUCTION

- In Python, packages allow us to create a hierarchical file directory structure of modules. For example, mymodule.mod1 stands for a module mod1, in the package my module.
- A Python package is a collection of modules which have a common purpose. In short, modules are grouped together to forms packages.
- A file is a collection of related data that acts as a container of storage as data permanently. The file processing refers to a process in which a program processes and accesses data stored in files.
- A file is a computer resource used for recording data in a computer storage device. The processing on a file is performed using read/write operations performed by programs.
- Python supports file handling and allows users to handle files i.e., to read and write files, along with many other file handling options, to operate on files.
- Python programming provides modules with functions that enable us to manipulate text files and binary files. Python allows us to create files, update their contents and also delete files.
- Files are named locations on disk to store related information. They are used to permanently store data in a non-volatile memory (e.g. hard disk).
- A text file is a file that stores information in the term of a sequence of characters (textual information), whiles a binary file stores data in the form of bits (0s and 1s) and used to store information in the form of text, images, audios, videos etc.
- The file handling plays an important role when the data needs to be stored permanently into the file. A file is a named location on disk to store related information.

- We can access the stored information (non-volatile) after the program termination. Python has several functions for creating, reading, updating, and deleting files.
- When an error occurs, or exception as we call it, Python will normally stop and generate an error message. Exception in Python is nothing but errors which are encountered at the run time.
- Exception Handling is the mechanism it is allow us to handle errors very smartly while the program is running.

2.2 MODULES

- Modules are primarily the (.py) files which contain Python programming code defining functions, class, variables, etc. with a suffix .py appended in its file name. A file containing .py python code is called a module.
- If we want to write a longer program, we can use file where we can do editing, correction. This is known as creating a script. As the program gets longer, we may want to split it into several files for easier maintenance.
- We may also want to use a function that we have written in several programs without copying its definition into each program.
- In Python we can put definitions in a file and use them in a script or in an interactive instance of the interpreter. Such a file is called a module.
- Modules and packages are constructs in Python programming that promote code modularization.
- The modularization (modular programming) refers to the process of breaking a large programming task into separate, smaller, more manageable subtasks or modules.
- A module in Python programming allows us to logically organize the python code. A module is a single source code file. The module in Python have the .py file extension. The name of the module will be the name of the file.
- A python module can be defined as a python program file which contains a python code including python functions, class, or variables. In other words, we can say that our Python code file saved with the extension (.py) is treated as the module.

2.2.1 Creating Module

- Writing a module means simply creating a file which can contains python definitions and statements. The file name is the module name with the extension .py. To include module in a file, use import statement.
- Follow the following steps to create modules:
  1. Create a first file as a python program with extension as .py. This is your module file where we can write a function which performs some task.
  2. Create a second file in the same directory called main file where we can import the module to the top of the file and call the function.
- Second file needs to be in the same directory so that Python knows where to find the module since it's not a built-in module.

Program 2.1: For creating a module. Type the following code and save it as p1.py.

```
def add(a, b):
 "This function adds two numbers and return the result"
   result = a + b
   return result
def sub(a, b):
 "This function subtract two numbers and return the result"
   result = a - b
   return result
def mul(a, b):
 "This function multiply two numbers and return the result"
   result = a * b
   return result
def div(a, b):
 "This function divide two numbers and return the result"
   result = a / b
   return result
```

Import the definitions inside a module:

```
import p1
print("Addition=" , p1.add(10,20))
print("Subtraction=" ,p1.sub(10,20))
print("Multiplication=" ,p1.mul(10,20))
print("division=" ,p1.div(10,20))
```

Output:

```
Addition= 30
Subtraction= -10
Multiplication= 200
division= 0.5
```

2.2.2 Importing Modules in Python Program

- The import statement is used to imports a specific module by using its name.
- The import statement creates a reference to that module in the current namespace. After using import statement we can refer the things defined in that module.
- We can import the definitions inside a module to another module or the interactive interpreter in Python. We use the import keyword to do this.
- Create second file. Let p2.py in same directory where p1.py is created. Write following code in p2.py.
- **Import the definitions inside a module:**

```
import p1
print(p1.add(10,20))
print(p1.sub(20,10))
```

Output:

```
30
10
```

- **Import the definitions using the interactive interpreter:**

```
>>> import p1
>>> p1.add(10,20)
30
>>> p1.sub(20,10)
10
>>>
```

Importing Objects From Module:

- Import statement in python is similar to #include header_file in C/C++. Python modules can get access to code from another module by importing the file/function using import.
- Python provides three different ways to import modules.

1. From x import a:

- Imports the module x, and creates references in the current namespace to all public objects defined by that module. If we run this statement, we can simply use a plain name to refer to things defined in module x.
- We can access attribute / mehods directly without dot notation.

Example 1: Import inside a module (from x import a).

```
from p1 import add
print("Addition=" , add(10,20))
```

Output:

```
Addition= 30
```

Example 2: For import on interactive interpreter.

```
>>> from math import pi
>>> pi
3.141592653589793
>>> from math import sqrt
>>> sqrt(144)
12.0
```

2. From x import a, b, c:

- Imports the module x and creates references in the current namespace to the given objects. Or we can use a, b and c function in our program.

Example 1: Import inside a module (from x import a, b, c).

```
from p1 import add, sub
print("Addition=" , add(10,20))
print("Subtraction=" ,sub(10,20))
```

Output:

```
Addition= 30
Subtraction= -10
```


Example 2: For import on interactive interpreter.

```
>>> from math import sqrt, ceil, floor
>>> sqrt(144)
12.0
>>> ceil(2.6)
3
>>> floor(2.6)
2
```

3. From x import *:

- We can use * (asterisk) operator to import everything from the module.

Example 1: Import inside a module (from x import *).

```
from p1 import *
print("Addition=" , add(10,20))
print("Subtraction=" ,sub(10,20))
print("Multiplication=" ,mul(10,20))
print("division=" ,div(10,20))
```

Output:

```
Addition= 30
Subtraction= -10
Multiplication= 200
division= 0.5
```

Example 2: For import on interactive interpreter.

```
>>> from math import *
>>> cos(60)
-0.9524129804151563
>>> sin(60)
-0.3048106211022167
>>> tan(60)
0.320040389379563
```

2.2.3 Built in Modules

- A module is a collection of Python objects such as functions, classes, and so on. Python interpreter is bundled with a standard library consisting of large number of built-in modules.
- Built-in modules are generally written in C and bundled with Python interpreter in precompiled form. A built-in module may be a Python script (with .py extension) containing useful utilities.
- A module may contain one or more functions, classes, variables, constants, or any other Python resources.
- This module provides numeric and math-related functions and data types.
- The numbers module defines an abstract hierarchy of numeric types. The math and cmath modules contain various mathematical functions for floating-point and complex numbers. The decimal module supports exact representations of decimal numbers, using arbitrary precision arithmetic.

1. **math and cmath Modules:**

- Python provides two mathematical modules namely math and cmath. The math module gives us access to hyperbolic, trigonometric, and logarithmic functions for real numbers and cmath module allows us to work with mathematical functions for complex numbers.

Example 1: For math module.

```
>>> import math
>>> math.ceil(1.001)
2
>>> from math import *
>>> ceil(1.001)
2
>>> floor(1.001)
1
>>> factorial(5)
120
>>> trunc(1.115)
1
>>> sin(90)
0.8939966636005579
>>> cos(60)
-0.9524129804151563
>>> exp(5)
148.4131591025766
>>> log(16)
2.772588722239781
>>> log(16,2)
4.0
>>> log(16,10)
1.2041199826559246
>>> pow(144,0.5)
12.0
>>> sqrt(144)
12.0
>>>
```

- The mathematical functions for complex numbers.

Example 2: For cmath module.

```
>>> from cmath import *
>>> c=2+2j
>>> exp(c)
(-3.07493232863935946.718849697428825j)
```



```
>>> log(c,2)
(1.5000000000000002+1.1330900354567985j)
>>> sqrt(c)
(1.5537739740300374+0.6435942529055826j)
```

2. **Decimal Module:**

- Decimal numbers are just the floating-point numbers with fixed decimal points. We can create decimals from integers, strings, floats, or tuples.
- A Decimal instance can represent any number exactly, round up or down, and apply a limit to the number of significant digits.

Example: For decimal module.

```
>>> from decimal import Decimal
>>> Decimal(121)
Decimal('121')
>>> Decimal(0.05)
Decimal('0.05000000000000000277555756156289135105907917022705078125')
>>> Decimal('0.15')
Decimal('0.15')
>>> Decimal('0.012')+Decimal('0.2')
Decimal('0.212')
>>> Decimal(72)/Decimal(7)
Decimal('10.285714285714285714285714429')
>>> Decimal(2).sqrt()
Decimal('1.4142135623730950488016887242')
```

3. **Fractions Module:**

- A fraction is a number which represents a whole number being divided into multiple parts. Python fractions module allows us to manage fractions in our Python programs.

- **Example:** For fractions module.

```
>>> import fractions
>>> for num, decimal in [(3, 2), (2, 5), (30, 4)]:
        fract = fractions.Fraction(num, decimal)
        print(fract)
3/2
2/5
15/2
```

- It is also possible to convert a decimal into a Fractional number. Let's look at a code snippet:

```
>>> import fractions
>>> for deci in ['0.6', '2.5', '2.3', '4e-1']:
        fract = fractions.Fraction(deci)
        print(fract)
```

Output:

```
3/5
5/2
23/10
2/5
>>>
```

4. Statistics Module:

- Statistics module provides access to different statistics functions. Example includes mean (average value), median (middle value), mode (most often value), standard deviation (spread of values).

Example: For statistics module.

```
>>> import statistics
>>> statistics.mean([2,5,6,9])                    # average
5.5
>>> import statistics
>>> statistics.median([1,2,3,8,9])                # central value
3
>>> statistics.median([1,2,3,7,8,9])
5.0
>>> import statistics
>>> statistics.mode([2,5,3,2,8,3,9,4,2,5,6])      # repeated value
2
>>> import statistics
>>> statistics.stdev([1,1.5,2,2.5,3,3.5,4,4.5,5])
1.3693063937629153
```

5. Time Module:

- There is a popular time module available in Python which provides functions for working with times and for converting between representations.
- Python has a module named time to handle time-related tasks. To use functions defined in the module, we need to import the module first like `import time`.
- Python "time" module has following functions:

(i) **time()**: This function returns the current time instant, a floating-point number of seconds since the epoch. Example. The function time() returns the current system time in ticks since 00:00:00 hrs January 1, 1970 (epoch).

```
import time; # This is required to include time module.
ticks = time.time()
  print "Number of ticks since 12:00am, January 1, 1970:", ticks
#Output Number of ticks since 12:00am, January 1, 1970: 7186862.73399
```

(ii) **gmtime(sec)**: This function returns a structure with 9 values each representing a time attribute in sequence. It converts seconds into time attributes(days, years, months etc.) till specified seconds from epoch. If no seconds are mentioned, time is calculated till present.

(iii) **asctime("time")**: This function takes a time attributed string produced by gmtime() and returns a 24 character string denoting time.

(iv) **ctime(sec)**: This function returns a 24 character time string but takes seconds as argument and computes time till mentioned seconds. If no argument is passed, time is calculated till present.

(v) **sleep(sec)**: This method is used to halt the program execution for the time specified in the arguments.

(vi) **tzset()**: Resets the time conversion rules used by the library routines. The environment variable TZ specifies how this is done.

(vii) **clock( )**: Returns the current CPU time as a floating-point number of seconds. To measure computational costs of different approaches, the value of time.clock is more useful than that of time.time().

Datetime Module:

A date in Python is not a data type of its own, but we can import a module named datetime to work with dates as date objects.

Example: Import the datetime module and display the current date:

```
import datetime
x = datetime.datetime.now()
print(x)
```

Output:

```
2021-07-24 12:41:12.701464
```

The date contains year, month, day, hour, minute, second, and microsecond.
The datetime module has many methods to return information about the date object.

Example: Return the year and name of weekday:

```
import datetime
x = datetime.datetime.now()
print(x.year)
print(x.strftime("%A"))
```

Output:

```
2021
Saturday
```

Creating Date Objects:

To create a date, we can use the datetime() class (constructor) of the datetime module.
The datetime() class requires three parameters to create a date: year, month, day.

Example: Create a date object.

```
import datetime
x = datetime.datetime(2020, 5, 17)
print(x)
```

Output:

```
2021-07-24  00:00:00
```

- **Strftime Method of datetime:** It display the month in the string format.

Example:

```
import datetime
x = datetime.datetime(2021, 7, 24)
print(x.strftime("%B"))
```

Output:

```
July
```

- The constants used with datetime are as follows:
  - `class datetime.date`
    An idealized naive date, assuming the current Gregorian calendar always was, and always will be, in effect. Attributes: year, month, and day.
  - `class datetime.time`
    An idealized time, independent of any particular day, assuming that every day has exactly 24*60*60 seconds. Attributes: hour, minute, second, microsecond, and tzinfo.
  - `class datetime.datetime`
    A combination of a date and a time. Attributes: year, month, day, hour, minute, second, microsecond, and tzinfo.
  - `class datetime.timedelta`
    A duration expressing the difference between two date, time, or datetime instances to microsecond resolution.
  - `class datetime.tzinfo`
    An abstract base class for time zone information objects. These are used by the datetime and time classes to provide a customizable notion of time adjustment (for example, to account for time zone and/or daylight saving time).

7. Calendar:

- This module allows you to output Gregorian calendars and provides additional useful functions related to the calendar.
  - `class calendar.Calendar(firstweekday=0)`
- Creates a Calendar object. firstweekday is an integer specifying the first day of the week. 0 is Monday (the default), 6 is Sunday.
- A Calendar object provides several methods that can be used for preparing the calendar data for formatting.

Example:

```
# Python program to display calendar of given month of the year
import calendar
yy = 2017
mm = 11
# display the calendar
print(calendar.month(yy, mm))
```


Output:

```
    November 2017
Mo Tu We Th Fr Sa Su
       1  2  3  4  5
 6  7  8  9 10 11 12
13 14 15 16 17 18 19
20 21 22 23 24 25 26
27 28 29 30
```

Example:

```
# Python code to display the calendar of the given year.
# importing calendar module
import calendar
# prints calendar of 2018
print ("The calendar of year 2021 is: ")
print (calendar.calendar(2021, 2, 1, 6))
```

Output:

8. **sys module:**

- The **sys module** in Python provides various functions and variables that are used to manipulate different parts of the Python runtime environment. It allows operating on the interpreter as it provides access to the variables and functions that interact strongly with the interpreter.

Example:

```
# sys.version is used which returns a string containing the version of
Python Interpreter with some additional information.
    >>> import sys
    >>> print(sys.version)
    3.9.6 (tags/v3.9.6:db3ff76, Jun 28 2021, 15:26:21)
                                        [MSC v.1929 64 bit (AMD64)]
    >>>
```

- The sys modules provide variables for better control over input or output. We can even redirect the input and output to other devices. This can be done using three variables:
  - stdin
  - stdout
  - stderr

1. **stdin:** It can be used to get input from the command line directly. It is used for standard input. It internally calls the input() method. It, also, automatically adds '\n' after each sentence.

   Example:

```
import sys
for line in sys.stdin:
 if 'q' == line.rstrip():
 break
 print(f'Input: {line}')
 print("Exit")
```

   Output:

```
======= RESTART: C:/Users/HP/AppData/Local/Programs/Python/Pyton39/
tre.py ====== tybba
Input : tybba
yes
Input : yes
```

2. **stdout:** A built-in file object that is analogous to the interpreter's standard output stream in Python. stdout is used to display output directly to the screen console.

```
import sys
sys.stdout.write('tybba')
```

   Output:

```
Tybba
```

3. **Stderr:** Whenever an exception occurs in Python it is written to sys.stderr.

2.2.4 Working with Random Modules

Random Module:

- Sometimes, we want the computer to pick a random number in a given range, pick a random element from a list etc.
- The random module provides functions to perform these types of operations. This function is not accessible directly, so we need to import random module and then we need to call this function using random static object.

- **Example:** For random module.

```
>>> import random
>>> print(random.random())# It generate a random number in the range
(0.0, 1.0)
0.27958089234907935
>>> print(random.randint(10,20)) # It generate a random integer between
x and y inclusive 13
>>> # Program to generate a random number between 0 and 9
# import the random module
import random
print(random.randint(0,9))
Python uses library random to generate random numbers.
>>> import random
```

Methods:

1. **randrange:** Generates integer between lower to upper argument. By default lower bound is 0.

   Example:

```
rno=random.randrange(50) # generates random numbers from 0 to 49
rno=random.randrange(10,20) # generates random numbers from 10 to 20
```

2. **shuffle(list):** It is used to shuffle the contents of the list.

   Example:

```
import random
Color=['Cyan', 'Magenta', 'Yellow', 'Black']
random.shuffle(Color)
print "Reshuffled color:", Color
which returns: Reshuffled color : ['Cyan', 'Magenta', 'Yellow', 'Black']
random.shuffle(Color)
print "Reshuffled color:", Color
which returns: Reshuffled color: ['Yellow', 'Black', 'Magenta', 'Cyan']
```

3. **uniform():**

   Syntax: `random.uniform(x,y)`

   where x is lower limit and y is upper limit of random float and returns random floating point number greater than or equal to x and less than y.

Example:

```
import random
print " uniform no(1,100):", random.uniform(1,100)
# uniform no(1,100) 9.38500684602
```

4. **random()**: This generates the random numbers from 0 to 1.

Example:

```
>>> from random import random
>>>  random()          # 0.8407436368337916
>>>  random()          # 0.3069951291690556
>>>  random()          # 0.4887997856049814
>>>  random()          # 0.35343343245685684
```

Each time random generates the different number.

5. **randint()**: random.randint(a,b) here a is lower and b is upper bound.

Example:

```
import random
print random.randint(0,5) #output either 1,2,3,4 or 5
```

6. **choice()**: random.choice(sequence)

Example:

```
import random
MyChoice  = random.choice[('1-Swimming', '2-Badminton', '3-Cricket',
'4-Basketball', '5-Hockey'])
print 'My choice is:', MyChoice #returns: My choice is: 4-Basketball
```

2.3 USER DEFINED FUNCTIONS

- A function is a set of statements that take inputs, do some specific computation and produce output. The idea is to put some commonly or repeatedly done tasks together and make a function so that instead of writing the same code again and again for different inputs, we can call the function.
- Functions that readily comes with Python are called built-in functions. Python provides built-in functions like print(), etc. but we can also create your own functions. These functions are known as **user defines functions**.
- All the functions that are written by any us comes under the category of user defined functions. Below are the steps for writing user defined functions in Python.
  - In Python, def keyword is used to declare user defined functions.
  - An indented block of statements follows the function name and arguments which contains the body of the function.

Syntax:

```
def function_name():
statements
.
.
```


Calling a Function:

- Defining a function only gives it a name, specifies the parameters that are to be included in the function and structures the blocks of code.
- Once the basic structure of a function is finalized, you can execute it by calling it from another function or directly from the Python prompt.
- To call a function means that you are telling the program to execute the function. To call the function, you write the name of the function followed by parentheses. In case you need to pass parameters/arguments to the function, you write them inside the parentheses.
- If there is a return value defined, the function would return the value, else the function would return none.

Syntax: `function_name(arg1, arg2)`

Example:

```
# Declaring a function
def fun():
print("Inside function")

# Calling function
fun()
```

Output:

```
Inside function
```

Example:

```
def avg_number(x, y):
 print("Average of ",x," and ",y, " is ",(x+y)/2)
avg_number(3, 4)
```

Output:

```
Average of 3 and 4 is 3.5
```

2.3.1 Structure of Python Modules

- A module is a file containing Python definitions and statements. A module can define functions, classes, and variables. A module can also include runnable code. Grouping related code into a module makes the code easier to understand and use. It also makes the code logically organized.
- Modules are primarily the (.py) files which contain Python programming code defining functions, class, variables, etc. with a suffix .py appended in its file name. A file containing .py python code is called a module.
- If we want to write a longer program, we can use file where we can do editing, correction. This is known as creating a script. As the program gets longer, we may want to split it into several files for easier maintenance.
- We may also want to use a function that we have written in several programs without copying its definition into each program.
- In Python we can put definitions in a file and use them in a script or in an interactive instance of the interpreter. Such a file is called a module.

2.4 PACKAGES

- Suppose we have developed a very large application that includes many modules. As the number of modules grows, it becomes difficult to keep track of them all as they have similar names or functionality.
- It is necessary to group and organize them by some mean which can be achieved by packages.
- A package is a hierarchical file directory structure that defines a single Python application environment that consists of modules and subpackages and sub-subpackages and so on.
- Packages allow for a hierarchical structuring of the module namespace using dot notation. Packages are a way of structuring many packages and modules which help in a well-organized hierarchy of data set, making the directories and modules easy to access.
- A package is a collection of Python modules, i.e., a package is a directory of Python modules containing an additional `__init__.py` file (For example: Phone/`__init__.py`).

2.4.1 Creating and Importing Packages

- Creating a package is quite easy, since it makes use of the operating system's inherent hierarchical file structure as shown in Fig. 2.1.
- Here, there is a directory named `mypack` that contains two modules, `p1.py` and `p2.py`. The contents of the modules are:

p1.py

```
def m1():
    print("first module")
```

p2.py

```
def m2():
    print("second module")
```

Fig. 2.1

- Here `mypack` a folder/directory which consist of p1.py and p2.py. We can refer these two modules with dot notation (`mypack.p1`, `mypack.p2`) and import them with the one of the following syntaxes:

Syntax 1: `import <module_name>[, <module_name> ...]`

Example:

```
>>> import mypack.p1,mypack.p2
>>> mypack.p1.m1()
first module
>>> p1.m1()
```


Syntax 2: `from <module_name> import <name(s)>`

Example:

```
>>> from mypack.p1 import m1
>>> m1()
first module
>>>
```

Syntax 3: `from <module_name> import <name> as <alt_name>`

Example:

```
>>> from mypack.p1 import m1 as function
>>> function()
first module
>>>
```

Syntax 4: `from <package_name> import <modules_name>[, <module_name> ...]`

Example:

```
>>> from mypack import p1,p2
>>> p1.m1()
first module
>>> p2.m2()
second module
>>>
```

2.4.2 Predefined Packages

- NumPy and SciPy are the standards packages used by Python programming.
- NumPy enriches the programming language Python with powerful data structures, implementing multi-dimensional arrays and matrices.
- SciPy (Scientific Python) is often mentioned in the same breath with NumPy. SciPy needs Numpy, as it is based on the data structures of Numpy and furthermore its basic creation and manipulation functions.
- It extends the capabilities of NumPy with further useful functions for minimization, regression, Fourier-transformation and many others.
- Both NumPy and SciPy are not part of a basic Python installation. They have to be installed after the Python installation. NumPy has to be installed before installing SciPy.

Math:

- Some of the most popular mathematical functions are defined in the math module. These include trigonometric functions, representation functions, logarithmic functions and angle conversion functions.
- Two mathematical constants are also defined in math module.

- **Pie (π)** is a well-known mathematical constant, which is defined as the ratio of the circumference to the diameter of a circle and its value is 3.141592653589793.

```
>>> import math
>>> math.pi
3.141592653589793
>>>
```

- Another well-known mathematical constant defined in the math module is **e**. It is called Euler's number and it is a base of the natural logarithm. Its value is 2.718281828459045.

```
>>> import math
>>> math.e
2.718281828459045
>>>
```

NumPy:

- NumPy is the fundamental package for scientific computing with Python. NumPy stands for "Numerical Python". It provides a high-performance multidimensional array object, and tools for working with these arrays.
- An array is a table of elements (usually numbers), all of the same type, indexed by a tuple of positive integers and represented by a single variable. NumPy's array class is called ndarray. It is also known by the alias array.
- In NumPy arrays, the individual data items are called elements. All elements of an array should be of the same type. Arrays can be made up of any number of dimensions.
- In NumPy, dimensions are called axes. Each dimension of an array has a length which is the total number of elements in that direction.
- The size of an array is the total number of elements contained in an array in all the dimension. The size of NumPy arrays are fixed; once created it cannot be changed again.
- Numpy arrays are great alternatives to Python Lists. Some of the key advantages of Numpy arrays are that they are fast, easy to work with, and give users the opportunity to perform calculations across entire arrays.

SciPy:

- SciPy is a library that uses NumPy for more mathematical functions. SciPy uses NumPy arrays as the basic data structure, and comes with modules for various commonly used tasks in scientific programming, including linear algebra, integration (calculus), ordinary differential equation solving, and signal processing.

Matplotlib:

- matplotlib.pyplot is a plotting library used for 2D graphics in python programming language. It can be used in python scripts, shell, web application servers and other graphical user interface toolkits.
- There are various plots which can be created using python matplotlib like bar graph, histogram, scatter plot, area plot, pie plot.

Pandas:

- Pandas is an open-source Python Library providing high-performance data manipulation and analysis tool using its powerful data structures.
- It is built on the NumPy package and its key data structure is called the DataFrame. DataFrames allow us to store and manipulate tabular data in rows of observations and columns of variables.

2.4.3 User Defined Packages

- We organize a large number of files in different folders and subfolders based on some criteria, so that we can find and manage them easily. In the same way, a package in Python takes the concept of the modular approach to next logical level.
- As we know, a module can contain multiple objects, such as classes, functions, etc. A package can contain one or more relevant modules.
- Physically, a package is actually a folder containing one or more module files. Let's create a package named Mypack, using the following steps:

Step 1: Create a folder Mypack on `C:\Users\AppData\Local\Programs\Python\Python3.9.6\"`.

Create modules Message.py and Mathematics.py with following code:

Message.py

```
def SayHello(name):
    print("Hello " + name)
    return
```

Mathematics.py

```
def sum(x,y):
    return x+y
def average(x,y):
    return (x+y)/2
def power(x,y):
    return x**y
```

Step 2: Create an empty _ _init_ _.py file in the Mypack folder. The package folder contains a special file called _ _init_ _.py, which stores the package's content. It serves two purposes:

1. The Python interpreter recognizes a folder as the package if it contains _ _init_ _.py file.
2. _ _init_ _.py exposes specified resources from its modules to be imported.

An empty _ _init_ _.py file makes all functions from above modules available when this package is imported. Note that _ _init_ _.py is essential for the folder to be recognized by Python as a package. We can optionally define functions from individual modules to be made available.

Step 3: Create P1.py file in Mypack folder and write following code:

```
from Mypack import Mathematics
from Mypack import Message
greet.SayHello("Meenakshi")
x=functions.power(3,2)
print("power(3,2): ", x)
```

Output:

```
Hello Meenakshi
power(3,2): 9
```

Using __init__.py File:

- The __init__.py file is normally kept empty. However, it can also be used to choose specific functions from modules in the package folder and make them available for import. Modify __init__.py as below:

__init__.py

```
from .Mathematics import average, power
from .Message import SayHello
```

- The specified functions can now be imported in the interpreter session or another executable script.

Create test.py in the Mypack folder and write following code:

test.py

```
from Mypack import power, average, SayHello
SayHello()
x=power(3,2)
print("power(3,2): ", x)
```

- Note that functions power() and SayHello() are imported from the package and not from their respective modules, as done earlier. The output of above script is:

```
Hello world
power(3,2): 9
```

2.4.4 Package Examples

- We have included a __init__.py, file inside a directory to tell Python that the current directory is a package.
- Whenever we want to create a package, then we have to include __init__.py file in the directory. We can write code inside it.
- Let's create a simple package that has the following structure:
  Package (university)
  - __init__.py
  - student.py
  - faculty.py


```
# student.py
class Student:
 def __init__(self, student):
 self.name = student['name']
 self.gender = student['gender']
 self.year = student['year']
 def get_student_details(self):
 return f"Name: {self.name}\nGender: {self.gender}\nYear: {self.year}"
# faculty.py
class Faculty:
 def __init__(self, faculty):
 self.name = faculty['nane']
 self.subject = faculty['subject']
 def get_faculty_details(self):
return f"Name: {self.name}\nSubject: {self.subject}"
```

- We have the above in the student.py and faculty.py files. Let's create another file to access those classed inside it. Now, inside the package directory create a file named testing.py and include the following code:

```
# testing.py
# importing the Student and Faculty classes from respective files
from student import Student
from faculty import Faculty
# creating dicts for student and faculty
student_dict = {'name': 'Ram', 'gender': 'Male', 'year': '3'}
faculty_dict = {'name': 'Radha', 'subject': 'Programming'}
# creating instances of the Student and Faculty classes
student = Student(student_dict)
faculty = Faculty(faculty_dict)
# getting and printing the student and faculty details
print(student.get_student_details())
print()
print(faculty.get_faculty_details())
```

Output:

```
Name: Ram
Gender: Male
Year: 3
Name: Radha
Subject: Programming
```

- We have seen how to create and to access a package in Python. And this is a simple package. There might be plenty of sub-packages and files inside a package. Let's see how to access subpackage modules.
- Create a directory with the following structure:

  Package (university)
  - __init__.py
  - Subpackage (student)
    - __init__.py
    - main.py
    - ...
  - testing.py
- Copy the above student code and place it here. Now, let's see how to access it in the testing.py file. Add the following in the testing.py file.

Program 2.2:

```
# testing.py
from student.main import Student
# creating dicts for student
student_dict = {'name': 'Ram', 'gender': 'Male', 'year': '3'}
# creating instances of the Student class
student = Student(student_dict)
# getting and printing the student details
print(student.get_student_details())
```

If you run the **testing.py** file, then you will get the following result.

Output:

```
Name: Ram
Gender: Male
Year: 3
```

Program 2.3: For NumPy with array object.

```
>>> import numpy as np
>>> a=np.array([1,2,3])       # one dimensional array
>>> print(a)
[1 2 3]
>>> arr=np.array([[1,2,3],[4,5,6]]) # two dimensional array
>>> print(arr)
[[1 2 3]
 [4 5 6]]
>>> type(arr)
<class 'numpy.ndarray'>
>>> print("No. of dimension: ", arr.ndim)
```



```
No. of dimension: 2
>>> print("Shape of array: ", arr.shape)
Shape of array: (2, 3)
>>> print("size of array: ", arr.size)
size of array: 6
>>> print("Type of elements in array: ", arr.dtype)
Type of elements in array: int32
>>> print("No of bytes:", arr.nbytes)
No of bytes: 24
```

Program 2.4: Using linalg sub package of SciPy.

```
>>> import numpy as np
>>> from scipy import linalg
>>> a = np.array([[1., 2.], [3., 4.]])
>>> linalg.inv(a)   # find inverse of array
array([[-2. , 1. ],
       [ 1.5, -0.5]])
>>>
```

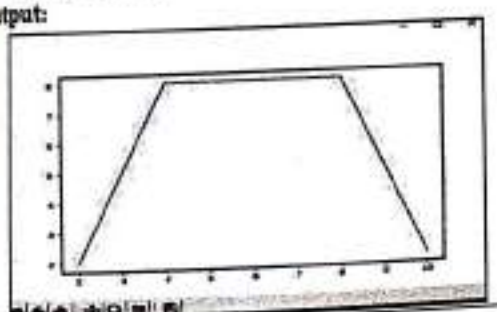
Program 2.5: Using linalg sub package of SciPy.

```
>>> import numpy as np
>>> from scipy import linalg
>>> a = np.array([[1,2,3], [4,5,6], [7,8,9]])
>>> linalg.det(a)   # find determinant of array
0.0
>>>
```

Program 2.6: For line plot.

```
>>> from matplotlib import pyplot as plt
>>> x=[2,4,8,10]
>>> y=[2,8,8,2]
>>> plt.plot(x,y)
[<matplotlib.lines.Line2D object at 0x02E69870>]
>>> plt.show()
```

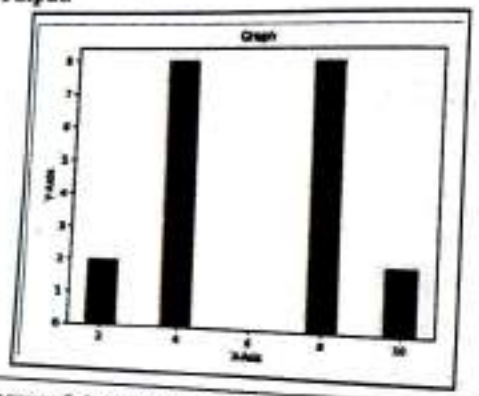
Output:



Program 2.7: For bar graph.

```
>>> from matplotlib import pyplot as plt
>>> x=[2,4,8,10]
>>> y=[2,8,8,2]
>>> plt.xlabel('X-Axis')
Text(0.5, 0, 'X-Axis')
>>> plt.ylabel('Y-Axis')
Text(0, 0.5, 'Y-Axis')
>>> plt.title('Graph')
Text(0.5, 1.0, 'Graph')
>>> plt.bar(x,y,label="Graph",color='r',width=.5)
<BarContainer object of 4 artists>
>>> plt.show()
```

Output:



Program 2.8: Using Series data structure of Panda.

```
>>> import pandas as pd
>>> import numpy as np
>>> numpy_arr = array([2, 4, 6, 8, 10, 20])
>>> si = pd.Series(arr)
>>> print(si)
0     2
1     4
2     6
3     8
4    10
5    20
dtype: int32
```


Program 2.9: Using Series data structure of Panda.

```
>>> import pandas as pd
>>> data=[10,20,30,40,50]
>>> index=['a','b','c','d','e']
>>> si=pd.Series(data,index)
>>> si
a 10
b 20
c 30
d 40
e 50
dtype: int64
>>>
```

Program 2.10: Write an anonymous function to calculate area of square.

Code:

```
area = lambda x:x*x
n=int(input("enter side of square"))
print("area of square:",area(n))
```

Output:

```
enter side of square5
area of square: 25
```

Program 2.11: Write a Python function to multiply all the numbers in a list. Sample-List:(8,2,3,-1,7) Expected Output: -336

Code:

```
def mul(str):
 sum=1
 for i in str:
 sum = sum*i
 return sum
str = [8,2,3,-1,7]
print(str)
print (mul(str))
```

Output:

```
string [8, 2, 3, -1, 7]
-336
```

Program 2.12: Write a Python function to check whether a number is in a given range.

Code:

```
def ran(n,m):
 if n in range(m):
 print("number is in range")
```

```
    else:
     print("number is out is range")
m=int(input("enter range from 1 to: "))
n=int(input("enter number: "))
ran(n,m+1)
```

Output:

```
enter range from 1 to: 10
enter number: 5
number is in range
enter range from 1 to: 10
enter number: 11
number is out is range
```

Program 2.13: Create a function showEmployee() in such a way that it should accept employee name, and it's salary and display both, and if the salary is missing in function call it should show it as 9000

Code:

```
def show_emp(name,sal=9000):
 print("employee name: ",name)
 print("salary: ",sal)
show_emp("nikhil")
```

Output:

```
employee name: nikhil
salary: 9000
```

Program 2.14: Write a Python function that takes a number as a parameter and check the number is prime or not.

Code:

```
def checkforprime(m):
 if m>1:
 for i in range(2,m):
 if(m%i)==0:
 return 0
 return 1
n=int(input("enter number: "))
a=checkforprime(n)
if a==1:
 print(n,": is prime")
else:
 print(n,": is not prime")
```

Output:

```
enter number: 5
5: is prime
enter number: 10
10: is not prime
```


Program 2.15: Write a generator function that reverses a given string.

Code:

```
def rev(str):
 return str[::-1]

n=input("input string")
print("string: ",n)
print("reversed string: ",rev(n))
```

Output:

```
input stringakshay
string: akshay
reversed string: yahska
```

Program 2.16: Write a recursive function to calculate the sum of numbers from 0 to 10.

Code:

```
def sum(n):
 if n<=1:
 return n
 else:
 return n+sum(n-1)
print("sum: ",sum(10))
```

Output:

```
sum: 55
```

Summary

- A module in Python programming allows us to logically organize the python code. A module is a single source code file. The module in Python have the .py file extension. The name of the module will be the name of the file.
- In Python, packages allow us to create a hierarchical file directory structure of modules. For example, mymodule.mod1 stands for a module mod1, in the package mymodule.
- A Python package is a collection of modules which have a common purpose. In short, modules are grouped together to forms packages.
- The import statement is used to imports a specific module by using its name.
- A module is a collection of Python objects such as functions, classes, and so on. Python interpreter is bundled with a standard library consisting of large number of built-in modules,
- math and cmath, Decimal Module, Fractions Module, Statistics Module, itertools Module, functools Module, Operator Module, Time Module are built in modules having several functions
- Datetime, calendar modules will display the date, time, and calendar.

- The sys modules provide variables for better control over input or output. We can even redirect the input and output to other devices. This can be done using the variables – stdin, stdout, stderr
- The random module provides functions to perform these types of operations. The function is not accessible directly, so we need to import random module and then we need to call this function using random static object.
- Function blocks begin with the keyword def followed by the function name and parentheses ().
- Any input parameters or arguments should be placed within these parentheses. You can also define parameters inside these parentheses.
- A package is a hierarchical file directory structure that defines a single Python application environment that consists of modules and subpackages and sub-subpackages and so on.
- A package is a collection of Python modules, i.e., a package is a directory of Python modules containing an additional _ _init_ _.py file (For example: Phone/_ _init_ _.py).
- NumPy and SciPy are the standards packages used by Python programming.
- matplotlib.pyplot is a plotting library used for 2D graphics in python.
- Pandas is an open-source Python Library providing high-performance data manipulation and analysis tool using its powerful data structures.

Check Your Understanding

1. Which of these definitions correctly describes a module?
   (a) Denoted by triple quotes for providing the specification of certain program elements
   (b) Design and implementation of specific functionality to be incorporated into a program
   (c) Defines the specification of how it is to be used
   (d) Any program that reuses code
2. Which of the following is not an advantage of using modules?
   (a) Provides a means of reuse of program code
   (b) Provides a means of dividing up tasks
   (c) Provides a means of reducing the size of the program
   (d) Provides a means of testing individual parts of the program
3. Program code making use of a given module is called a _____ of the module.
   (a) Client (b) Docstring
   (c) Interface (d) Modularity
4. _____ is a string literal denoted by triple quotes for providing the specifications of certain program elements.
   (a) Interface (b) Modularity
   (c) Client (d) Docstring

5. Which of the following is false about "from-import" form of import?
   (a) The syntax is: from modulename import identifier
   (b) This form of import prevents name clash
   (c) The namespace of imported module becomes part of importing module
   (d) The identifiers in module are accessed directly as: identifier
6. _____ is an amazing visualization library in Python for 2D plots of arrays.
   (a) Scilearn (b) matplotlib
   (c) matlab (d) scilab
7. A Python module is a file with the _____ file extension that contains valid Python code.
   (a) .module (b) .py
   (c) .pym (d) .pymodule
8. Which of the following is contained in NumPy library?
   (a) n-dimensional array object
   (b) tools for integrating C/C++ and Fortran code
   (c) fourier transform
   (d) all of the Mentioned
9. To create sequences of numbers, NumPy provides a function _____ analogous to range that returns arrays instead of lists.
   (a) arange (b) aspace
   (c) aline (d) all of the Mentioned
10. _____ basic data structure of Pandas can be think of SQL table or a spreadsheet data representation.
    (a) Dataframe (b) list
    (c) table (d) matrix

Answers

1. (b)	2. (c)	3. (a)	4. (d)	5. (b)	6. (b)	7. (b)	8. (d)	9. (a)	10. (a)

Practice Questions

Q I. Answer the following questions in short.

1. What is module? Write the steps of creating the module.
2. What is the use of import statement?
3. Name any five built-in modules in python.
4. What is the use of math module?
5. Write any three function of Math module.
6. Which module is used in python to find mean and mode?
7. State any five functions of time module.

8. State three variable used by sys module.
9. What are the advantages of using Matplotlib library?
10. What are the different output formats supported by Matplotlib library.
11. What is the use of random function?
12. State predefine packages in python.
13. Which package is used for 2D graphics in python?

Q II. Answer the following questions in long.

1. Write the use of import statement with an example.
2. Which are the three different ways to import modules in Python?
3. Explain math and cmath Modules in detail.
4. Explain the use of statistics module with an example.
5. Explain functools module in brief.
6. Explain Datetime module with an example.
7. Write a python program to display the current date and time.
8. Explain the sys module in detail.
9. How to create a package in python.
10. Explain the features of NumPy.
11. Differentiate between python list and NumPy array.
12. Explain the features of Pandas in python.
13. What are the benefits of pandas over c/c++ for data analysis?

Q III. Define terms.

1. Module
2. package
3. import

3... Classes, Objects and Inheritance

Learning Objectives ...

- To learn about Classes, Object and Inheritance in Python.
- To understand concept Classes as User Defined Data Types.
- To know about Object as Instance of Classes.
- To study Variables and methods in class.
- To learn Inheritance in details.
- To understand IS-A Relationship and HAS-A Relationship.

3.1 CLASSES AND OBJECTS

- Python is an "Object Oriented Programming Language." This means that almost all the code is implemented using a special construct called classes.
- Programmers use classes to keep related things together. This is done using the keyword "class," which is a grouping of object oriented constructs.
- For example, for any bank employee who want to fetch the customer details online would go to customer class, where all its attributes like transaction details, withdrawal and deposit details, outstanding debt, etc. would be listed out.
- A class is a Python object with several characteristics:
  - You can call a class object as if it were a function. The call returns another object, known as an instance of the class; the class is also known as the type of the instance.
  - A class has arbitrarily named attributes that you can bind and reference.
  - The values of class attributes can be descriptors (including functions), normal data objects.
  - Class attributes bound to functions are also known as methods of the class.
  - A method can have a special Python-defined name with two leading and two trailing underscores. Python implicitly invokes such special methods, if a class supplies them, when various kinds of operations take place on instances of that class.

- An Object is an instance of a Class. A class is like a blueprint while an instance is a copy of the class with actual values. It's not an idea anymore, it's an actual dog, like a dog of breed pug who's seven years old. You can have many dogs to create many different instances, but without the class as a guide, you would be lost, not knowing what information is required.
- An object consists of :
  - **State:** It is represented by the attributes of an object. It also reflects the properties of an object.
  - **Behavior:** It is represented by the methods of an object. It also reflects the response of an object to other objects.
  - **Identity:** It gives a unique name to an object and enables one object to interact with other objects.

3.1.1 Classes as User Defined Data Type

- A class in python is user defined data type.
- A class combination of data member and member function.
- The data member and member functions of a class can be accessed after creating an instance of a class.
- Instance of class is known as Object.
- We can define a custom data type by creating a class in Python.

Example:

```
class Data:
    pass
d=Data()
print(type(d))
```

Output:

```
<class '__main__.Data'>
```

Here Data is the user define class and d is the object of the class.

3.1.2 Objects as Instances of Classes

- An object is created using the constructor of the class. This object will then be called the instance of the class.
- Instance variables are variables used for data that is unique to a particular instance. Whereas, Class Variables are variables that are shared by all instances of a class.
- In Python, we create instances in the following manner:

```
Instance = class(arguments)
```

Example:

```
"This would create first object of Employee class"
emp1 = Employee("Nirali", 10000)
"This would create second object of Employee class"
emp2 = Employee("Prakashan", 5000)
```


- You can also provide the values for the attributes at runtime. This is done by defining the attributes inside the init method.
- To create instances of a class, you call the class using class name and pass in whatever arguments its __init__ method accepts.
- Use the __init__() function to assign values to object properties, or other operations that are necessary to do when the object is being created.

Program 3.1: Create a class named Person, use the __init__() function to assign values for name and age.

```
class Person:
  def __init__(self, name, age):
    self.name = name
    self.age = age

p1 = Person("Omkar", 26)

print(p1.name)
print(p1.age)
```

Output:

```
Omkar
26
```

3.1.3 Creating Class and Objects

- Python classes are data structures used to define objects. They contain data values and define behavioral characteristics. In Python, you define a class by using the key *class*.
- A Class is like an object constructor, or a "blueprint" for creating objects. The *class* statement creates a new class definition. The name of the class immediately follows the keyword *class* followed by a colon as follows.
- **Syntax:**

```
class ClassName:
    <statement-1>
    .
    .
    .
    <statement-N>
```

- **Example:** Create a class named MyClass, with a property named x.

```
class MyClass:
  x = 5
print(MyClass)
```

- To define class you need to consider following points:

Step 1 : In Python, classes are defined by the "**Class**" keyword.

```
class myClass():
```

Step 2 : Inside classes, you can define functions or methods that are part of this class.

```
def method1 (self):
    print "Nirali Prakashan"
def method2 (self,someString):
    print "Software Testing:" + SomeString
```

- Here we have defined method1 that prints "Nirali Prakashan."
- Another method we have defined is method2 that prints "Software Testing"+ SomeString. SomeString is the variable supplied by the calling method

Step 3 : Everything in a class is indented, just like the code in the function, loop, if statement, etc.

Note: About using "self" in Python

- The self-argument refers to the object itself. Hence the use of the word self. So inside this method, self will refer to the specific instance of this object that's being operated on.
- Self is the name preferred by convention by Pythons to indicate the first parameter of instance methods in Python. It is part of the Python syntax to access members of objects.

Step 4 : To make an object of the class.

```
c = myClass()
```

Step 5 : To call a method in a class.

```
c.method1()
c.method2("Testing is fun")
```

- Notice that when we call the method1 or method2, we don't have to supply the self keyword. That's automatically handled for us by the Python runtime.
- Python runtime will pass "self" value when you call an instance method on in instance, whether you provide it deliberately or not.
- You just have to care about the non-self arguments.

Step 6 : Here is the complete code

Program 3.2: Program for class and object creation.

```
class myClass():
  def method1(self):
      print("Nirali prakashan")

  def method2(self,someString):
      print("Software Testing:" + someString)
def main():
  # exercise the class methods
  c = myClass ()
```



```
  c.method1()
  c.method2(" Testing is fun")
if __name__== "__main__":
  main()
```

Output:

```
Nirali Prakashan
Software Testing: Testing is fun
```

Accessing Members:

- The object's attributes are access by using the dot operator with object. Class variable would be accessed using class name as follows:

```
emp1.displayEmployee()
emp2.displayEmployee()
print "Total Employee %d" % Employee.empCount
```

- The syntax used to assign a value to an attribute of an object is,

```
<object> . <attribut> = <value>
```

- Now, putting all the concepts together:

Program 3.3: Program for accessing members.

```
class Employee:
   'Common base class for all employees'
   empCount = 0

   def __init__(self, name, salary):
      self.name = name
      self.salary = salary
      Employee.empCount += 1

   def displayCount(self):
     print("Total Employee %d" % Employee.empCount)

   def displayEmployee(self):
      print("Name: ", self.name,  ", Salary: ", self.salary)

"This would create first object of Employee class"
emp1 = Employee("Nirali", 10000)
"This would create second object of Employee class"
emp2 = Employee("Pragati", 5000)
emp1.displayEmployee()
emp2.displayEmployee()
print("Total Employee %d", % Employee.empCount)
```

Output:

```
Name:  Nirali ,Salary:  10000
Name:  Pragati ,Salary:  5000
Total Employee 2
```

- You can add, remove, or modify attributes of classes and objects at any time:

```
emp1.age = 7  # Add an 'age' attribute.
emp1.age = 8  # Modify 'age' attribute.
del emp1.age  # Delete 'age' attribute.
```

3.1.4 Creating Objects By Passing Values

- In pass by value the function is provided with a copy of the argument object passed to it by the caller. That means the original object stays intact and all changes made are to a copy of the same and stored at different memory locations.
- When you pass function arguments by reference, those arguments are only references to existing values. In contrast, when you pass arguments by value, those arguments become independent copies of the original values.
- Python utilizes a system, which is known as "Call by Object Reference" or "Call by assignment". In the event that you pass arguments like whole numbers, strings or tuples to a function, the passing is like call-by-value because you can not change the value of the immutable objects being passed to the function.
- Whereas passing mutable objects can be considered as call by reference because when their values are changed inside the function, then it will also be reflected outside the function.

Program 3.4: Program for call by value.

```
string = "Python"
def test(string):
    string = "Pythonprogramming"
    print("Inside Function:", string)
test(string)
print("Outside Function:", string)
```

Output:

```
Inside Function: Pythonprogramming
Outside Function: Python
```

3.1.5 Variables and Methods in a Class

3.1.5.1 Class Variables

- Class or static variables are shared by all objects. Instance or non-static variables are different for different objects (every object has a copy of it).
- Class variables are defined within the class construction. Because they are owned by the class itself, class variables are shared by all instances of the class. They therefore will generally have the same value for every instance unless you are using the class variable to initialize a variable.
- Defined outside of all the methods, class variables are, by convention, typically placed right below the class header and before the constructor method and other methods.
- Instance variables are owned by instances of the class. This means that for each object or instance of a class, the instance variables are different.

- **Syntax:**

```
class ClassName(object):
    self.instance_variable = value #value specific to instance
    class_variable = value #value shared across all class instances

#accessing instance variable
class_instance = ClassName()
class_instance.instance_variable

#accessing class variable
ClassName.class_variable
```

- For example, let a computer science student be represented by class **CSStudent**. The class may have a static variable whose value is "cse" for all objects. And class may also have non-static members like **name** and **roll**.
- The Python approach is simple, it doesn't require a static keyword. All variables which are assigned a value in class declaration are class variables. And variables which are assigned values inside class methods are instance variables.

Program 3.5: Python program to accessing the class variables.

```
# Class for Computer Science Student
class CSStudent:
    stream = 'cse'                  # Class Variable
    def __init__(self,name,roll):
        self.name = name            # Instance Variable
        self.roll = roll            # Instance Variable

# Objects of CSStudent class
a = CSStudent('Geek', 1)
b = CSStudent('Nerd', 2)

print(a.stream)  # prints "cse"
print(b.stream)  # prints "cse"
print(a.name)    # prints "Geek"
print(b.name)    # prints "Nerd"
print(a.roll)    # prints "1"
print(b.roll)    # prints "2"

# Class variables can be accessed using class
# name also
print(CSStudent.stream) # prints "cse"
```

Output:

```
cse
cse
Geek
Nerd
1
2
cse
```

3.1.5.2 Class Methods

- There are three types of methods in Python: **Instance methods, Static methods,** and **Class methods.**

1. **Instance Methods:**

- Instance methods are the most common type of methods in Python classes. These are so called because they can access unique data of their instance. If you have two objects each created from a car class, then they each may have different properties. They may have different colors, engine sizes, seats, and so on.
- Instance methods must have **self** as a parameter, but you don't need to pass this in every time. Self is another Python special term. Inside any instance method, you can use self to access any data or methods that may reside in your class. You won't be able to access them without going through self.

2. **Static Methods:**

- Static methods are methods that are related to a class in some way, but don't need to access any class-specific data. Static methods are defined without self arguments and don't need to instantiate an instance, they can be called directly on the class itself.
- Static methods are great for utility functions, which perform a task in isolation. They don't need to (and cannot) access class data. They should be completely self-contained, and only work with data passed in as arguments. You may use a static method to add two numbers together, or print a given string.
- Static methods and class methods are created by wrapping methods in objects of the static method classes. Static methods are defined without self arguments, and they can be called directly on the class itself.

3. **Class Methods:**

- Class methods know about their class. They can't access specific instance data, but they can call other static methods.
- Class methods don't need self as an argument, but they do need a parameter called cls. This stands for class, and like self, gets automatically passed in by Python.
- Class methods are created using the @classmethod decorator. Class methods can manipulate the class itself, which is useful when you're working on larger, more complex projects.
- Here is a simple example:

```
class MyClass:
    def smeth();
        print('This is a static method')
    smeth = staticmethod(smeth)
    def cmeth(cls):
        print('This is a class method of', cls)
    cmeth = classmethod(cmeth)
```

- In Python 2.4, a new syntax was introduced for wrapping methods like this, called decorators. (They actually work with any callable objects as wrappers and can be used on both methods and functions.) You specify one or more decorators (which are applied in reverse order) by listing them above the method (or function), using the @ operator.


```
class MyClass:
    @staticmethod
def smeth():
    print('This is a static method')
@classmethod
def cmeth(cls):
    print('This is a class method of', cls)
```

Once you've defined these methods, they can be used like this (that is, without instantiating the class):

```
>>> MyClass.smeth()
This is a static method
>>> MyClass.cmeth()
This is a class method of <class 1__main__.MyClass',
```

Differentiate between deep and shallow copy:

- A deep copy copies an object into another. This means that if you make a change to a copy of an object, it won't affect the original object. In Python, we use the function deepcopy() for this, and we import the module copy. We use it like:

```
>>> import copy
>>> b=copy.deepcopy(a)
```

- A shallow copy, however, copies one object's reference to another. So, if we make a change in the copy, it will affect the original object. For this, we have the function copy(). We use it like:

```
>>> b=copy.copy(a)
```

3.2 INHERITANCE

- Inheritance is a feature of Object Oriented Programming. It is used to specify that one class will get most or all of its features from its parent class. It is a very powerful feature which facilitates users to create a new class with a few or more modification to an existing class. The new class is called child class or derived class and the main class from which it inherits the properties is called base class or parent class.
- **Benefits of using Inheritance:**

1. Less code repetition, as the code which is common can be placed in the parent class, hence making it available to all the child classes.
2. **Structured Code**: By dividing the code into classes, we can structure our software better by dividing functionality into classes.
3. Make the code more scalable.

- Python Inheritance **Syntax:**

```
class BaseClass:
  Body of base class
class DerivedClass(BaseClass):
  Body of derived class
```

- Derived class inherits features from the base class where new features can be added to it. This results in re-usability of code.

3.2.1 Single Inheritance

- In inheritance, the child class acquires the properties and can access all the data members and functions defined in the parent class. A child class can also provide its specific implementation to the functions of the parent class.

Fig. 3.1: Single Inheritance

- **Syntax:**

1. class DerivedClassName(BaseClassName):
```
    <statement-1>
    .
    .
    .
    <statement-N>
```
2. class DerivedClassName(modulename.BaseClassName):
```
    <statement-1>
    .
    .
    .
    <statement-N>
```

Program 3.6: Program for single inheritance.

```
class Animal:
    def eat(self):
        print('Eating...')
class Dog(Animal):
    def bark(self):
        print('Barking...')
d=Dog()
d.eat()
d.bark()
```

Output:

```
Eating...
Barking...
```

Program 3.7: Program to create base class student and derived class test using single inheritance and display the marks of the student.

```
class Student:
#Common base class for all students
    def getData(self,rollno,name,course):
```



```
        self.rollno = rollno
        self.name = name
        self.course = course
    def displayStudent(self):
        print("Roll Number:", self.rollno)
        print("Name:", self.name)
        print("Course:", self.course)
#Inheritance
class Test(Student):
    def getMarks(self, marks):
        self.marks=marks
    def displayMarks(self):
        print("Total Marks:", self.marks)
r = int(input("Enter Roll Number:"))
n = input("Enter Name:")
c = input("Enter Course Name:")
m = int(input("Enter Marks:"))
#creating the object
print ("Result")
stud1=Test()
stud1.getData(r,n,c)
stud1.getMarks(m)
stud1.displayStudent()
stud1.displayMarks()
```

Output:

```
Enter Roll Number:20
Enter Name:Omkar
Enter Course Name:Python
Enter Marks:80
Result
Roll Number: 20
Name: Omkar
Course: Python
Total Marks: 80
```

3.2.2 Multiple Inheritance

- It is possible to inherit from more than one parent class. Such type of inheritance is called multiple inheritance. In this case all the attributes and methods of both the parent class will be available in the child class after inheritance. Fig. 3.2 illustrates multiple inheritance.

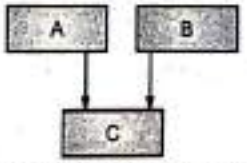


Fig. 3.2: Multiple Inheritance

- Here A and B are the parent classes and C is the child class. The attributes and methods of both classes A and B are now available in C after inheritance.
- Similar way, you can derive a class from multiple parent classes as follows.
- **Syntax:**

```
class A:        # define your class A
.....

class B:        # define your class B
.....

class C(A, B): # subclass of A and B
.....
```

Program 3.8: Program for Multiple inheritance.

```
class First(object):
  def __init__(self):
    super(First, self).__init__()
    print("first")

class Second(object):
  def __init__(self):
    super(Second, self).__init__()
    print("second")

class Third(Second, First):
  def __init__(self):
    super(Third, self).__init__()
    print("third")

Third();
```

Output:

```
first
second
third
```

- You can use issubclass() or isinstance() functions to check a relationships of two classes and instances.
  - The **issubclass(sub, sup)** boolean function returns true if the given subclass **sub** is indeed a subclass of the superclass **sup**.
  - The **isinstance(obj, Class)** boolean function returns true if *obj* is an instance of class *Class* or is an instance of a subclass of Class.

Program 3.9: Program for Inheritance.

```
class Student:
#Common base class for all students
    def getData(self,rollno,name,course):
        self.rollno=rollno
        self.name=name
        self.course=course
    def displayStudent(self):
        print("Roll Number:",self.rollno)
        print("name:", self.name)
        print("Course:", self.course)
#Inheritance
class Test (Student):
    def getMarks(self, marks):
        self.marks=marks
    def displayMars (self):
        print("Total Marks:", self.marks)
class Sports:
    def getSportsMarks(self, spmarks):
        self.spmarks=spmarks
    def displaySportsMarks(self):
        print("Sports Marks:", self,spmarks)
#Multiple Inheritance
class Result(Test,Sports):
    def calculateGrade(self):
        m=self.marks+self.spmarks
        if m>480: self.grade="Distinction"
        elif m>360:self.grade="First Class"
        elif m>240:self.grade="Second Class"
        else:self.grade="Failed"
    print("Result:",self.grade)
#Main Program
r=int(input("Enter Roll Number:"))
n=input("Enter Name:")
c=input("Enter Course Name:")
m=int(input("Enter Marks:"))
s=int(input("Enter Sports marks:"))
stud1=Result()  #instance of child
stud1.getData(r,n,c)
```

```
studl.getMarks(m)
studl.getSportsMarks(s)
studl.displayStudent()
studl.displayMarks()
studl.displaySportsMarks()
studl.calculateGrade()
```

Output:

```
Enter Roll Number:10
Enter Name:Bob
Enter Course Name:MS
Enter Marks:190
Enter Sports marks:200
Result
Roll Number: 10
Name: Bob
Course: MS
Total Marks: 190
Sports Marks: 200
Result: First Class
```

3.2.3 Multilevel Inheritance

- We can inherit a derived class from another derived class, this process is known as multilevel inheritance.

Fig. 3.3: Multilevel inheritance

Syntax:

```
class class1:
    .
    .
class class2(class1):
    .
    .
class class3(class2):
    .
    .
```


Program 3.10: Program for Python Multilevel Inheritance.

```
class Animal:
    def eat(self):
        print 'Eating...'
class Dog(Animal):
    def bark(self):
        print 'Barking...'
class BabyDog(Dog):
    def weep(self):
        print 'Weeping...'
d=BabyDog()
d.eat()
d.bark()
d.weep()
```

Output:

```
Eating...
Barking...
Weeping
```

3.2.4 Hybrid Inheritance

- Inheritance consisting of multiple types of inheritance is called hybrid inheritance.

Program 3.11: Program for Hybrid inheritance.

```
class School:
    def func1(self):
        print("This function is in school.")
class Student1(School):
    def func2(self):
        print("This function is in student 1. ")
class Student2(School):
    def func3(self):
        print("This function is in student 2.")
class Student3(Student1, School):
    def func4(self):
        print("This function is in student 3.")

# Driver's code
object = Student3()
object.func1()
object.func2()
```

Output:

```
This function is in school.
This function is in student 1.
```

3.2.5 Hierarchical Inheritance

- When more than one derived classes are created from a single base this type of inheritance is called hierarchical inheritance. In following program, we have a parent (base) class and two child (derived) classes.

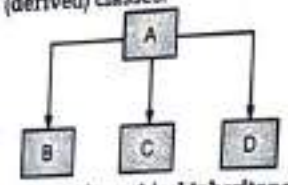


Fig. 3.4: Hierarchical Inheritance

Program 3.12: Program for Hierarchical Inheritance.

```
class Parent:
      def func1(self):
          print("This function is in parent class.")
# Derived class1
class Child1(Parent):
      def func2(self):
          print("This function is in child 1.")
# Derivied class2
class Child2(Parent):
      def func3(self):
          print("This function is in child 2.")
# Driver's code
object1 = Child1()
object2 = Child2()
object1.func1()
object1.func2()
object2.func1()
object2.func3()
```

Output:

```
This function is in parent class.
This function is in child 1.
This function is in parent class.
This function is in child 2.
```

3.2.6 IS-A Relationship and HAS-A Relationship

- Composition is a concept that models a HAS-A relationship. It enables creating complex types by combining objects of other types. This means that a class Composite can contain an object of another class Component. This relationship means that a Composite has a Component.

- The member of one class inside a class can be access using two concepts:
  - By Composition(Has-A Relation)
  - By Inheritance(Is-A Relation)

IS-A Relationship:

- In object oriented programming, the concept of IS-A is a totally based on Inheritance, which can be of two types Class Inheritance or Interface Inheritance. It is just like saying "A is a B type of thing". For example, Apple is a Fruit, Car is a Vehicle etc. Inheritance is unidirectional. For example, House is a Building. But Building is not a House.

HAS-A Relationship:

- Composition (HAS-A) simply mean the use of instance variables that are references to other objects. For example, Maruti has Engine, or House has Bathroom.

```
class college:
    # college specific functionality
    ...
    ...
    ...

class Teacher:
    ob = college()
    ob.method1()
    ob.method2()
    ...
    ...
```

- In above example, class Teacher HAS-A College class reference. Here inside class Teacher also we can create different variables and methods. Using object reference of College class inside Teacher class we can easily access each and every member of College class inside Teacher class.
- IS-A relationship based on Inheritance, which can be of two types Class Inheritance or Interface Inheritance.
- Has-a relationship is composition relationship which is a productive way of code reuse.

Program 3.13: Accessing the members of the class using instance of the class inside another class.

```
class Employee:
    # constructor for initialization
    def __init__(self, name, age):
        self.name = name
        self.age = age

    # instance method
```

```
    def emp_data(self):
        print('Name of Employee : ', self.name)
        print('Age of Employee : ', self.age)

class Data:
    def __init__(self, address, salary, emp_obj):
        self.address = address
        self.salary = salary

        # creating object of Employee class
        self.emp_obj = emp_obj

    # instance method
    def display(self):

        # calling Employee class emp_data()
        # method
        self.emp_obj.emp_data()
        print('Address of Employee : ', self.address)
        print('Salary of Employee : ', self.salary)

# creating Employee class object
emp = Employee('Ronil', 20)

# passing obj. of Emp. class during creation
# of Data class object
data = Data('Indore', 25000, emp)

# call Data class instance method
data.display()
```

Output:

```
Name of Employee :  Ram
Age of Employee :  20
Address of Employee :  Pune
Salary of Employee :  50000
```

- Here we have 2 classes 'Employee' and 'Data'. Inside 'Data' class Constructor we are creating an object of Employee class due to which we can access the members of the Employee class easily. Inside the Data class Employee class object becomes an instance variable of "Data" class.

Additional Programs

Program 3.14: Python Program to Create a Class in which One Method Accepts a String from the User and Another method Prints it. Define a class named Country which has a method called printNationality. Define subclass named state from Country which has a method called printState. Write a method to print state, country and nationality.

```
class accept:
    def getDa(self):
        self.x= input("Enter a string")
    def putDa(self):
        print("String is:",self.x)
a=accept()
a.getDa()
a.putDa()
class country:
    def getNa(self):
        self.cnt= input("Enter a country")
        self.na= input("Enter Nationality")
    def printNationality(self):
        print("Country is:",self.cnt)
        print("Nationality is:",self.na)

class state(country):

    def getState(self):
        self.s= input("Enter a State")
    def printState(self):
        print("State is:",self.s)
st=state()
st.getNa()
st.printNationality()
st.getState()
st.printState()
```

Output:

```
Enter a string saranga
('String is:', 'saranga')
Enter a country india
Enter Nationality indian
('Country is:', 'india')
('Nationality is:', 'indian')
Enter a State maharashtra
('State is:', 'maharashtra')
```


Program 3.15: Write a Python class which has two methods get_String and print_String. get_String accept a string from the user and print_String print the string in upper case. Further modify the program to reverse a string word by word and print it in lower case.

```
ans=" "
ans1=" "
class String1:
    def get_string(self):
        self.s=raw_input("enter a string")
```

```
    def print_string(self):
        print(self.s.upper())
        print(self.s[::-1])
s= String1()
s.get_string()
s.print_string()
```

Output:

```
enter a string saranga sharad kulkarni
SARANGA SHARAD KULKARNI
inrakluk darahs agnaras
```

Program 3.16: Define a class named Rectangle which can be constructed by a length and width. The Rectangle class has a method which can compute the area and volume.

```
import math
class Rectangle():
        def __init__(self,l,b,h):
            self.l=l
        self.b=b
        self.h=h
        def area(self):
            print "Area of Rectangle is:",self.l*self.b
        def volume(self):
            print "Volume of Rectangle is:",self.l*self.b*self.h
l=int(input("Enter length of rectangle"))
b=int(input("Enter width of rectangle"))
h=int(input("Enter height of rectangle"))
r=Rectangle(l,b,h)
r.area()
r.volume()
```

Output:

```
Enter length of rectangle 12
Enter width of rectangle 34
Enter height of rectangle 56
Area of Rectangle is: 408
Volume of Rectangle is: 22848
```

Program 3.17: Define a class named Shape and its subclass (Square/Circle). The subclass has an init function which takes a an argument (length/redious). Both classes have an area and volume function which can print the area and volume of the shape where Shape's area is 0 by default.

```
import math
class Shape:
    def __init__(self):
        self.area=0
```



```
    def area():
        print("Inside shape")
class Square(Shape):
        def __init__(self,r):
            self.r=r
        def area(self):
            print("Area of Square is:",self.r*self.r)
        def perimeter(self):
            print("Perimeter of Square is:",2*self.r)
class Circle(Shape):
        def __init__(self,r):
            self.r=r
        def area(self):
            print("Area of Circle is:",math.pi*self.r*self.r)
        def perimeter(self):
            print("Perimeter of Square is:",2*math.pi*self.r)
s=Square(2)
s.area()
s.perimeter()
c=Circle(2)
c.area()
c.perimeter()
```

Output:

```
Area of Square is: 4
Perimeter of Square is: 4
Area of Circle is: 12.566370614359172
Perimeter of Square is: 12.566370614359172
```

Program 3.18: Write a Python Program to Accept, Delete and Display students details such as Roll.No, Name, Marks in three subject, using Classes. Also display percentage of each student.

```
class student:
    def accept(self):
        self.rno=int(raw_input("Enter roll no"))
        self.nm=raw_input("Enter name")
        self.m=list()
        for self.i in range(0,3):
            self.a=int(raw_input("Enter marks"))
            self.m.append(self.a)
    def display(self):
        print("Roll NO:",self.rno)
```

```
        print("Name:",self.nm)
        print("Subject marks:",self.m)
        print("Percentage: ",sum(self.m)*100/300,"%")
n=int(raw_input("how many object u want:"))
s=[]
for i in range(0,n):
    x=raw_input("enter object:")
    s.append(x)
print(s)

for j in range(len(s)):
    s[j]=student()
    s[j].accept()
    s[j].display()
```

Output:

```
how many object u want:3
enter object:s1
enter object:s2
enter object:s3
['s1', 's2', 's3']
Enter roll no 12
Enter name saranga
Enter marks 78
Enter marks 89
Enter marks 87
('Roll NO:', 12)
('Name:', 'saranga')
('Subject marks:', [78, 89, 87])
('Percentage: ', 84, '%')
Enter roll no 13
Enter name ketaki
Enter marks 67
Enter marks 78
Enter marks 89
('Roll NO:', 13)
('Name:', 'ketaki')
('Subject marks:', [67, 78, 89])
('Percentage: ', 78, '%')
Enter roll no 15
Enter name abhi
Enter marks 56
Enter marks 67
Enter marks 78
('Roll NO:', 15)
('Name:', 'abhi')
('Subject marks:', [56, 67, 78])
('Percentage: ', 67, '%')
```


Program 3.19: Write a Python program that defines a class named circle with attributes radius and center, where center is a point object and radius is number. Accept center and radius from user. Instantiate a circle object that represents a circle with its center and radius as accepted input.

```
class Circle:
    def _init_(self):
        self.r=raw_input("enter radius of circle:=")
        self.c=raw_input("enter Center of circle:=")

    def _print_(self):
        print("Radius of circle:=",self.r)
        print("Center of circle:=",self.c)

c=Circle()
c._init_()
c._print_()
```

Output:

```
enter radius of circle:=3
enter Center of circle:=4
('Radius of circle:=', '3')
('Center of circle:=', '4')
```

Program 3.20: Python Program to Create a Class which Performs Basic Calculator Operations.

```
class cal():
    def __init__(self,a,b):
        self.a=a
        self.b=b
    def add(self):
        print(self.a+self.b)
    def mul(self):
        print(self.a*self.b)
    def div(self):
        print(self.a/self.b)
    def sub(self):
        print(self.a-self.b)
```

```
a=int(input("Enter first number: "))
b=int(input("Enter second number: "))
obj=cal(a,b)
obj.add()
obj.mul()
obj.div()
obj.sub()
```

Output:

```
Enter first number: 23
Enter second number: 45
68
1035
0
-22
```

Summary

- Python is an object oriented programming language.
- A class is a code template for creating objects. Objects have member variables and have behaviour associated with them. In python a class is created by the keyword `class`.
- An object is also called an instance of a class and the process of creating this object is called instantiation
- The `__init__()` function is called automatically every time the class is being used to create a new object.
- Inheritance, in object-oriented programming, is the ability of a class to inherit members of another class as part of its own definition. The inheriting class is called a subclass (also "derived class" or "child class"), and the class inherited from is called the superclass (also "base class" or "parent class").
- We can inherit a derived class from another derived class, this process is known as multilevel inheritance.
- When a derived class inherits only from syntax, the base class is called single inheritance. If it has one base class and one derived class it is called single inheritance.
- Multi-Level inheritance is possible in python like other object-oriented languages. Multi-level inheritance is archived when a derived class inherits another derived class. There is no limit on the number of levels up to which, the multi-level inheritance is archived in python.
- Multiple inheritance is the concept where a subclass inherits properties from multiple base classes. If it has two base classes and one derived class it is called multilevel inheritance.

- The issubclass(sub, sup) method is used to check the relationships between the specified classes. It returns true if the first class is the subclass of the second class, and false otherwise.
- The isinstance() method is used to check the relationship between the objects and classes. It returns true if the first parameter, i.e., obj is the instance of the second parameter, i.e., class.

Check Your Understanding

Q.I Multiple Choice Questions.

1. Which of the following keywords mark the beginning of the class definition?
   (a) def (b) return
   (c) class (d) cls.
2. Which Of The Following Is Required To Create A New Instance Of The Class?
   (a) constructor (b) A class
   (c) A value-returning method (d) A destructor
3. Which of the following statements can be used to check, whether an object "obj" is an instance of class a or not?
   (a) obj.isinstance(A) (b) A.isinstance(obj)
   (c) isinstance(obj, A) (d) isinstance(A, obj)
4. Without which argument static methods are defined?
   (a) Def (b) Cls
   (c) init (d) Self
5. Which feature in OOP used to specify that one class will get most or all of its features from its parent class.
   (a) Inheritance (b) Overriding
   (c) Overloading (d) Polymorphism
6. Which of the following best describes inheritance?
   (a) Ability of a class to derive members of another class as a part of its own definition
   (b) Means of bundling instance variables and methods in order to restrict access to certain class members
   (c) Focuses on variables and passing of variables to functions
   (d) Allows for implementation of elegant software that is well designed and easily modified
7. Which of the following statements is wrong about inheritance?
   (a) Protected members of a class can be inherited
   (b) The inheriting class is called a subclass
   (c) Private members of a class can be inherited and accessed
   (d) Inheritance is one of the features of OOP

8. What will be the output of the following Python code?

```
class Test:
    def __init__(self):
        self.x = 0
class Derived_Test(Test):
    def __init__(self):
        self.y = 1
def main():
    b = Derived_Test()
    print(b.x,b.y)
main()
```

(a) 0 1
(b) 0 0
(c) Error because class B inherits A but variable x isn't inherited
(d) Error because when object is created, argument must be passed in Derived_Test(1)

9. What will be the output of the following Python code?

```
class A():
    def disp(self):
        print("A disp()")
class B(A):
    pass
obj = B()
obj.disp()
```

(a) Invalid syntax for inheritance
(b) Error because when object is created, argument must be passed
(c) Nothing is printed
(d) A disp()

10. Suppose B is a subclass of A, to invoke the __init__ method in A from B, what is the line of code you should write?
(a) A.__init__(self) (b) B.__init__(self)
(c) A.__init__(B) (d) B.__init__(A)

11. Suppose B is a subclass of A, to invoke the __init__ method in A from B, what is the line of code you should write?
(a) A.__init__(self) (b) B.__init__(self)
(c) A.__init__(B) (d) B.__init__(A)

12. What will be the output of the following Python code?

```
class Test:
    def __init__(self):
        self.x = 0
class Derived_Test(Test):
    def __init__(self):
        Test.__init__(self)
        self.y = 1
def main():
    b = Derived_Test()
    print(b.x,b.y)
main()
```

(a) Error because class B inherits A but variable x isn't inherited
(b) 0 0
(c) 0 1
(d) Error, the syntax of the invoking method is wrong

13. What will be the output of the following Python code?

```
class A:
    def __init__(self, x= 1):
        self.x = x
class der(A):
    def __init__(self,y = 2):
        super().__init__()
        self.y = y
def main():
    obj = der()
    print(obj.x, obj.y)
main()
```

(a) Error, the syntax of the invoking method is wrong
(b) The program runs fine but nothing is printed
(c) 1 0
(d) 1 2

14. Which of the following is not a type of inheritance?
(a) Double-level (b) Multi-level
(c) Single-level (d) Multiple

15. What does built-in function help do in context of classes?
(a) Determines the object name of any value
(b) Determines the class identifiers of any value
(c) Determines class description of any built-in type
(d) Determines class description of any user-defined built-in type

16. What will be the output of the following Python code?

```
class A:
    def one(self):
        return self.two()
```

```
    def two(self):
        return 'A'

class B(A):
    def two(self):
        return 'B'
obj1=A()
obj2=B()
print(obj1.two(),obj2.two())
```

(a) A A (b) A B
(c) B B (d) An exception is thrown

17. What type of inheritance is illustrated in the following Python code?

```
class A():
    pass
class B():
    pass
class C(A,B):
    pass
```

(a) Multi-level inheritance (b) Multiple inheritance
(c) Hierarchical inheritance (d) Single-level inheritance

18. ____ represents an entity in the real world with its identity and behaviour.
(a) A method (b) An object
(c) A class (d) An operator

19. ____ is used to create an object.
(a) class (b) constructor
(c) User-defined functions (d) In-built functions

20. What will be the output of the following Python code?

```
class test:
    def __init__(self,a="Hello World"):
        self.a=a

    def display(self):
        print(self.a)
obj=test()
obj.display()
```

(a) The program has an error because constructor can't have default arguments
(b) Nothing is displayed
(c) "Hello World" is displayed
(d) The program has an error display function doesn't have parameters

21. What is setattr() used for?
(a) To access the attribute of the object (b) To set an attribute
(c) To check if an attribute exists or not (d) To delete an attribute

22. What is getattr() used for?
(a) To access the attribute of the object (b) To delete an attribute
(c) To check if an attribute exists or not (d) To set an attribute

23. What will be the output of the following Python code?

```
class change:
    def __init__(self, x, y, z):
        self.a = x + y + z

x = change(1,2,3)
y = getattr(x, 'a')
setattr(x, 'a', y+1)
print(x.a)
```

(a) 6 (b) 7
(c) Error (d) 0

Answers

1. (c)	2. (a)	3. (c)	4. (d)	5. (a)	6. (a)	7. (c)	8. (c)	9. (d)	10. (a)
11. (a)	12. (c)	13. (d)	14. (a)	15. (c)	16. (b)	17. (b)	18. (b)	19. (b)	20. (c)
21. (b)	22. (a)	23. (b)							

Q.II Write True/False.

1. Python is a procedure oriented Language.
2. The process of creating this object is called instantiation
3. Class variables are defined within the class construction.
4. Class methods needs self as an argument.

Answers

1. False	2. True	3. True	4. False

Trace The Output

1. What will be the output of the following Python code?

```
class test:
    def __init__(self,a="Hello World"):
        self.a=a
    def display(self):
        print(self.a)
obj=test()
obj.display()
```

Output: Hello World

2. What will be the output of the following Python code?

```
class change:
    def __init__(self, x, y, z):
        self.a = x + y + z
```

```
x = change(1,2,3)
y = getattr(x, 'a')
setattr(x, 'a', y+1)
print(x.a)
```

Output: 7

3. What will be the output of the following Python code?

```
class test:
    def __init__(self):
        self.variable = 'Old'
        self.Change(self.variable)
    def Change(self, var):
        var = 'New'
obj=test()
print(obj.variable)
```

Output: Old

4. What will be the output of the following Python code?

```
class fruits:
    def __init__(self, price):
        self.price = price
obj=fruits(50)
obj.quantity=10
obj.bags=2
print(obj.quantity+len(obj.__dict__))
```

Output: 13

Practice Questions

Q.I Answer the following Questions in short:

1. Define the term: class, object.
2. What is Syntax of class?
3. How to create class and objects in python?
4. Write the definition of class method.
5. What is a class variables?

Q.II Answer the following Questions:

1. Write benefits of Inheritance.
2. Python Program to Create a Class which Performs Basic Calculator Operations.

■■■

4... Exception Handling

Learning Objectives ...

- To learn about Concept of Exception.
- To learn about Techniques to Handle Exception.
- To study about try – finally clause.
- To understand Custom Exception and assert statement.

4.1 PYTHON EXCEPTION

- Python has many built-in exceptions that are raised when your program encounters an error (something in the program goes wrong).
- An exception is also called as runtime error that can halt the execution of the program.
- An exception is an error that happens/occurs during execution of a program. When that error occurs, Python generate an exception that can be handled, which avoids the normal flow of the program's instructions.
- When we executes a Python program, there may be a few uncertain conditions which occur, known as errors. Errors also referred to as bugs that are incorrect or inaccurate action that may cause the problems in the running of the program or may interrupt the execution of program.
- There are following three type of error occurs:
  1. **Compile Time Errors:** Occurs at the time of compilation, include due error occur to the violation of syntax rules like missing of a colon (:).
  2. **Run Time Errors:** Occurs during the runtime of a program, example, include error occur due to wrong input submitted to program by user.
  3. **Logical Errors:** Occurs due to wrong logic written in the program.
- Errors occurs at runtime are known as exception. Errors detected during execution of program. Python provides a feature (Exception handling) for handling any unreported errors in program.
- When exception occurs in the program, execution gets terminated. In such cases we ystem generated error message.

4... Exception Handling

Learning Objectives ...

- To learn about Concept of Exception.
- To learn about Techniques to Handle Exception.
- To study about try – finally clause.
- To understand Custom Exception and assert statement.

4.1 PYTHON EXCEPTION

- Python has many built-in exceptions that are raised when your program encounters an error (something in the program goes wrong).
- An exception is also called as runtime error that can halt the execution of the program.
- An exception is an error that happens/occurs during execution of a program. When that error occurs, Python generate an exception that can be handled, which avoids the normal flow of the program's instructions.
- When we executes a Python program, there may be a few uncertain conditions which occur, known as errors. Errors also referred to as bugs that are incorrect or inaccurate action that may cause the problems in the running of the program or may interrupt the execution of program.
- There are following three type of error occurs:
  1. **Compile Time Errors:** Occurs at the time of compilation, include due error occur to the violation of syntax rules like missing of a colon (:).
  2. **Run Time Errors:** Occurs during the runtime of a program, example, include error occur due to wrong input submitted to program by user.
  3. **Logical Errors:** Occurs due to wrong logic written in the program.
- Errors occurs at runtime are known as exception. Errors detected during execution of program. Python provides a feature (Exception handling) for handling any unreported errors in program.
- When exception occurs in the program, execution gets terminated. In such cases we get system generated error message.

- By handling the exceptions, we can provide a meaningful message to the user about the problem rather than system generated error message, which may not be understandable to the user.
- Exception can be either built-in exceptions or user defined exceptions.
- The interpreter or built-in functions can generate the built-in exceptions while user defined exceptions are custom exceptions created by the user.
- **Example:** For exceptions.

```
>>> a=3
>>> if (a<5)
SyntaxError: invalid syntax
>>> 5/0
Traceback (most recent call last):
 File "<pyshell#2>", line 1, in <module>
 5/0
ZeroDivisionError: division by zero
```

- Python provides two very important features to handle any unexpected error in Python programs and to add debugging capabilities in them:
  - Exception Handling
  - Assertions

4.2 COMMON EXCEPTION

Standard Built-in Exceptions:

- Python has many built-in exceptions which forces your program to output an error when something in it goes wrong.
- The following table shows some of Standard built-in Exceptions in Python:

Table 4.1: Build-in Exception

Exception	Cause of Error
ArithmeticError	Base class for all errors that occur for numeric calculation.
AssertionError	Raised in case of failure of the assert statement.
AttributeError	Raised in case of failure of attribute reference or assignment.
Exception	Base class for all exceptions.
EOFError	Raised when there is no input from either the raw_input() or input() function and the end of file is reached.
EnvironmentError	Base class for all exceptions that occur outside the Python environment.
FloatingPointError	Raised when a floating point calculation fails.
ImportError	Raised when an import statement fails.
IndexError	Raised when an index is not found in a sequence.

contd. ...

IOError	Raised when an input/output operation fails, such as the print statement or the open() function when trying to open a file that does not exist.
IndentationError	Raised when indentation is not specified properly.
KeyboardInterrupt	Raised when the user interrupts program execution, usually by pressing Ctrl+c.
KeyError	Raised when the specified key is not found in the dictionary.
LookupError	Base class for all lookup errors.
NameError	Raised when an identifier is not found in the local or global namespace.
NotImplementedError	Raised when an abstract method that needs to be implemented in an inherited class is not actually implemented.
OverflowError	Raised when a calculation exceeds maximum limit for a numeric type.
OSError	Raised for operating system-related errors.
RuntimeError	Raised when a generated error does not fall into any category.
StopIteration	Raised when the next() method of an iterator does not point to any object.
SystemExit	Raised by the sys.exit() function.
StandardError	Base class for all built-in exceptions except StopIteration and SystemExit.
SyntaxError	Raised when there is an error in Python syntax.
SystemError	Raised when the interpreter finds an internal problem, but when this error is encountered the Python interpreter does not exit.
SystemExit	Raised when Python interpreter is quit by using the sys.exit() function. If not handled in the code, causes the interpreter to exit.
TypeError	Raised when an operation or function is attempted that is invalid for the specified data type.
UnboundLocalError	Raised when trying to access a local variable in a function or method but no value has been assigned to it.
ValueError	Raised when the built-in function for a data type has the valid type of arguments, but the arguments have invalid values specified.
ZeroDivisionError	Raised when division or modulo by zero takes place for all numeric types.

4.3 EXCEPTION HANDLING IN PYTHON (TRY-EXCEPT-ELSE)

- An exception is an event, which occurs during the execution of a program that disrupts the normal flow of the program's instructions.
- In general, when a Python script encounters a situation that it cannot cope with, it raises an exception. An exception is a Python object at represents an error.
- For example, if function A calls function B which in turn calls function C and an exception occurs in function C. If it is not handled in C, the exception passes to B and then to A. When a Python script raises an exception, it must either handle the exception immediately otherwise it terminates and quits.
- The exception handling is a process that provides a way to handle exceptions that occur at runtime. The exception handling is done by writing exception handlers in the program.
- The exception handlers are blocks that execute when some exception occurs at runtime. Exception handlers displays same message that represents information about the exception.
- For handling exception in Python, the exception handler block needs to be written which consists of set of statements that need to be executed according to raised exception. There are three blocks that are used in the exception handling process, namely, try, except and finally.
  1. **try Block:** A set of statements that may cause error during runtime are to be written in the try block.
  2. **except Block:** It is written to display the execution details to the user when certain exception occurs in the program. The except block executed only when a certain type exception occurs in the execution of statements written in the try block.
  3. **finally Block:** This is the last block written while writing, an exception handler in the program which indicates the set of statements that are used to clean up the resources used by the program.

Exception Handling using try-except-else:

- In Python, exceptions can be handled using a try statement. A try block consisting of one or more statements is used by programmers to partition code that might be affected by an exception.
- A critical operation which can raise exception is placed inside the try clause and the code that handles exception is written in except clause.
- The associated except blocks are used to handle any resulting exceptions thrown in the try block. If any statement within the try block throws an exception, control immediately shifts to the catch block. If no exceptions is thrown in the try block, the catch block is skipped.
- There can be one or more except blocks. Multiple except blocks with different exception names can be chained together.

- The except blocks are evaluated from top to bottom in the code, but only one except block is executed for each exception that is thrown.
- The first except block that specifies the exact exception name of the thrown exception is executed. If no except block specifies a matching exception name then an except block that does not have an exception name is selected, if one is present in the code.
- **Syntax:**

```
try:
 certain operations here
 ......................
except Exception1:
 If there is Exception1, then execute this block.
except Exception2:
 If there is Exception2, then execute this block.
 ......................
else:
 If there is no exception then execute this block.
```

Program 4.1: For try-except-else statement.

```
n=10
m=0
try:
 n/m
except ZeroDivisionError:
 print("Divide by zero error")
else:
 print (n/m)
```

Output:

```
Divide by zero error
```

4.3.1 The Except Clause with No Exceptions

- If we define except clause with no exceptions, it can handle all types of exceptions. However, neither it's a good coding practice nor it is recommended.
- **Syntax:**

```
try:
 You do your operations here
 ......................
except:
 If there is any exception, then execute this block.
 ......................
else:
 If there is no exception then execute this block.
```

- This kind of a **try-except** statement catches all the exceptions that occur. Using this kind of try-except statement is not considered a good programming practice though, because it catches all exceptions but does not make the programmer identify the root cause of the problem that may occur.

Program 4.2: For try-except statement with no exception.

```
try:
 a=10/0;
except:
 print "Arithmetic Exception"
else:
 print "Successfully Done"
```

Output:

```
Arithmetic Exception
```

Program 4.3: For try-except statement with no exception.

```
while True:
 try:
   a=int(input("Enter an integer: "))
   div=10/a
   break
 except:
   print("Error Occurred")
   print("Please enter valid value")
   print()
print("Division is: ",div)
```

Output:

```
Enter an integer: b
Error Occurred
Please enter valid value
Enter an integer: 2.5
Error Occurred
Please enter valid value
Enter an integer: 0
Error Occurred
Please enter valid value
Enter an integer: 5
Division is: 2.0
```


4.3.2 Multiple Exceptions

- Python allows us to declare multiple exceptions using the same except statement.
- **Syntax:**

```
try:
  You do your operations here
  ......................
except(Exception1[, Exception2[,...ExceptionN]]]):
  If there is any exception from the given exception list,
  then execute this block.
  ......................
else:
  If there is no exception then execute this block.
```

Program 4.4: For Multiple Exceptions.

```
try:
 a=10/0;
except ArithmeticError,StandardError:
 print "Arithmetic Exception"
else:
 print "Successfully Done"
```

Output:

```
Arithmetic Exception
```

4.3.3 raise Statement

- We can raise an existing exception by using raise keyword. So, we just simply write *raise* keyword and then the name of the exception.
- The raise statement allows the programmer to force a specified exception to occur.
  Syntax: `raise [Exception [, argument [, traceback]]]`
- Here, Exception is the type of exception (for example, IOError) and argument is a value for the exception argument. This argument is optional. The exception argument is None if we does not supply any argument. The argument, `traceback`, is also optional. If this is used, then the `traceback` object is used for the exception. This is rarely used in Programming.

Program 4.5: We can use raise to throw an exception if age is less than 18 conditions occurs.

```
age = int(input("Enter your age for election: "))
try:
  if age < 18:
    raise Exception
```

```
    else:
      print("You are eligible for election")
  except Exception:
     print("This value is too small, try again")
```

Output:

```
Enter your age for election: 11
This value is too small, try again
Enter your age for election: 18
You are eligible for election
```

4.4 THE TRY-FINALLY CLAUSE

- Python provides the optional finally statement, which is used with the try statement. It is executed no matter what exception occurs and used to release the external resource. The finally block provides a guarantee of the execution.
- We can use the finally block with the try block in which we can pace the necessary code, which must be executed before the try statement throws an exception.
- **Syntax:**

```
try:
  You do your operations here;
  ......................
  Due to any exception, this may be skipped.
finally:
  This would always be executed.
  ......................
```

- **Note:** You can provide except clause(s), or a finally clause, but not both. You cannot use *else* clause as well along with a finally clause.
- **Example:**

```
try:
  fh = open("testfile", "w")
  fh.write("This is my test file for exception handling!!")
finally:
    print ("Error: can\'t find file or read data")
    fh.close()
```

- If you do not have permission to open the file in writing mode, then this will produce the following result:

```
Error: can't find file or read data
```

- Same example can be written more cleanly as follows:

```
try:
  fh = open("testfile", "w")
```



```
  try:
  fh.write("This is my test file for exception handling!!")
  finally:
  print ("Going to close the file")
  fh.close()
  except IOError:
  print ("Error: can\'t find file or read data")
```

- This produces the following result:

```
Going to close the file:
```

- When an exception is thrown in the try block, the execution immediately passes to the finally block. After all the statements in the finally block are executed, the exception is raised again and is handled in the except statements if present in the next higher layer of the try-except statement.

4.5 CUSTOM EXCEPTION AND ASSERT STATEMENT

4.5.1 Custom Exception/User Defined Exception

- A user can create his own error using exception class. Creating your own Exception class or User Defined Exceptions are known as Custom Exception.
- Python throws errors and exceptions, when there is a code gone wrong, which may cause program to stop abruptly. Python also provides exception handling method with the help of try-except.
- Some of the standard exceptions which are most frequent include IndexError, ImportError, IOError, ZeroDivisionError, TypeError and FileNotFoundError.
- Python also allows you to create your own exceptions by deriving classes from the standard built-in exceptions.
- Here is an example related to RuntimeError. Here, a class is created that is subclassed from RuntimeError. This is useful when you need to display more specific information when an exception is caught.
- In the try block, the user-defined exception is raised and caught in the except block. The variable e is used to create an instance of the class Networkerror.

```
class Networkerror(RuntimeError):
  def __init__(self, arg):
  self.args = arg
```

- So once you have defined the above class, you can raise the exception as follows –

```
try:
  raise Networkerror("Bad hostname")
except Networkerror,e:
  print e.args
```

Program 4.6: A python program to create user-defined exception.

```
# class MyError is derived from super class Exception
class MyError(Exception):
 # Constructor or Initializer
 def __init__(self, value):
 self.value = value
 # __str__ is to print() the value
 def __str__(self):
 return(repr(self.value))
try:
 raise(MyError(3*2))
# Value of Exception is stored in error
except MyError as error:
 print('A New Exception occured: ',error.value)
```

Output:

```
('A New Exception occured: ', 6)
```

- When we are developing a large Python program, it is a good practice to place all the user-defined exceptions that our program raises in a separate file. Many standard modules do this. They define their exceptions separately as `exceptions.py` or `errors.py` (generally but not always).
- User-defined exception class can implement everything a normal class can do, but we generally make them simple and concise. Most implementations declare a custom base class and derive others exception classes from this base class.

4.5.2 Assert Statement

- An assertion is a (debugging) boolean expression to self check an assumption about the internal state of a function or object. If true, nothing happens otherwise, execution halts and an error is thrown.
- For example, while writing a division function, you're confident the divisor shouldn't be zero, you assert divisor is not equal to zero.
- Assertions are carried out by the assert statement.
- Programmers often place assertions at the start of a function to check for valid input, and after a function call to check for valid output.
- Python has built-in assert statement to use assertion condition in the program.

```
assert <condition>
assert <condition>,<error message>
```


Points to be noted:

- Assertions are the condition or boolean expression which are always supposed to be true in the code.
- Assert statement takes an expression and optional message.
- Assert statement is used to check types, values of argument and the output of the function.
- Assert statement is used as debugging tool as it halts the program at the point where an error occurs.

In Python, we can use assert statement in two ways as mentioned above.

1. Assert statement has a condition and if the condition is not satisfied the program will stop and give AssertionError.
2. Assert statement can also have a condition and optional error message. If the condition is not satisfied assert stops the program and gives AssertionError along with the error message.

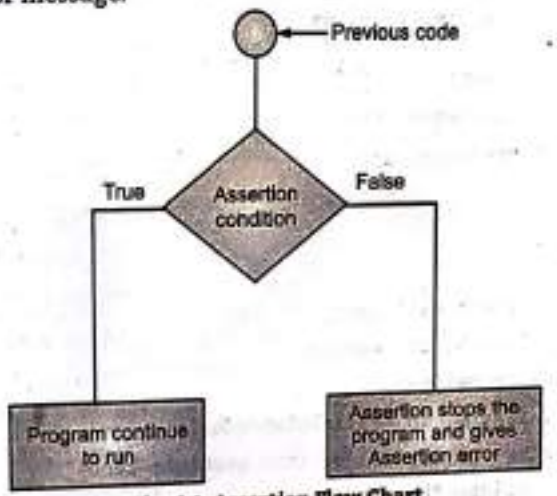


Fig. 4.1: Assertion Flow Chart

Program 4.7: The assert without error message.

```
def avg(marks):
 assert len(marks) != 0
 return sum(marks)/len(marks)
mark1 = []
print("Average of mark1:",avg(mark1))
```

Output:

```
AssertionError
```

Program 4.8: Assertion with error message.

```
def avg(marks):
 assert len(marks) != 0,"List is empty."
 return sum(marks)/len(marks)
```

```
mark2 = [55,88,78,90,79]
print("Average of mark2:",avg(mark2))

mark1 = []
print("Average of mark1:",avg(mark1))
```

Output:

```
Average of mark2: 78.0
AssertionError: List is empty.
```

Program 4.9: A function that converts a given temperature from degrees Kelvin to degrees Fahrenheit. Since 0° K is as cold as it gets, the function bails out if it sees a negative temperature

```
def KelvinToFahrenheit(Temperature):
 assert (Temperature >= 0),"Colder than absolute zero!"
 return ((Temperature-273)*1.8)+32

print (KelvinToFahrenheit(273))
print (int(KelvinToFahrenheit(505.78)))
print (KelvinToFahrenheit(-5))
```

Output:

```
32.0
451
Traceback (most recent call last):
File "test.py", line 9, in <module>
print KelvinToFahrenheit(-5)
File "test.py", line 4, in KelvinToFahrenheit
assert (Temperature >= 0),"Colder than absolute zero!"
AssertionError: Colder than absolute zero!
```

Additional Programs

Program 4.10: Write a program to calculate average of list elements passed by user. element should not be null in list, if its null raise error, using assertion.

```
def listing(l):
 a=len(l)
 assert(a!=0),"List is empty"
 print("Average of list element=",sum(l)/a)
l=[]
n=int(input("enter how many elements you want in list"))
for i in range(n):
 num=int(input("enter element in list"))
```



```
 assert(num!=0),"element should not be zero"
 l.append(num)
print("Entered List is=",l)
listing(l)
```

Output:

```
bash-4.1$ python calcavg.py
enter how many elements you want in list4
enter element in list0
Traceback (most recent call last):
 File "calcavg.py", line 11, in <module>
 assert(num!=0),"element should not be zero"
AssertionError: element should not be zero

bash-4.1$ python calcavg.py
enter how many elements you want in list3
enter element in list1
enter element in list2
enter element in list3
Entered List is= [1, 2, 3]
Average of list element= 2.0
```

Program 4.11: Write a python program which define class to find validity of string parentheses, '(',')','{','}','[',']'. brackets must closed in correct order, if they are valid print valid else raise exception Invalid.

```
class Error1(Exception):
    def __init__(self,str1):
        self.str1=str1
    def print_data(self):
        print(self.str1)
class Validate1:
    def check(self,str2):
        if("()" in str2 or "{}"in str2 or "[]"in str2):
            print("Valid Brackets")
        else:
            raise(Error1("Invalid Bracket"))
n=raw_input("Enter the brackets")
v=Validate1()
v.check(n)
```

Output:

```
bash-4.1$ python brackets.py
Enter the brackets(){}{}
Valid Brackets

bash-4.1$ python brackets.py
Enter the brackets{[)]
Traceback (most recent call last):
 File "brackets.py", line 14, in <module>
---> 14 v.check(n) in check(self, str2)
 9 print("Valid Brackets")
 10 else:
---> 11 raise(Error1("Invalid Bracket"))
Error1: Invalid Bracket
```

Program 4.12: Write a program to convert temperature, fahrenit to kelvin and kelvin to fahrenit using assertion.

```
def temptokf(temp):
 ans=temp*9/5+32
 print(ans,"F")
 ans1=temp*9/5-32
 print(ans1,"K")

temp=int(input("enter tempreature:="))
assert(temp>0),"tempreature should not be negative value or zero"
temptokf(temp)
```

Output:

```
bash-4.1$ python temp.py
enter tempreature:=12
53.6 F
-10.399999999999999 K

bash-4.1$ python temp.py
enter tempreature:=-12
Traceback (most recent call last):
AssertionError
assert(temp>0),"tempreature should not be negative value or zero"
temptokf(temp)
AssertionError: tempreature should not be negative value or zero
```


Program 4.13: Write a text file test.txt that contains int, char, float, numbers. Write a python program to read that test.txt file and print appropriate message using exception handling that line contains char, int or float.

```
fname=input("enter the filename that you want to open:=")
assert(fname!=""),"file name should not be null"
with open (fname,"r") as f:
 for line in f.readlines():
 words=line.split()
 for letters in words:
 if(letters.isdigit()):
 print("Digit Found")
 else:
 print("Character found")
```

Output:

```
bash-4.1$ python files.py
enter the filename that you want to open:=TYBCA.txt
Character found
Digit Found
=>TYBCA.txt file(contents)
tybca 12345
```

Program 4.14: Program to create user-defined exception

```
# class MyError is derived from super class Exception
class MyError(Exception):
    # Constructor or Initializer
    def __init__(self, value):
        self.value = value
    # __str__ is to print() the value
    def __str__(self):
        return(repr(self.value))

try:
    raise(MyError(3*2))
# Value of Exception is stored in error
except MyError as error:
    print('A New Exception occured: ',error.value)
```

Output:

```
A New Exception occured:  6
>>>
```

Summary

- An exception is an error that happens during the execution of a program.
- Error handling is generally resolved by saving the state of execution at the moment the error occurred and interrupting the normal flow of the program to execute a special function or piece of code, which is known as the exception handler.
- Assertions are simply boolean expressions that checks if the conditions return true or not. If it is true, the program does nothing and move to the next line of code. However, if it's false, the program stops and throws an error.
- try Block: A set of statements that may cause error during runtime are to be written in the try block.
- except Block: It is written to display the execution details to the user when certain exception occurs in the program. The except block executed only when a certain type exception occurs in the execution of statements written in the try block.
- finally Block: This is the last block written while writing, an exception handler in the program which indicates the set of statements that are used to clean up the resources used by the program.
- Python throws errors and exceptions, when there is a code gone wrong, which may cause program to stop abruptly. Python also provides exception handling method with the help of try-except.
- Most frequently used standard exceptions are: IndexError, ImportError, IOError, ZeroDivisionError, TypeError and FileNotFoundError.
- A user can create his own error using exception class called a user-defined exception.

Check Your Understanding

Q.I Multiple Choice Questions:

1. When will the else part of try-except-else be executed?
   (a) Always (b) When an exception occurs
   (c) When no exception occurs (d) When an execution occurs into except block
2. Which statement is true for Python exception?
   (a) You cannot create custom exception.
   (b) You can create a user defined exception deriving a class from error class
   (c) You can create a user defined exception deriving a class from Exception class.
   (d) None of the above.
3. Which exception Raised in case of failure of the Assert statement.
   (a) indexError (b) AssertionError
   (c) AbsoluteError (d) keyError

4. Exceptions occurs during _____.
   (a) Run-time (b) Design -time
   (c) Compile time (d) None of the above
5. Which statement is used to show assertion condition in the program?
   (a) Assert (b) Try
   (c) Finally (d) Raise
6. Which block lets you test a block of code for errors?
   (a) try (b) except
   (c) finally (d) None of the above
7. What will be output for the following code?

```
try:
  print(x)
except:
  print(""An exception occurred"")
```

   (a) x (b) An exception occurred
   (c) Error (d) None of the above
8. Which exception raised when a calculation exceeds maximum limit for a numeric type?
   (a) StandardError (b) ArithmeticError
   (c) OverflowError (d) FloatingPointError
9. How many except statements can a try-except block have?
   (a) zero (b) one
   (c) more than one (d) more than zero
11. Is the following Python code valid?

```
try:
  # Do something
except:
  # Do something
finally:
  # Do something
```

   (a) No, there is no such thing as finally
   (b) No, finally cannot be used with except
   (c) No, finally must come before except
   (d) Yes

Answers

1. (c)	2. (c)	3. (b)	4. (a)	5. (a)	6. (a)	7. (b)	8. (c)	9. (d)	10. (b)

Q.II Write True/False:

1. An exception is an object in Python.
2. Exception 'indexerror' raised when an index found in a sequence.
3. Python has assert statement to use assertion condition.
4. A single try statement can have only one except statement.
5. Raising an exception does not beresk current code execution.

Answers

1. True	2. False	3. True	4. False	5. False

Practice Questions

Q.I Answer the following Questions in short:

1. What is Exception in Python?
2. What is assertion?
3. What is the use of try-finally clause.
4. Write syntax of Raise statement and explain it.
5. List out standard built in exceptions.

Q.II Answer the following Questions:

1. Which are built-in exceptions in Python?
2. How to handle exception in Python?
3. Explain Except clause with no exceptions.
4. Explain user defined exceptions.
5. Which are standard expectations?

Q.III Define the terms:

1. Try
2. Except
3. Else
4. Assertion
5. Exception handling
6. Error

■■■

5... GUI Programming

Learning Objectives ...

- To study Tkinter programming.
- To learn about Tkinter widgets.
- To understand about Frame.
- To Study about Button, Label, Entry

5.1 INTRODUCTION OF GUI

- A GUI or a graphical user interface is an interactive environment to take responses from users on various situations such as forms, documents, tests, etc. It provides the user with a good interactive screen than a traditional Command Line Interface (CLI).
- According to the definition of GUI, it is an interface through which a user communicates with electronic devices like computers, mobiles and other devices. This interface utilizes symbols, icons, menus, and other graphics to display information. Related users control the text-based interface, where commands and data are in text form.
- GUI representations can be managed and manipulated by a pointing device such as a mouse in computers and a finger on a touch screen.
- The need for the GUI framework is very justified. Because in the first computer the text interface is created by the keyboard. The command given by the keyboard to initiate responses from a computer needs exact spelling which creates difficulties and inappropriate interface.
- Python helps the programmers to overcome this issue. It has various options for GUI Frameworks for developers.
- Python provides various options for developing Graphical User Interfaces (GUIs). Most important are listed below.
  1. **Tkinter:**
     - Out of all the GUI methods, tkinter is the most commonly used method. It is a standard Python interface to the Tk GUI toolkit shipped with Python.

5... GUI Programming

Learning Objectives ...

- To study Tkinter programming.
- To learn about Tkinter widgets.
- To understand about Frame.
- To Study about Button, Label, Entry

5.1 INTRODUCTION OF GUI

- A GUI or a graphical user interface is an interactive environment to take responses from users on various situations such as forms, documents, tests, etc. It provides the user with a good interactive screen than a traditional Command Line Interface (CLI).
- According to the definition of GUI, it is an interface through which a user communicates with electronic devices like computers, mobiles and other devices. This interface utilizes symbols, icons, menus, and other graphics to display information. Related users control the text-based interface, where commands and data are in text form.
- GUI representations can be managed and manipulated by a pointing device such as a mouse in computers and a finger on a touch screen.
- The need for the GUI framework is very justified. Because in the first computer the text interface is created by the keyboard. The command given by the keyboard to initiate responses from a computer needs exact spelling which creates difficulties and inappropriate interface.
- Python helps the programmers to overcome this issue. It has various options for GUI Frameworks for developers.
- Python provides various options for developing Graphical User Interfaces (GUIs). Most important are listed below.

1. **Tkinter:**
    - Out of all the GUI methods, tkinter is the most commonly used method. It is a standard Python interface to the Tk GUI toolkit shipped with Python.

- Python with tkinter is the fastest and easiest way to create the GUI applications. Creating a GUI using tkinter is an easy task. We will discuss this point in detailed in the same unit.

2. **wxPython:**
   - This is an open-source Python interface for wxWindows. wxPython comes with a richer set of widgets out of the box thantkinter, including trees and HTML viewers. wxPython is Open Source.
   - wxPython is a cross-platform toolkit. This means that the same program will run on multiple platforms without modification. Currently, the Supported platforms are Microsoft Windows, Mac OS X and macOS, and Linux.
3. **PyGUI:**
   - PyGUI is a graphical application cross-platform framework for UNIX, Macintosh and Windows. Compared to some other GUI frameworks, PyGUI is by far the simplest and lightweight of them all, as the API is purely in synchronous with Python.
   - PyGUI inserts very less code between the GUI platform and Python application, hence the display of the application usually displays the natural GUI of the platform.
4. **PyQt:**
   - PyQt is a full featured GUI library and runs portably today on Windows, Mac OS X, and UNIX. It is a Python interface for Qt, one of the most powerful, and popular cross-platform GUI library. PyQt is a blend of Python programming language and the Qt library.
   - Creating a simple GUI application using PyQt involves the following steps:
     1. Import QtGui module.
     2. Create an application object.
     3. A QWidget object creates top level window. Add QLabel object in it.
     4. Set the caption of label as "hello world".
     5. Define the size and position of window by `setGeometry()` method.
     6. Enter the mainloop of application by `app.exec_()` method.

```
import sys
from PyQt4 import QtGui
def window():
 app =QtGui.QApplication(sys.argv)
 w =QtGui.QWidget()
 b =QtGui.QLabel(w)
 b.setText("Hello World!")
 w.setGeometry(100,100,200,50)
 b.move(50,20)
```



```
 w.setWindowTitle("PyQt")
 w.show()
 sys.exit(app.exec_())
if __name__ =='__main__':
 window()
```

- The above code produces the following output:

Fig. 5.1

5. **PyGTK:**
   - A Python interface to GTK, a portable GUI library originally used as the core of the Gnome window system on Linux. PyGTK is part of the GNOME project. It offers comprehensive tools for building desktop applications in Python.
   - GTK+, or the GIMP Toolkit, is a multi-platform toolkit for creating graphical user interfaces. GTK+ has been designed from the ground up to support a wide range of languages. PyGTK is a Python wrapper for GTK+.
   - The underlying GTK+ library provides all kinds of visual elements and utilities for it to develop full-featured applications for the GNOME Desktop. PyGTK is a cross-platform library.
6. **Kivy:**
   - Another GUI framework is Kivy. Kivy is an Open source Python library for the rapid development of applications that make use of innovative user interfaces, such as multi-touch applications.
   - Kivy runs on Linux, Windows, OS X, Android, IOS, and Raspberry Pi. You can run the same code on all supported platforms. It can natively use most inputs, protocols and devices including WM_Touch, WM_Pen, Mac OS X Trackpad and Magic Mouse, Mtdev, Linux Kernel HID. Kivy is 100% free to use, under an MIT license.
7. **PySide:**
   - PySide is a Python binding for the QT side. It is a Python extension or API for QT which is probably industry standards for user interface development for cross-platform. So you can actually run your graphical user interface using PySide in Windows, Mac, and Linux without changing your source code much. That is a great advantage.

Note: Tkinter, Kivy, and wxPython are the free GUI libraries for Python.

5.1.1 Comparison of GUI

Table 5.1: Tkinter vs Kivy

Tkinter	Kivy
Tkinter is rock solid with some cross-platform characteristics.	The aim is to permit fast and simple interaction design.
It's extra flexible and consistent.	These include the multi-touch functionalities that are essential for mobile apps.
It is mature stable and widely ported.	Along with rapid prototyping, reusable code, and deployable functionalities.
Tkinter is a simple API and it is easy to learn.	Kivy is greatly suited for touch interfaces and games.
Tkinter use native widgets on the mac and windows.	Kivy looks attractive sweet in terms of porting to all major platforms and supporting touch screens.
Tkinter has a very simple syntax.	Kivy is designed in python based on OpenGL.

Table 5.2: PyQt vs PySide

PySide	PyQT
PySide is Lesser General Public License.	PyQT is General Public License.
PySide is developed by Nokia.	PyQT by Riverbank computing.
PySide has documentation extra intuitive.	PyQt also maintains this latest API.
PySide you just have to replace the import line in the beginning.	PyQt has the benefit of Python 3 help and incumbency.
PySide gives a permit application in commercial projects.	PyQt represents a set of Python V2 and V3 connections for the Qt application framework.
PySide utilizes various library names than PyQt.	It can be used on all platforms that support QT, such as Windows, OS X, Linux, IOS, and android.

Table 5.3: Kivy Vs PyQT

Kivy	PyQt
Kivy describes an open source Python library created for the fast development of applications that need innovative user interfaces.	PyQt represents a set of Python V2 and v3 connections for the QT application framework.

contd...

Kivy works smoothly on Linux, Windows, OS X Android, and Raspberry Pi.	It is mostly linked with PyQt5 to Qt 5, while PyQt4 is supported by Qt v4, and developed in close relation with Qt v5.
Kivy language is excellent for offering the syntax of the Kivy program.	PyQt relies on bindings, which are set in place through a set of Python modules of extra than 1,000 classes.
Kivy included classes, inherited classes, and widget configuration for better represent functionalities.	Most companies that develop apps rely extra on PyQt, as this offers a wide array of functionalities and tools for adaptability.
You will need to mix py and kv languages for access complex applications with the aid of specific algorithms.	Yet due to the vast knowledge and steep learning curve requirement, most of the programs turn their attention for user-friendly Kivy Programming.
Kivy is a python library meant to be used for establishing multitouch-enabled media-rich apps.	Along with PyQt, these two are labeled as full-stack frameworks and cross-platform mobile development software.

Table 5.4: WxPython vs Tkinter

WxPython	Tkinter
The WxPython version has 76 lines.	The Tkinter version has 48 lines.
The WxPython pattern utilizes nested HBOX and VBOX sizers.	The Tkinter pattern utilizes a grid layout.
The WxPython externalizes the layout classes in its sizer hierarchy.	Tkinter internalizes layout.
WxPython takes 6 seconds of load time.	Tkinter takes 1 second of load time.
WxPython is use in commercial products as well as in freeware or shareware.	Tkinter you will have to develop your GUI through typing code.
This is a free and open source.	Tkinter is easy to learn.

5.2 Tkinter PROGRAMMING

- Tkinter is the standard GUI library for Python. Python combined with Tkinter provides a fast and easy way to create GUI applications. Tkinter provides a powerful object-oriented interface to the Tk GUI toolkit.
- Creating a GUI application using Tkinter is an easy task. All you need to do is perform the following steps:
  1. Import the Tkinter module.

```
import tkinter
or
from tkinter import*
```

2. Create the GUI application main window.
3. Add one or more of widgets to the GUI application.
4. Enter the main event loop (`mainloop()`) to take action against each event triggered by the user.

- Let's build a very simple GUI with the help of Tkinter and understand it with the help of a flow diagram.

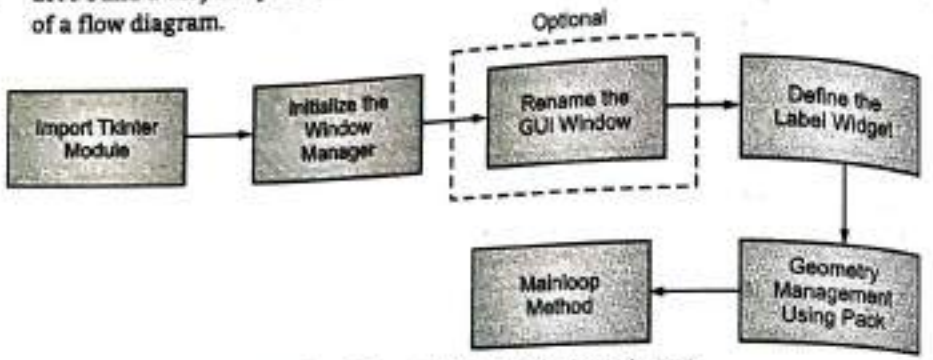


Fig. 5.2: Flow diagram for rendering a Basic GUI

Flow Diagram Explanation:

1. First, you import the key component, i.e., the `Tkinter` module.
2. Next, you initialize the window manager with the `tkinter.Tk()` method and assign it to a variable. This method creates a blank window with close, maximize, and minimize buttons on the top as a usual GUI should have.
3. Then as an optional step, you will rename the title of the window as you like with `window.title(title_of_the_window)`.
4. Next, you make use of a widget called `Label`, which is used to insert some text into the window.
5. Then, you make use of Tkinter's geometry management attribute called `pack()` to display the widget in size it requires.
6. Finally, as the last step, you use the `mainloop()` method to display the window until you manually close it. It runs an infinite loop in the backend.

- The `mainloop()` method called last puts the label on the screen and enters a Tkinter wait state, which watches for user-generated GUI events. Within the `mainloop()` function, Tkinter internally monitors things such as the keyboard and mouse to detect user generated events.
- Python/Tkinter programs are entirely *event driven*: they build displays and register handlers for events, and then do nothing but wait for events to occur. During the wait, the Tk GUI library runs an event loop that watches for mouse clicks, keyboard presses, and so on. All application program processing happens in the registered callback handlers in response to events.

Advantages:

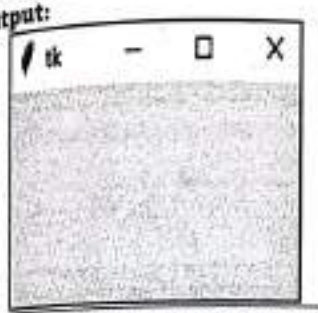
1. If you're using a recent version of Python, Tkinter is most likely already installed

2. Tkinter offers a vast collection of well known widgets, including all the most common ones like buttons, labels, checkboxes etc.
3. Adding code to each widget is straightforward.

Program 5.1: Program for Tkinter.

```
import tkinter
top = tkinter.Tk()
# Code to add widgets will go here...
top.mainloop()
```

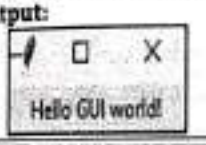
Output:



Program 5.2: Program to insert label.

```
from tkinter import Label                          # get a widget object
widget = Label(None, text='Hello GUI world!')      # make the instance
widget.pack()                                      # arrange it
widget.mainloop()                                  # start event loop
```

Output:



5.3 TKINTER WIDGETS

- Widgets are similar to elements in HTML. You will find different types of widgets for different types of elements in the Tkinter. They are standard GUI elements.
- Tkinter provides various controls, such as buttons, labels and text boxes used in a GUI application. These controls are commonly called widgets.
- The following are the widgets in Tkinter and their brief description is shown in the following table:

Table 5.5: Tkinter Widgets

Widgets	Description
`Button`	The Button widget is used to display buttons in your application.
`Canvas`	The Canvas widget is used to draw shapes, such as lines, ovals, polygons and rectangles, in your application.

contd. ...

Checkbutton	The Checkbutton widget is used to display a number of options as checkboxes. The user can select multiple options at a time.
Entry	The Entry widget is used to display a single-line text field for accepting values from a user.
Frame	The Frame widget is used as a container widget to organize other widgets.
Label	The Label widget is used to provide a single-line caption for other widgets. It can also contain images.
Listbox	The Listbox widget is used to provide a list of options to a user.
Menubutton	The Menubutton widget is used to display menus in your application.
Menu	The Menu widget is used to provide various commands to a user. These commands are contained inside Menubutton.
Message	The Message widget is used to display multiline text fields for accepting values from a user.
Radiobutton	The Radiobutton widget is used to display a number of options as radio buttons. The user can select only one option at a time.
Scale	The Scale widget is used to provide a slider widget.
Scrollbar	The Scrollbar widget is used to add scrolling capability to various widgets, such as list boxes.
Text	The Text widget is used to display text in multiple lines.
Toplevel	The Toplevel widget is used to provide a separate window container.
Spinbox	The Spinbox widget is a variant of the standard Tkinter Entry widget, which can be used to select from a fixed number of values.
PanedWindow	A PanedWindow is a container widget that may contain any number of panes, arranged horizontally or vertically.
LabelFrame	A labelframe is a simple container widget. Its primary purpose is to act as a spacer or container for complex window layouts.
tkMessageBox	This module is used to display message boxes in your applications.

Geometry Management:

- Creating a new widget doesn't mean that it will appear on the screen. To display it, we need to call a special method: either grid, pack (example above), or place.
  - **The pack():** This method manages the geometry of widgets in blocks.
  - **The grid():** This method organizes widgets in a tabular structure.
  - **The place():** This method organizes the widgets to place them in a specific position.

Making Widgets:

- When widgets are constructed in Tkinter, we can specify how they should be configured.
- The GUI script passes two arguments to the Label class constructor: (refer Program 5.2).

1. The first is a parent-widget object. which we want the new label to be attached to. Here, None means "attach the new Label to the default top-level window of this program." Later, we'll pass real widgets in this position to attach our labels to other container objects.
2. The second is a configuration option for the Label, passed as a keyword argument: the text option specifies a text string to appear as the label's message. Most widget constructors accept multiple keyword arguments for specifying a variety of options (color, size, callback handlers, and so on). Most widget configuration options have reasonable defaults per platform, though, and this accounts for much of Tkinter's simplicity.
3. In example 5.3, The Tkinter mainloop can be called with or without a widget (i.e., as a function or method). We didn't pass Label a parent argument in this version, either: it simply defaults to None when omitted.

Example 5.3: Program without a widget.

```
from tkinter import *
Label(text='Hello GUI world!').pack()
mainloop()
```

Output:

Widget Resizing:

- Widget resizing is done by using pack method by two options: **expand and fill.**
- When widgets are packed, we can specify whether a widget should expand to take up all available space by using **expand=YES option.**
- Asks the packer to expand the allocated space for the widget in general into any unclaimed space in the widget's parent by using fill option.
- Fill option can be used to stretch the widget to occupy all of its allocated space.

Program 5.4: Program for Resizing window.

```
from tkinter import *
Label(text='Hello GUI world!').pack(expand=YES, fill=BOTH)
mainloop()
```

Output:

Configuring Widget Options and Window Titles:

- There are two other ways to specify widget configuration options. In Program 5.5, the text option of the label is set after it is constructed, by assigning to the widget's text key.
- Widget objects overload (intercept) index operations such that options are also available as mapping keys, much like a dictionary.
- The config method (which can also be called by its synonym, configure) can be called at any time after construction to change the appearance of a widget on the fly.

Program 5.5: Program for GUI with expansion and window title.

```
from tkinter import *
root = Tk()
widget = Label(root)
widget.config(text='Hello GUI world!')
widget.pack(side=TOP, expand=YES, fill=BOTH)
root.title('gui1g.py')
root.mainloop()
```

Output:

GUI with expansion and window title

5.4 FRAME

- Frame is used as containers in the Tkinter for grouping and adequately organizing the widgets.
- **Syntax:**

```
frame_      = tk.Frame(widget, option=placeholder)
```


Where, widget is the parameter for the parent window/frame while option is a placeholder that can have various values like border-width, height and width of widget, highlightcolor (color when widget has to be focused).

- To arrange the layout in the window, you will use a Frame widget class. Let's create a simple program to see how the Frame works.
- We will define two frames - top and bottom with the help of the pack class.
- The Frame class will help in creating a division between the window. Basically, a single-window will be replicated twice as top and bottom in the form of a Frame.
- Following are the various options used with Tkinter frame widgets:

Table 5.1: Frame Widget Options

Option	Description
bg	The normal background color displayed behind the label and indicator.
bd	The size of the border around the indicator. Default is 2 pixels.
cursor	If you set this option to a cursor name (arrow, dot etc.), the mouse cursor will change to that pattern when it is over the checkbutton.
height	The vertical dimension of the new frame.
highlightbackground	Color of the focus highlight when the frame does not have focus.
highlightcolor	Color shown in the focus highlight when the frame has the focus.
highlightthickness	Thickness of the focus highlight.
relief	With the default value, relief=FLAT, the checkbutton does not stand out from its background. You may set this option to any of the other styles
width	The default width of a checkbutton is determined by the size of the displayed image or text. You can set this option to a number of characters and the checkbutton will always have room for that many characters.

Program 5.6: Diplay the massage after clicking on the button.

```
importtkinterastk

classApplication(tk.Frame):
def__init__(self,master=None):
super().__init__(master)
self.master=master
self.pack()
self.create_widgets()
```

```
defcreate_widgets(self):
self.hi_there=tk.Button(self)
self.hi_there["text"]="Hello World\n(click me)"
self.hi_there["command"]=self.say_hi
self.hi_there.pack(side="top")

self.quit=tk.Button(self,text="QUIT",fg="red",
command=self.master.destroy)
self.quit.pack(side="bottom")

defsay_hi(self):
print("hi there, everyone!")

root=tk.Tk()
app=Application(master=root)
app.mainloop()
```

Output in Pycharm IDLE:

- When we click on button, the msg hi there, everyone! will be display on the console.also when we click on Quit, the window disappears.

5.5 BUTTON

- Button widget has a property for switching on/off. When a user clicks the button, an event is triggered in the Tkinter.
- **Syntax:** `button_widget = tk.Button(widget, option=placeholder)`
- Where, `widget` is the argument for the parent window/frame while `option` is a `placeholder` that can have various values like foreground and background color, font command (for function call), image, height, and width of button.

- **Button Widget Options:** Following are the various options used with Tkinter button widgets:

Table 5.2: Button Widget Options

Option name	Description
activebackground	This option indicates the background of the button at the time when the mouse hovers the button.
bd	This option is used to represent the width of the border in pixels.
bg	This option is used to represent the background color of the button.
command	The command option is used to set the function call which is scheduled at the time when the function is called.
activeforeground	This option mainly represents the font color of the button when the mouse hovers the button.
fg	This option represents the foreground color of the button.
font	This option indicates the font of the button.
height	This option indicates the height of the button. This height indicates the number of text lines in the case of text lines and it indicates the number of pixels in the case of images.
image	This option indicates the image displayed on the button.
higlightcolor	This option indicates the highlight color when there is a focus on the button
justify	This option is used to indicate the way by which the multiple text lines are represented. For left justification, it is set to LEFT and it is set to RIGHT for the right justification, and CENTER for the center justification.
padx	This option indicates the additional padding of the button in the horizontal direction.
pady	This option indicates the additional padding of the button in the vertical direction.
underline	This option is used to underline the text of the button.
width	This option specifies the width of the button. For textual buttons, it exists as a number of letters or for image buttons it indicates the pixels.
Wraplength	In the case, if this option's value is set to a positive number, the text lines will be wrapped in order to fit within this length.
state	This option's value set to DISABLED to make the button unresponsive. The ACTIVE mainly represents the active state of the button.

Program 5.7: The following code line creates a Button with a yellow background and blue text. It also sets the width and height to 25 and 5 text units, respectively.

```
button=tk.Button
{
    text="Click me!",
    width=25,
    height=5,
    bg="yellow",
    fg="blue",
}
```

Program 5.8: Python application to create a simple button.

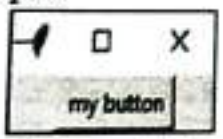
```
from tkinter import *
top = Tk()
top.geometry("200x100")
b = Button(top, text="Simple")
b.pack()
top.mainloop()
```

Output:

Program 5.9: Program to display button

```
Import sys
from tkinter import *
widget = Button(None, text='my button', command=sys.exit)
widget.pack()
widget.mainloop()
```

Output:



A button on the top

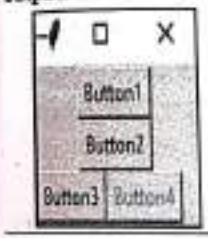
Program 5.10: Program to add buttons in the different frames

```
import sys
import tkinter
```



```
# Let's create the Tkinter window.
window = tkinter.Tk()
window.title("GUI")

# You will first create a division with the help of Frame class.
# And align them on TOP and BOTTOM with pack() method.
top_frame = tkinter.Frame(window).pack()
bottom_frame = tkinter.Frame(window).pack(side = "bottom")

# Once the frames are created.
# then you are all set to add widgets in both the frames.
btn1 = tkinter.Button(top_frame, text = "Button1", fg = "red").pack()
#'fg or foreground' is for coloring the contents (buttons).
btn2 = tkinter.Button(top_frame, text = "Button2", fg = "green").pack()
btn3 = tkinter.Button(bottom_frame, text = "Button3", fg =
"purple").pack(side = "left")
#'side' is used to left or right align the widgets.
btn4 = tkinter.Button(bottom_frame, text = "Button4", fg =
"orange").pack(side = "left")
window.mainloop()
```

Output:



Program 5.11: Program using Button with event.

```
import sys
from tkinter import *
def hello(event):
print('Press twice to exit') # on single-left click
def quit(event): # on double-left click
print('Hello, I must be going...') # event gives widget, x/y, etc.
sys.exit()
widget = Button(None, text='Hello event world')
widget.pack()
widget.bind('<Button-1>', hello) # bind left mouse clicks
widget.bind('<Double-1>', quit) # bind double-left clicks
widget.mainloop()
```

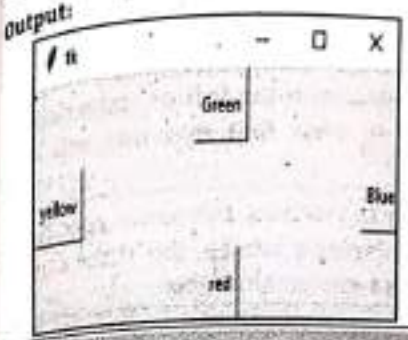
Output:

- Here, on one click the msg displayed is "Press twice to exit" and on two clicks, the window is disappeared with msg "Hello, I must be going..".

Program 5.12: Tkinter Button Widget - Add style and Event handler.

```
Import sys
import tkinter
from tkinter import *
from tkinter import messagebox

top = Tk()
top.geometry("300x150")
def click():
    messagebox.showinfo("Hello", "Green Button clicked")
a = Button(top, text="yellow", activeforeground="yellow",
                                    activebackground="orange", pady=10)
b = Button(top, text="Blue", activeforeground="blue",
                                    activebackground="orange", pady=10)
# adding click function to the below button
c = Button(top, text="Green", command=click, activeforeground = "green",
                                    activebackground="orange", pady=10)
d = Button(top, text="red", activeforeground="yellow",
                                    activebackground="orange", pady=10)

a.pack(side = LEFT)
b.pack(side = RIGHT)
c.pack(side = TOP)
d.pack(side = BOTTOM)
top.mainloop()
```


Output:



5.6 LABEL

- Label is used to create a single line widgets like text, images, etc.
- Label widgets display text with the default system text color and the default system text background color. These are typically black and white, respectively, but you may see different colors if you have changed these settings in your operating system.
- You can control Label text and background colors using the foreground and background parameters.

  Syntax: `label_widget = tk.Label(widget, option=placeholder)`
- Where, `widget` is the parameter for the parent window/frame while `option` is a `placeholder` that can have various values like the font of a button, background color, image, width, and height, etc.

  Example:

  1. `label=tk.Label(text="Hello, Tkinter")`
  2. `label=tk.Label(text="Hello, Tkinter",fg="white",bg="black")`

Label Widget Options:

- Following are the various options used with Tkinter label widgets:

Table 5.3: Label Widget Options

Option	Description
anchor	This options controls where the text is positioned if the widget has more space than the text needs. The default is anchor=CENTER, which centers the text in the available space.
bg	The normal background color displayed behind the label and indicator.
bitmap	Set this option equal to a bitmap or image object and the label will display that graphic.
bd	The size of the border around the indicator. Default is 2 pixels.

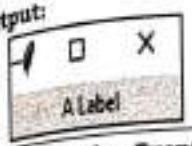
contd. ...

cursor	If you set this option to a cursor name (arrow, dot etc.), the mouse cursor will change to that pattern when it is over the checkbutton.
font	If you are displaying text in this label (with the text or textvariable option, the font option specifies in what font that text will be displayed.
fg	If you are displaying text or a bitmap in this label, this option specifies the color of the text. If you are displaying a bitmap, this is the color that will appear at the position of the 1-bits in the bitmap.
height	The vertical dimension of the new frame.
image	To display a static image in the label widget, set this option to an image object.
justify	Specifies how multiple lines of text will be aligned with respect to each other: LEFT for flush left, CENTER for centered (the default), or RIGHT for right-justified.
padx	Extra space added to the left and right of the text within the widget. Default is 1.
pady	Extra space added above and below the text within the widget. Default is 1.
relief	Specifies the appearance of a decorative border around the label. The default is FLAT; for other values.
text	To display one or more lines of text in a label widget, set this option to a string containing the text. Internal newlines ("\n") will force a line break.
textvariable	To slave the text displayed in a label widget to a control variable of class StringVar, set this option to that variable.
underline	You can display an underline (_) below the nth letter of the text, counting from 0, by setting this option to n. The default is underline=-1, which means no underlining.
width	Width of the label in characters (not pixels!). If this option is not set, the label will be sized to fit its contents.
wraplength	You can limit the number of characters in each line by setting this option to the desired number. The default value, 0, means that lines will be broken only at newlines.

Program 5.13: Program for Label widget.

```
import tkinter
parent_widget = tkinter.Tk()
label_widget = tkinter.Label(parent_widget, text="A Label")
label_widget.pack()
tkinter.mainloop()
```

Output:



Mouse Clicking Event via the Bind Method:

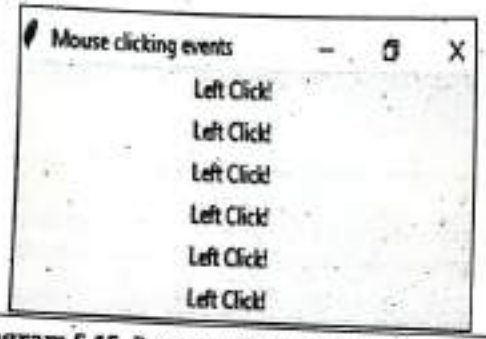
- The Bind method provides you with a very simplistic approach to implementing the mouse clicking events. There are three pre-defined functions which can be used out of the box with the bind method.
- Clicking events are of three types `leftClick`, `middleClick`, and `rightClick`.
  - <Button-1> parameter of the bind method is the left-clicking event, i.e., when you click the left button, the bind method will call the function specified as a second parameter to it.
  - <Button-2> for middle-click
  - <Button-3> for right-click
- To call a particular function based on the event that occurs will be discussed below:
  - Run the following program and click the left, middle, right buttons to call a specific function.
  - That function will create a new label with the specified text.

Program 5.14: Create a GUI which can capture the events that have been fired from the mouse. The mouse has right click, left click and middle click. So whenever the user will click the mouse button it will capture the event and write it on the GUI main window.

```
import tkinter
# Let's create the Tkinter window
window = tkinter.Tk()
window.title("Mouse clicking events")
#You will create three different functions for three different events
def left_click(event):
 tkinter.Label(window, text = "Left Click!").pack()

def middle_click(event):
 tkinter.Label(window, text = "Middle Click!").pack()

def right_click(event):
 tkinter.Label(window, text = "Right Click!").pack()
```

```
window.bind("<Button-1>", left_click)
window.bind("<Button-2>", middle_click)
window.bind("<Button-3>", right_click)

window.mainloop()
```

Output:



Program 5.15: Program of event handling on button.

```
import tkinter
# Let's create the Tkinter window
window = tkinter.Tk()
window.title("Mouse clicking events")
top_frame = tkinter.Frame(window).pack()
bottom_frame = tkinter.Frame(window).pack(side="bottom")
#'fg or foreground' is for coloring the contents (buttons).
btn1 = tkinter.Button(top_frame, text = "Button1", fg = "red").pack()
btn2 = tkinter.Button(top_frame, text = "Button2", fg = "green").pack()
btn3 = tkinter.Button(bottom_frame, text = "Button3", fg = "purple").pack()
#three different functions creates for three different events.
def left_click(event):
 tkinter.Label(window, text = "Left Click!").pack()

def middle_click(event):
 tkinter.Label(window, text = "Middle Click!").pack()

def right_click(event):
 tkinter.Label(window, text = "Right Click!").pack()
window.bind("<Button-1>", left_click)
window.bind("<Button-2>", middle_click)
window.bind("<Button-3>", right_click)
window.mainloop()
```


Output:

When we pressed on Button 1, the msg "Left Clicked" is displayed.

Adding Multiple Widgets:

- Following Program makes a Frame widget and attaches three other widget objects to it, a Label and two Buttons, by passing the Frame as their first argument.
- In tkinter terms, we say that the Frame becomes a parent to the other three widgets. Both buttons on this display trigger callbacks:
  - Pressing the Hello button triggers the greeting function defined within this file, which prints to `Hello stdout world!...again.`
  - Pressing the Quit button calls the standard tkinter quit method, inherited by win from the Frame class (Frame.quit has the same effect as the Tk.quit we used earlier).

Program 5.16: Program with button and label.

```
from tkinter import *
def greeting():
print('Hello stdout world!...')
win = Frame()
win.pack()
Label(win, text='Hello container world').pack(side=TOP)
Button(win, text='Hello', command=greeting).pack(side=LEFT)
Button(win, text='Quit', command=win.quit).pack(side=RIGHT)
win.mainloop()
```

Output:

On terminal we get the output as:

```
C:/Users/HP/PycharmProjects/pythonProject1/gui2.py
Hello stdout world!...
Process finished with exit code 0
```

5.7 ENTRY

- The Entry widget is a simple, single-line text input field. It is typically used for input fields in form-like dialogs and anywhere else you need the user to type a value into a field of a larger display. Entry also supports advanced concepts such as scrolling, key bindings for editing, and text selections.
- **Syntax:**

```
entry_widget = tk.Entry(widget, option=placeholder)
```

Where, widget is the parameter for the parent window/frame while option is a placeholder that can have various values like border-width, background color, width & height etc.

- When you need to get a little bit of text from a user, like a name or an email address, use an Entry widget. They display a *small text box* that the user can type some text into it. Creating and styling an Entry widget works like Label and Button widgets.
- For example, the following line creates a widget with a blue background, some yellow text, and a width of 50 text units:

```
entry=tk.Entry(fg="yellow",bg="blue",width=50)
```

- Entry widgets is not for how to style, but, it is for how to use them to *get input from a user*. There are three main operations that you can perform with Entry widgets:
  1. Retrieving text **with .get()**
  2. Deleting text **with .delete()**
  3. Inserting text **with .insert()**
- The following example shows how to create Entry widgets and interact with it. First, import tkinter and create a new window on python shell.

```
>>>import tkinter *
>>>window=tk.Tk()
```

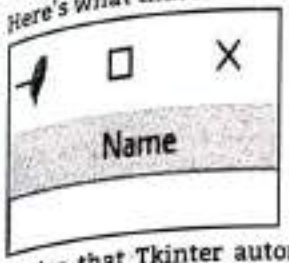
- Now create a Label and an Entry widget:

```
>>>label=tk.Label(text="Name")
>>>entry=tk.Entry()
```

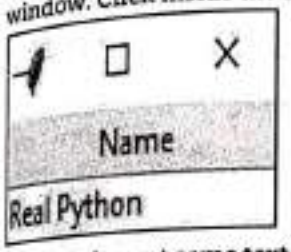
- The Label describes what sort of text should go in the Entry widget. It doesn't enforce any sort of requirements on the Entry, but it tells the user what your program expects them to put there. You need to `.pack()` the widgets into the window so that they're visible:

```
>>>label.pack()
>>>entry.pack()
```


- Here's what that looks like:



- Notice that Tkinter automatically centers the Label above the Entry widget in the window. Click inside the Entry widget with your mouse and type `"Real Python"`:



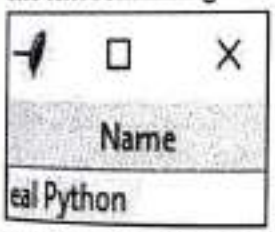
- Now you've got some text entered into the Entry widget, but that text hasn't been sent to your program yet. You can use `.get()` to retrieve the text and assign it to a variable called name:

```
>>>name=entry.get()
>>>name
'Real Python'
```

- You can `.delete()` text as well. This method takes an integer argument that tells Python which character to remove. For example, the code block below shows how .delete(0) deletes the first character from the Entry:

```
>>>entry.delete(0)
```

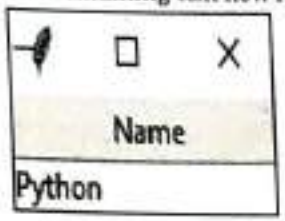
- The text remaining in the widget is now `"eal Python"`:



- Note that, just like Python string objects, text in an Entry widget is indexed starting with 0.
- If you need to remove several characters from an Entry, then pass a second integer argument to `.delete()` indicating the index of the character where deletion should stop. For example, the following code deletes the first four letters in the Entry:

```
>>>entry.delete(0,4)
```

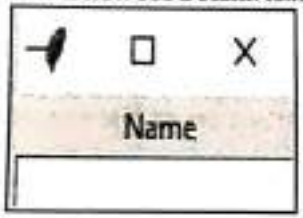
- The remaining text now reads "Python":



- `entry.delete()` works just like string slicing. The first argument determines the starting index, and the deletion continues up to but not including the index passed as the second argument. Use the special constant `tk.END` for the second argument of `.delete()` to remove all text in an Entry:

```
>>>entry.delete(0,tk.END)
```

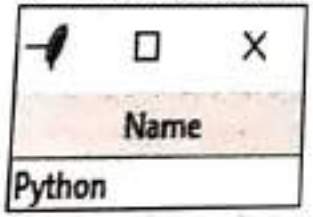
- You'll now see a blank text box:



- On the opposite end of the spectrum, you can also `.insert()` text into an Entry widget:

```
>>>entry.insert(0,"Python")
```

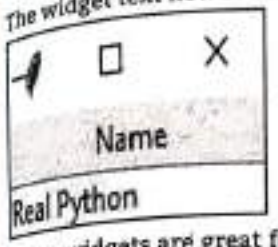
- The window now looks like this:



- The first argument tells `.insert()` where to insert the text. If there is no text in the Entry, then the new text will always be inserted at the beginning of the widget, no matter what value you pass as the first argument. For example, calling `.insert()` with 100 as the first argument instead of 0, as you did above, would have generated the same output.
- If an Entry already contains some text, then .insert() will insert the new text at the specified position and shift all existing text to the right:

```
>>>entry.insert(0,"Real ")
```


- The widget text now reads "Real Python":



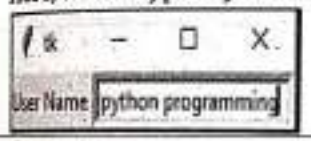
- Entry widgets are great for capturing small amounts of text from a user, but because they're only displayed on a single line, they're not ideal for gathering large amounts of text.

Program 5.17: Program for entry widget.

```
from tkinter import *
top = Tk()
L1 = Label(top, text="User Name")
L1.pack( side = LEFT)
E1 = Entry(top, bd =5)
E1.pack(side = RIGHT)
top.mainloop()
```

Output:

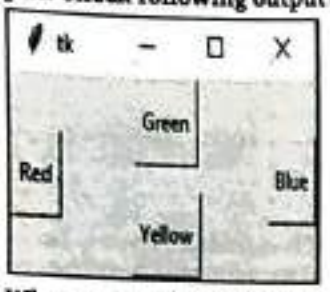
Here, we can type any text in the input box, say "python programming".



Program 5.18: Program for displaying background.

```
from tkinter import *
top = Tk()
top.geometry("200x100")
def fun():
 messagebox.showinfo("Hello", "Red Button clicked")
b1 = Button(top, text="Red", command=fun, activeforeground="red",
activebackground="pink", pady=10)
b2 = Button(top, text="Blue",
                    activeforeground="blue",activebackground="pink",pady=10)
b3 = Button(top, text="Green", activeforeground="green",
                                                activebackground="pink",pady=10)
b4 = Button(top,text="Yellow",activeforeground="yellow",
                                                activebackground="pink",pady=10)
b1.pack(side=LEFT)
b2.pack(side=RIGHT)
b3.pack(side=TOP)
b4.pack(side=BOTTOM)
top.mainloop()
```

Output: Check following output comment



When Red button is clicked, the following window is displayed.

5.6 CONFIGURING WIDGET APPEARANCE

- The following example introduces some of the configuration options available in tkinter.

Program 5.19

```
from tkinter import *
root = Tk()
labelfont = ('times', 20, 'bold') # family, size, style
widget = Label(root, text='Hello config world')
widget.config(bg='black', fg='yellow') # yellow text on black label
widget.config(font=labelfont) # use a larger font
widget.config(height=3, width=20) # initial size: lines,chars
widget.pack(expand=YES, fill=BOTH)
root.mainloop()
```

Output:

Additional Programs

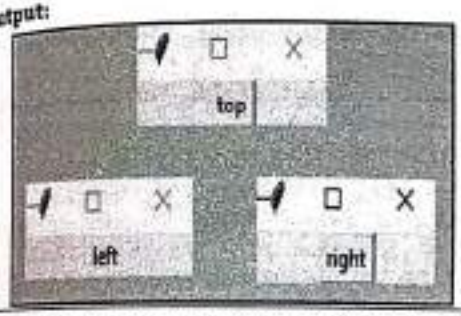
1. **Multiple windows**

- The two Toplevel windows on the right are full-fledged windows; they can be independently iconified, maximized, and so on. Toplevels are typically used to implement multiple-window displays and pop-up modal and nonmodal dialogs.

Program 5.20: Program for multiple windows.

```
import sys
from tkinter import Toplevel, Button, Label
win1 = Toplevel() # two independent windows
win2 = Toplevel() # but part of same process
Button(win1, text='top', command=sys.exit).pack()
Button(win2, text='right', command=sys.exit).pack()
Label(text='left').pack() # on default Tk() root window
win1.mainloop()
```

Output:



Program 5.21: Program for Creating a Calculator with Tkinter.

```
from tkinter import *
# Let's create the Tkinter window
window = Tk()

# You will define the size of the window in width and height, using the
geometry method
window.geometry("312x324")

# Prevent the window from getting resized you will call 'resizable' method
on the window.
window.resizable(0, 0)

#Finally, define the title of the window.
window.title("Calcualtor")

# Let's now define the required functions for the Calculator to function
properly.
```

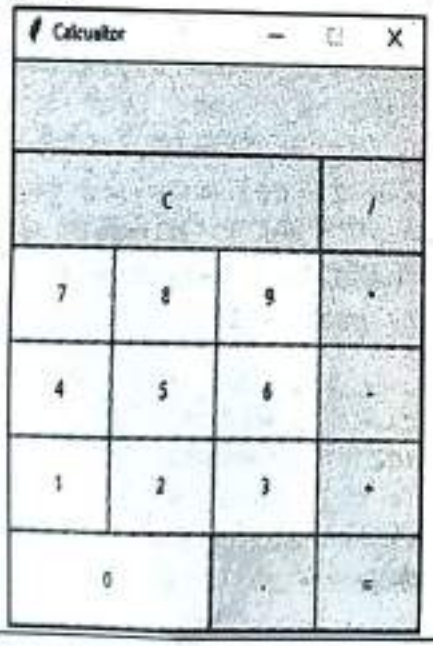
```
# 1. First is the button click 'btn_click' function which will continuously update the
# input field whenever a number is entered or any button is pressed it will act as a
# button click update.
def btn_click(item):
global expression
 expression = expression + str(item)
 input_text.set(expression)

# 2. Second is the button clear 'btn_clear' function clears the input field or
# previous calculations using the button "C"
def btn_clear():
global expression
 expression = ""
input_text.set("")

# 3. Third and the final function is button equal ("=") 'btn_equal' function which
# will calculate the expression present in input field. For example: User clicks
# button 2, + and 3 then clicks "=" will result in an output 5.
def btn_equal():
global expression
 result = str(eval(expression))
# 'eval' function is used for evaluating the string expressions directly
# you also implement your own function to evalute the expression istead of 'eval'
# function
input_text.set(result)
 expression = ""

expression = ""
# In order to get the instance of the input field 'StringVar()' is used
input_text = StringVar()

# Once all the functions are defined then comes the main section where you will start defining the structure of the calculator inside the GUI.

# The first thing is to create a frame for the input field input_frame = Frame(window, width = 312, height = 50, bd = 0, highlightbackground = "black", highlightcolor = "black", highlightthickness = 1)
input_frame.pack(side = TOP)

# Then you will create an input field inside the 'Frame' that was created in the previous step. Here the digits or the output will be displayed 'right' aligned
```



```
input_field = Entry(input_frame, font = ('arial', 18, 'bold'), textvariable = input_text, width = 50, bg = "#eee", bd = 0, justify = RIGHT)
input_field.grid(row = 0, column = 0) input_field.pack(ipady = 10) # 'ipady' is an internal padding to increase the height of input field

# Once you have the input field defined then you need a separate frame which will incorporate all the buttons inside it below the 'input field'
btns_frame = Frame(window, width = 312, height = 272.5, bg = "grey")
btns_frame.pack()

# The first row will comprise of the buttons 'Clear (C)' and 'Divide (/)'
clear = Button(btns_frame, text = "C", fg = "black", width = 32, height = 3, bd = 0, bg = "#eee", cursor = "hand2", command = lambda: btn_clear()).grid(row = 0, column = 0, columnspan = 3, padx = 1, pady = 1)
divide = Button(btns_frame, text = "/", fg = "black", width = 10, height = 3, bd = 0, bg = "#eee", cursor = "hand2", command = lambda: btn_click("/")).grid(row = 0, column = 3, padx = 1, pady = 1)

# The second row will comprise of the buttons '7', '8', '9' and 'Multiply (*)'
seven = Button(btns_frame, text = "7", fg = "black", width = 10, height = 3, bd = 0, bg = "#fff", cursor = "hand2", command = lambda: btn_click(7)).grid(row = 1, column = 0, padx = 1, pady = 1)
eight = Button(btns_frame, text = "8", fg = "black", width = 10, height = 3, bd = 0, bg = "#fff", cursor = "hand2", command = lambda: btn_click(8)).grid(row = 1, column = 1, padx = 1, pady = 1)
nine = Button(btns_frame, text = "9", fg = "black", width = 10, height = 3, bd = 0, bg = "#fff", cursor = "hand2", command = lambda: btn_click(9)).grid(row = 1, column = 2, padx = 1, pady = 1)
multiply = Button(btns_frame, text = "*", fg = "black", width = 10, height = 3, bd = 0, bg = "#eee", cursor = "hand2", command = lambda: btn_click("*")).grid(row = 1, column = 3, padx = 1, pady = 1)

# The third row will comprise of the buttons '4', '5', '6' and 'Subtract (-)'
four = Button(btns_frame, text = "4", fg = "black", width = 10, height = 3, bd = 0, bg = "#fff", cursor = "hand2", command = lambda: btn_click(4)).grid(row = 2, column = 0, padx = 1, pady = 1)
five = Button(btns_frame, text = "5", fg = "black", width = 10, height = 3, bd = 0, bg = "#fff", cursor = "hand2", command = lambda: btn_click(5)).grid(row = 2, column = 1, padx = 1, pady = 1)
six = Button(btns_frame, text = "6", fg = "black", width = 10, height = 3, bd = 0, bg = "#fff", cursor = "hand2", command = lambda: btn_click(6)).grid(row = 2, column = 2, padx = 1, pady = 1)
minus = Button(btns_frame, text = "-", fg = "black", width = 10, height = 3, bd = 0, bg = "#eee", cursor = "hand2", command = lambda: btn_click("-")).grid(row = 2, column = 3, padx = 1, pady = 1)

# The fourth row will comprise of the buttons '1', '2', '3' and 'Addition (+)'
one = Button(btns_frame, text = "1", fg = "black", width = 10, height = C, bg = "#fff", cursor = "hand2", command = lambda:
```

```
btn_click(1)).grid(row = 3, column = 0, padx = 1, pady = 1) two =
Button(btns_frame, text = "2", fg = "black", width = 10, height = 3, bd =
0, bg = "#fff", cursor = "hand2", command = lambda: btn_click(2)).grid(row
= 3, column = 1, padx = 1, pady = 1) three = Button(btns_frame, text = "3",
fg = "black", width = 10, height = 3, bd = 0, bg = "#fff", cursor =
"hand2", command = lambda: btn_click(3)).grid(row = 3, column = 2, padx =
1, pady = 1) plus = Button(btns_frame, text = "+", fg = "black", width =
10, height = 3, bd = 0, bg = "#eee", cursor = "hand2", command = lambda:
btn_click("+")).grid(row = 3, column = 3, padx = 1, pady = 1)

# Finally, the fifth row will comprise of the buttons '0', 'Decimal (.)',
and 'Equal To (=)' zero = Button(btns_frame, text = "0", fg = "black",
width = 21, height = 3, bd = 0, bg = "#fff", cursor = "hand2", command =
lambda: btn_click(0)).grid(row = 4, column = 0, columnspan = 2, padx = 1,
pady = 1) point = Button(btns_frame, text = ".", fg = "black", width = 10,
height = 3, bd = 0, bg = "#eee", cursor = "hand2", command = lambda:
btn_click(".")).grid(row = 4, column = 2, padx = 1, pady = 1)
equals = Button(btns_frame, text = "=", fg = "black", width = 10, height =
3, bd = 0, bg = "#eee", cursor = "hand2", command = lambda:
btn_equal()).grid(row = 4, column = 3, padx = 1, pady = 1)
window.mainloop()
```

Output:



Check Your Understanding

1. Essential thing to create a window screen using tkinter python?
   (a) call tk() function (b) create a button
   (c) To define a geometry (d) All of the above
2. fg in tkinter widget stands for _____.
   (a) foreground (b) background
   (c) forgap (d) None of the above

3. What is the purpose of bg in tkinter widget?
   (a) To change the direction of widget
   (b) To change the size of widget
   (c) To change the color of widget
   (d) To change the background of widget
4. How pack() function work in tkinter widget?
   (a) According to x,y coordinate (b) According to row and column vise
   (c) According to left,right,up,down (d) None of the above
5. In Python context, what does GUI stand for?
   (a) Graphical Unix interface (b) Graphical User Interface
   (c) General User Interface (d) Graphical User Interaction
6. Which of the following is invalid Tkinter widgets?
   (a) Button (b) Text
   (c) ColorPicher (d) Entry
7. Which of the following geometry managers are not available in Tkinter?
   (a) .place() (b) .grid()
   (c) .pack() (d) .flex()
8. How we import tkinker in python program?
   (a) import tkinter (b) import tkinter as t
   (c) from tkinter import (d) All of the above
9. Tkinker tool in python provides the _____
   (a) Database (b) OS commands
   (c) GUI (d) All of the above
10. What is the syntax to create a frame?
    (a) Frame(window,options) (b) win.frame(options)
    (c) Both of the above (d) None of the above

Answers

1. (a)	2. (a)	3. (d)	4. (c)	5. (b)	6. (c)	7. (d)	8. (d)	9. (c)	10. (a)

Practice Questions

Q.I Answer the following question in short.

1. Name the different tools Python provides for developing graphical user interfaces (GUIs).
2. Name the free GUI libraries for Python.
3. What is the purpose of tkinker?
4. Name the different widgets available in Tkinter.
5. List out Geometry Management methods.

6. List out any 5 button option.
7. List out any 5 label option.
8. List out any 5 frame option.

Q.II Answer the following question in long.

1. State the differences between:
   (a) Tkinter vs Kivy
   (b) PyQt vs PySide
   (c) Kivy Vs PyQT
   (d) WxPython vs Tkinter
2. Write the steps for GUI script coding to most tkinter programs follows.
3. Explain any three widgets available in Tkinter in brief.
4. Explain button widget in tkinker with an example.
5. Explain label widget in tkinker with an example.
6. Explain Frame widget in tkinker with an example.
7. Explain entry widget in tkinker with an example.
8. Explain widget resizing with an example.
9. Explain the Methods for Geometry Management.
10. Explain the following with proper syntax and example:
    Entry.delete, entry.insert

Q.III Define the term.

1. Tkinter
2. Geometry management
3. Bind method
4. grid()
5. pack()

■■■

6... Python Libraries

Learning Objectives ...

- To understand Statistical Analysis- NumPy, SciPy, Pandas, StatsModels.
- To learn Data Visualization- Matplotlib, Seaborn, Plotly.
- To know about Data Modelling and Machine Learning- Scikit-learn, XGBoost, Eli5.
- To study Deep Learning- TensorFlow, Pytorch, Keras.
- To understand Natural Language Processing (NLP)- NLTK, SpaCy, Gensim.

6.1 STATISTICAL ANALYSIS - NumPy, SciPy, Pandas, StatsModels

Why Python for Statistics?

- Today R language and Python language is more popular for statistical analysis.
- R is a language dedicated to statistics. Python is a general-purpose language with statistics modules.
- R has more statistical analysis features than Python, and specialized syntaxes. However, when it comes to building complex analysis pipelines that mix statistics with e.g. image analysis, text mining, or control of a physical experiment, the richness of Python is an invaluable asset.

6.1.1 NumPy

- It is a Python library for scientific computing in Python.
- It contains a collection of tools and techniques that can be used to solve on a computer mathematical models of problems in Science and Engineering.
- One of these tools is a high-performance multidimensional array object that is a powerful data structure for efficient computation of arrays and matrices. To work with these arrays, there's a huge amount of high-level mathematical functions operate on these matrices and arrays.
- NumPy is a general-purpose array-processing package.
- NumPy is the core library for scientific computing in Python.
- In 2005, Travis Oliphant created NumPy package by incorporating the features of Numarray into Numeric package.

- It is an extension module for Python, mostly written in C which makes sure that the precompiled mathematical and numerical functions and functionalities of NumPy guarantee great execution speed.
- Though python has a list, we need NumPy, because we cannot perform operations on all the elements of two lists directly. For example, we cannot multiply two lists directly we will have to do it element wise. This is where the role of NumPy comes into play.

Example: Python program to demonstrate a need of NumPy.

```
list1 = [1, 2, 3, 4 ,5, 6]
list2 = [10, 9, 8, 7, 6, 5]
# Multiplying both lists directly would give an error.
print(list1*list2)
```

Output:

```
TypeError: can't multiply sequence by non-int of type 'list'
Where as this can easily be done with NumPy arrays.
```

Example: Python program to demonstrate the use of NumPy arrays.

```
import NumPy as np
list1 = [1, 2, 3, 4, 5, 6]
list2 = [12, 9, 4, 7, 6, 5]
# Convert list1 into a NumPy array
a1 = np.array(list1)
# Convert list2 into a NumPy array
a2 = np.array(list2)
print(a1*a2)
```

Output:

```
array([12, 18, 12, 28, 30, 30])
```

Features of NumPy:

1. A powerful N-dimensional array object, multi-dimensional arrays and matrices.
2. A large library of high-level mathematical functions to operate on these matrices and arrays.
3. Sophisticated (broadcasting) functions; new multi-iterator object created for easy handling in C of broadcasting.
4. Tools for integrating C/C++ and Fortran code.
5. Useful linear algebra, Fourier transform, and random number capabilities.
6. Used as an efficient multi-dimensional container of generic data.
7. Arbitrary data-types can be defined using NumPy which allows NumPy to seamlessly and speedily integrate with a wide variety of databases.

8. Arrays can be more easily read from text files and created from buffers and iterators.
9. Arrays can be quickly written to files in text and/or binary mode.
10. Fancy indexing can be done on arrays using integer sequences and Boolean masks.
11. NumPy is often used along with packages like SciPy (Scientific Python) and Matplotlib (plotting library). This combination is widely used as a replacement for MatLab, a popular platform for technical computing.

Advantages of using NumPy with Python:

1. It is open source and free.
2. It supports array oriented computing.
3. It is efficiently implemented for multi-dimensional arrays and designed for scientific computation.
4. It is an extension on Python, embedding code from other languages like C, C++ and FORTRAN is very simple.
5. NumPy is extremely fast as compared to core Python.
6. Many advanced Python libraries, such as Scikit-Learn, Scipy, and Keras, make extensive use of the NumPy library. Therefore, to pursue a career in data science or machine learning, NumPy is a very good tool to master.
7. NumPy comes with a variety of built-in functionalities, which in core Python would take a fair bit of custom code.

Table 6.1: Difference between Python lists and NumPy array

Factors	Python lists	NumPy arrays
Library	The Python core library provided Lists.	NumPy is the core library for scientific computing in Python.
Definition	A list is the Python equivalent of an array, but is resizeable and can contain elements of different types.	A NumPy array is a grid of values, all of the same type, and is indexed by a tuple of nonnegative integers.
Memory	Pythons list requires more memory.	NumPy arrays requires smaller memory consumption and have better runtime behavior.
Speed	Python list operation takes more time and less convenient.	NumPy is faster and more convenient than a list.
Slicing	Slices of python lists are shallow copies.	All arrays generated by NumPy basic slicing are always views of the original array.

6.1.2 SciPy

- SciPy, pronounced as Sigh Pi, is a scientific python open source, distributed under the Berkeley Software Distribution (BSD) licensed library to perform Mathematical, Scientific and Engineering Computations.
- It adds substantial capabilities to NumPy. The SciPy package contains various toolboxes dedicated to common issues in scientific computing.
- Its different submodules correspond to different applications, such as interpolation, integration, optimization, image processing, statistics, special functions, etc.
- SciPy is open-source software for mathematics, science, and engineering. SciPy is a collection of mathematical algorithms and convenience functions built on the NumPy extension of Python. It adds significant power to the interactive Python session by providing the user with high-level commands and classes for manipulating and visualizing data.
- With SciPy an interactive Python session becomes a data-processing and system-prototyping environment rivaling systems such as MATLAB, IDL, Octave, R-Lab, and SciLab. It includes modules for statistics, optimization, integration, linear algebra, Fourier transforms, signal and image processing and more.
- The main reason for building the SciPy library is that, it should work with NumPy arrays. It provides many user-friendly and efficient numerical practices such as routines for numerical integration and optimization. Together, they run on all popular operating systems, are quick to install, and are free of charge. NumPy and SciPy are easy to use, but powerful enough to be depended upon by some of the world's leading scientists and engineers.
- The additional benefit of basing SciPy on Python is that this also makes a powerful programming language available for use in developing sophisticated programs and specialized applications. Scientific applications using SciPy benefit from the development of additional modules in numerous niches of the software landscape by developers across the world. Everything like web programming, database subroutines and classes has been made available to the Python programmer. All of this power is available in addition to the mathematical libraries in SciPy.

Advantages and Features of SciPy:

- SciPy contains varieties of sub packages which help to solve the most common issue related to Scientific Computation.
- SciPy package in Python is the most used Scientific library only second to GNU Scientific Library for C/C++ or Matlab's.
- Easy to use and understand as well as fast computational power.
- It can operate on an array of NumPy library.

SciPy Organization:

- SciPy is organized into sub-packages covering different scientific computing domains. These are summarized in the following table:

Table 6.2

Scipy Packages	Package Domain
scipy.cluster	Vector quantization / Kmeans.
scipy.constants	Physical and mathematical constants.
scipy.fftpack	Fourier transform.
scipy.integrate	Integration routines.
scipy.interpolate	Interpolation.
scipy.io	Data input and output.
scipy.linalg	Linear algebra routines.
scipy.ndimage	n-dimensional image package.
scipy.odr	Orthogonal distance regression.
scipy.optimize	Optimization.
scipy.signal	Signal processing.
scipy.sparse	Sparse matrices.
scipy.spatjal	Spatial data structures and algorithms.
scipy.special	Any special mathematical functions.
scipy.stats	Statistics.

- The basic data structure used by SciPy is a multidimensional array provided by the NumPy module. NumPy provides some functions for Linear Algebra, Fourier Transforms and Random Number Generation, but not with the generality of the equivalent functions in SciPy.

6.1.3 Pandas

- It is one of the most widely used python libraries in data science.
- It provides high-performance, easy to use structures and data analysis tools. Unlike NumPy library which provides objects for multi-dimensional arrays.
- Pandas provides in-memory 2-D table object called Dataframe. It is like a spreadsheet with column names and row labels.
- Pandas is an open source, providing high-performance, easy-to-use data structures and data analysis tools for the Python programming language.
- The panda's library has emerged into a power house of data manipulation tasks in python since it was developed in 2008. With its intuitive syntax and flexible data structure, it's easy to learn and enables faster data computation.
- The development of NumPy and pandas libraries has extended python's multipurpose nature to solve machine learning problems as well. The acceptance of python language in machine learning has been phenomenal since then.

- Python has long been great for data munging and preparation, but less so for data analysis and modeling. Pandas, enabling you to carry out your entire data analysis workflow in Python without having to switch to a more domain specific language like R. Python excels in performance, productivity, and the ability to collaborate by Combining with the excellent IPython toolkit and other libraries.

Features of Pandas:

1. A fast and efficient DataFrame object for data manipulation with integrated indexing.
2. Tools for reading and writing data between in-memory data structures and different formats: CSV and text files, Microsoft Excel, SQL databases, and the fast HDFS format.
3. Intelligent data alignment and integrated handling of missing data: Gain automatic label-based alignment in computations and easily manipulate messy data into an orderly form.
4. Flexible reshaping and pivoting of data sets.
5. Intelligent label-based slicing, fancy indexing, and subsetting of large data sets.
6. Columns can be inserted and deleted from data structures for size mutability.
7. Aggregating or transforming data with a powerful group by engine allowing split-apply-combine operations on data sets.
8. High performance merging and joining of data sets.
9. Hierarchical axis indexing provides an intuitive way of working with high-dimensional data in a lower-dimensional data structure.
10. Time series-functionality: Date range generation and frequency conversion, moving window statistics, moving window linear regressions, date shifting and lagging. Even create domain-specific time offsets and join time series without losing data.
11. Highly optimized for performance, with critical code paths written in Cython (C extension for python) or C.
12. Python with pandas is in use in a wide variety of academic and commercial domains, including Finance, Neuroscience, Economics, Statistics, Advertising, Web Analytics, and more.

Advantages of Pandas

1. An Extensive set of features.
2. Less writing, more work done.
3. Excellent data representation.
4. Make the data flexible and customizable.

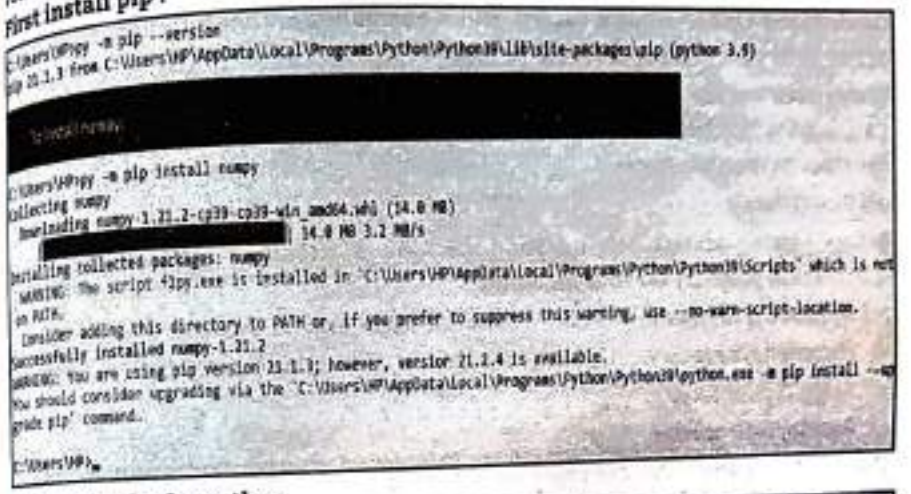
2.4 Installing NumPy, Scipy and Pandas

You can download Python from https://www.python.org/download/.

Installing via pip:

Goto start and run cmd command , we get the command prompt. Then do the following steps.

1. **First install pip :**



2. **Now upgrade the python**

3. **Now install pandas:**

The same command is used

py -m pip install pandas

Same way we can install matplotlib package.

6.1.5 StatsModels

- StatsModels is built on top of NumPy, SciPy, and matplotlib, but it contains more advanced functions for statistical testing and modelling.
- It is used to explore data, perform statistical tests and estimate statistical models.
- It includes regression models and linear models such as Linear regression, Generalized Linear Models (GLMs) and Generalized Estimating Equations (GEE) to perform all statistical operations.
- It also uses Pandas for data handling and Patsy for R-like formula interface. It takes its graphics functions from matplotlib. It is known to provide statistical background for other python packages.

Install StatsModels:

- Before getting StatsModels on your machine, StatsModels assumes the following functioning properly on your machine:
  - Python 2.6 or later
  - NumPy 1.6 or later
  - Scipy 0.11 or later
  - Pandas 0.12 or later
- Once you have these you can begin with installation.
- To install using pip, open your terminal and type the following command:

```
py -m  pip install statsmodels
```

- Once you are done with the installation, you can use StatsModels easily in your Python code by importing it:

```
import statsmodels
```

6.2 DATA VISUALIZATION- MATPLOTLIB, SEABORN, PLOTLY

- **Data visualization** is a generic term used to describe any attempt to help understanding of data by providing visual representation.
- Visualization of data makes it much easier to analyse and understand the textual and numeric data. Apart from saving time, increased used of data for decision making further adds to the importance and need of data visualization.

- Data visualization is a very important part of data analysis. You can use it to explore your data. If you understand your data well, you'll have a better chance to find some insights. Finally, when you find any insights, you can use visualizations again to be able to share your findings with other people.
- Following are visualization packages in Python:
  1. Matplotlib
  2. Seaborn
  3. Plotly

6.2.1 Matplotlib

- Matplotlib is an amazing visualization library in Python for 2D plots of arrays.
- Matplotlib is a multiplatform data visualization library built on NumPy arrays and designed to work with the broader SciPy stack. It was introduced by John Hunter in the year 2002.
- Matplotlib has a module named pyplot which makes things easy for plotting by providing feature to control line styles, font properties, formatting axes etc. It supports a very wide variety of graphs and plots namely - histogram, bar charts, power spectra, error charts etc.
- It is used along with NumPy to provide an environment that is an effective open source alternative for MatLab. It can also be used with graphics toolkits like PyQt and wxPython.
- Matplotlib is an excellent 2D and 3D graphics library for generating scientific figures.

Advantages of Matplotlib library:

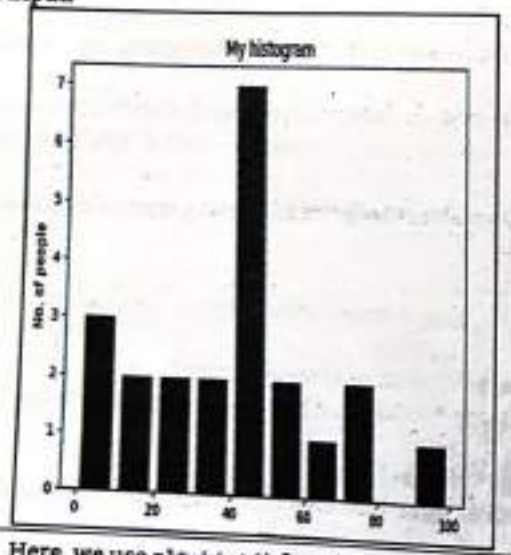
1. Easy to get started.
2. Matplotlib is free and open source code. It is also object-oriented and can be used in an object oriented way.
3. It could be used on any operating system via its array of backend.
4. It has a familiar interface similar to MATLAB and had a coherent vision to do 2D graphics, and do them well.
5. Support for \(\LaTeX\) formatted labels and texts.
6. Great control of every element in a figure, including figure size and DPI (Dots Per Inch).
7. High-quality output in many formats, including PNG, PDF, SVG, EPS.
8. GUI for interactively exploring figures and support for headless generation of figure files (useful for batch jobs).
9. Another characteristic of Matplotlib is its steep learning curve, which means that users usually make rapid progress after having started.

Program 6.1: Program for plotting histogram.

```
import matplotlib.pyplot as plt

# frequencies
ages = [2,5,70,40,30,45,50,45,43,40,44,60,7,13,57,18,90,77,32,21,20,40]
```

```
# setting the ranges and no. of intervals
range = (0, 100)
bins = 10

# plotting a histogram
plt.hist(ages, bins, range, color = 'green', histtype = 'bar', rwidth =
0.8)

# x-axis label
plt.xlabel('age')
# frequency label
plt.ylabel('No. of people')
# plot title
plt.title('My histogram')

# function to show the plot
plt.show()
```

Output:



- Here, we use `plt.hist()` function to plot a histogram.
- Frequencies are passed as the ages list.
- Range could be set by defining a tuple containing min and max value.
- Next step is to "bin" the range of values - that is, divide the entire range of values into a series of intervals and then count how many values fall into each interval. Here we have defined bins = 10. So, there are a total 100/10 = 10 intervals.

6.2.2 Seaborn

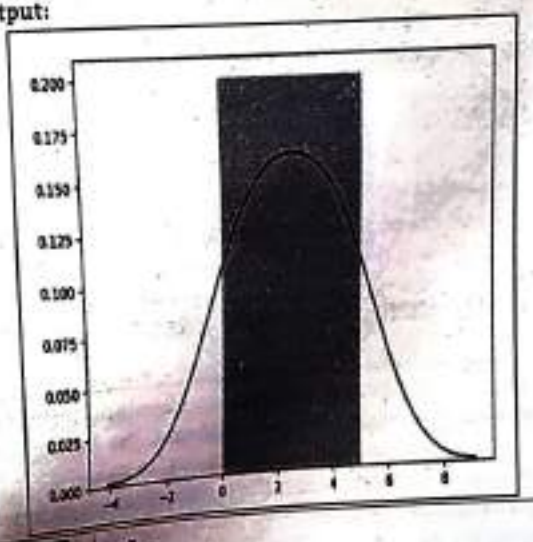
- Seaborn is a data visualization library built on top of Matplotlib and closely integrated with pandas data structures in Python.

- Seaborn offers the following functionalities:
  1. Dataset oriented API to determine the relationship between variables.
  2. Automatic estimation and plotting of linear regression plots.
  3. It supports high-level abstractions for multi-plot grids.
  4. Visualizing univariate and bivariate distribution.
- Using Seaborn we can plot wide varieties of plots like:
  1. Distribution Plots
  2. Pie Chart and Bar Chart
  3. Scatter Plots
  4. Pair Plots
  5. Heat maps

Program 6.2: Program for plotting displot.

```
#Plotting a Displot
import matplotlib.pyplot as plt
import seaborn as sns
sns.distplot([0, 1, 2, 3, 4, 5])
plt.show()
```

Output:



6.2.3 Plotly

- Python Plotly Library is an open-source library that can be used for data visualization and understanding data simply and easily. Plotly supports various types of plots like line charts, scatter plots, histograms, cox plots, etc.

- Plotly allows users to import, copy and paste, or stream data to be analyzed and visualized. For analysis and styling graphs, Plotly offers a Python sandbox (NumPy supported), datagrid, and GUI. Python scripts can be saved, shared, and collaboratively edited in Plotly.

Advantages Plotly over other visualization tools:

1. Plotly has hover tool capabilities that allow us to detect any outliers or anomalies in a large number of data points.
2. It is visually attractive that can be accepted by a wide range of audiences.
3. It allows us for the endless customization of our graphs that makes our plot more meaningful and understandable for others.

Program 6.3: Program for plotting the line chart

```
importplotly.express as px
# using the iris dataset
df =px.data.iris()
# plotting the line chart
fig =px.line(df, x="species", y="petal_width")
# showing the plot
fig.show()
```

Output:

6.3 DATA MODELLING AND MACHINE LEARNING - SCIKIT-LEARN, XGBOOST, ELI5

- The hidden power of Machine Learning (ML) lies in its ability to learn and improve algorithms by studying available data. The higher the data volume, the better the learning process.
- In order for data pipelines to work in an ML environment, the Data Architecture must be designed to work with the underlying data platform.
- ML works best with Big Data with high-volume data samples. To model such samples using machine learning many libraries are used in python like sciket-learn, XGBoost, ELI5.

6.3.1 Scikit-Learn

- There are several Python libraries which provide solid implementations of a range of machine learning algorithms. One of the best known is Scikit-Learn, a package that provides efficient versions of a large number of common algorithms.
- Scikit-Learn is characterized by a clean, uniform, and streamlined API, as well as useful and complete online documentation. A benefit of this is uniformity that once you understand the basic use and syntax of Scikit-Learn for one type of model, switching to a new model or algorithm is very straightforward.
- Scikit-Learn was initially developed by David Cournapeau as a Google summer of code project in 2007. Later Matthieu Brucher joined the project and started to use it as a part of his thesis work. In 2010, INRIA got involved and the first public release (v0.1 beta) was published in late January 2010. The project now has more than 30 active contributors and has had paid sponsorship from INRIA, Google, Tinyclues and the Python Software Foundation.

What is Scikit-Learn?

- Scikit-learn is pronounced as *sy-kit learn*, *sci* stands for science.
- Scikit-learn provides a range of supervised and unsupervised learning algorithms via a consistent interface in Python.
- Scikit-learn is probably the most useful library for machine learning in Python. It is on NumPy, SciPy and Matplotlib, this library contains a lot of efficient tools for machine learning and statistical modeling including classification, regression, clustering and dimensionality reduction. It is used for building the model and should not be used for reading the data, manipulating and summarizing it.
- The library is built upon the SciPy (Scientific Python) that must be installed before you can use Scikit-learn. This stack that includes:
  - **NumPy:** Base n-dimensional array package.
  - **SciPy:** Fundamental library for scientific computing.
  - **Matplotlib:** Comprehensive 2D/3D plotting.
  - **IPython:** Enhanced interactive console.
  - **Sympy:** Symbolic mathematics.
  - **Pandas:** Data structures and analysis.
- Extensions or modules for SciPy care conventionally named SciKits. Such as, the module provides learning algorithms and is named Scikit-learn.
- The vision for the library is a level of robustness and support required for use in production systems. This means a deep focus on concerns such as ease of use, code quality, collaboration, documentation and performance.
- Although the interface is Python, c-libraries are leverage for performance such as NumPy for arrays and matrix operations, LAPACK, LibSVM and the careful use of cython.

- Every machine learning algorithm in Scikit-Learn is implemented via the Estimator API, which provides a consistent interface for a wide range of machine learning applications.

Features of Scikit-Learn:

1. Simple and efficient tools for data mining and data analysis. It features various classification, regression and clustering algorithms including support vector machines, random forests, gradient boosting, k-means, etc.
2. Accessible to everybody and reusable in various contexts.
3. Built on the top of NumPy, SciPy and Matplotlib.
4. Open source, commercially usable – BSD license.
5. The library is focused on modeling data. It is not focused on loading, manipulating and summarizing data.

Some popular groups of models provided by Scikit-learn include:

1. **Clustering:** For grouping unlabeled data such as K-Means.
2. **Cross Validation:** For estimating the performance of supervised models on unseen data.
3. **Datasets:** For test datasets and for generating datasets with specific properties for investigating model behavior.
4. **Dimensionality Reduction:** For reducing the number of attributes in data for summarization, visualization and feature selection such as principal component analysis.
5. **Ensemble Methods:** For combining the predictions of multiple supervised models.
6. **Feature Extraction:** For defining attributes in image and text data.
7. **Feature Selection:** For identifying meaningful attributes from which to create supervised models.
8. **Parameter Tuning:** For getting the most out of supervised models.
9. **Manifold Learning:** For summarizing and depicting complex multi-dimensional data.
10. **Supervised Models:** A vast array not limited to generalize linear models, discriminate analysis, naive bayes, lazy methods, neural networks, support vector machines and decision trees.

The Scikit-Learn API is designed with the following guiding principles:

1. **Consistency:** All objects share a common interface drawn from a limited set of methods, with consistent documentation.
2. **Inspection:** All specified parameter values are exposed as public attributes.
3. **Limited object hierarchy:** Only algorithms are represented by Python classes; datasets are represented in standard formats (NumPy arrays, Pandas DataFrames, SciPy sparse matrices) and parameter names use standard Python strings.
4. **Composition:** Many machine learning tasks can be expressed as sequences of more fundamental algorithms, and Scikit-Learn makes use of this wherever possible.

5. **Sensible defaults:** When models require user-specified parameters, the library defines an appropriate default value.
6. In practice, these principles make Scikit-Learn very easy to use, once the basic principles are understood.

6.3.2 XGBoost

- XGBoost stands for "Extreme Gradient Boost".
- XGBoost is used for supervised learning problems, where we use the training data (with multiple features) xi to predict a target variable yi.
- It implements machine learning algorithms under the Gradient Boosting framework. XGBoost provides a parallel tree boosting that solve many data science problems in a fast and accurate way.
- In supervised learning, Boosting takes a more iterative approach. In this technique many models are combined together to perform the final one, but takes a more clever approach.
- Rather than training all of the models in isolation of one another, boosting trains models in succession, with each new model being trained to correct the errors made by the previous ones. Models are added sequentially until no further improvements can be made.
- The advantage of this iterative approach is that the new models being added are focused on correcting the mistakes which were caused by other models. In a standard ensemble method where models are trained in isolation, all of the models might simply end up making the same mistakes!
- **Gradient Boosting** specifically is an approach where new models are trained to predict the residuals (i.e. errors) of prior models.
- The following figure shows the boosting process.

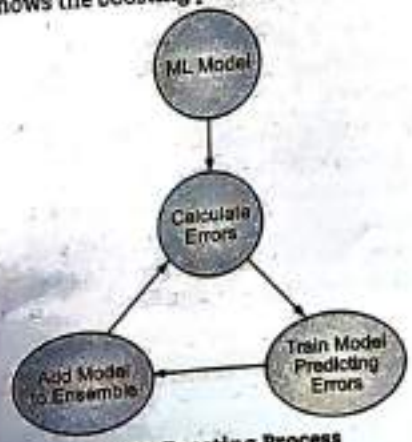


Fig. 6.1: Boosting Process

6.3.3 ELI5

- ELI5 is a python package which helps to debut machine learning classifiers and explain their predictions.

Why ELI5?

- Debugging is an important step when working with machine learning models. For example, we have to check if we have any noise in our features, when we are working with text that affects the predictions such as unwanted symbols or numbers. ELI5 is a library which can help us debug machine learning models.
- ELI5 can handle not only simples cases, but even for simple cases having a unified API for inspection has as value:
  - We can get a nicely formatted result immediately by calling ready-made function from ELI5.
  - Formatting code can be reused between machine learning frameworks.
  - The code like feature filtering or text highlighting can be reused.

6.4 DEEP LEARNING - TENSORFLOW, PYTORCH, KERAS

- Python offers concise and readable code. While complex algorithms and versatile workflows stand behind machine learning and AI, Python's simplicity allows developers to write reliable systems.
- Python code is understandable by humans, which makes it easier to build models for machine learning.
- Deep learning is the most interesting and powerful machine learning technique right now.
- Deep learning models are being used for very difficult problems and making progress, like colorizing image and videos based on the context in the scene.
- Deep learning models are being used in bold new ways, such as cutting the head off a network trained on one problem and tuning it for a completely different problem, and getting impressive results.
- Combinations of deep learning models are being used to both identify objects in photographs and then generate textual descriptions of those objects, a complex multi-media problem that was previously thought to require large artificial intelligence systems.
- Top deep learning libraries are available on the Python ecosystem like TensorFlow , Keras, the best-of-breed applied deep learning library.

6.4.1 TensorFlow

- TensorFlow is a Python library for fast numerical computing created and released by Google.
- It is a foundation library that can be used to create Deep Learning models directly or by using wrapper libraries that simplify the process built on top of TensorFlow.

- TensorFlow is an end-to-end open source platform for machine learning. It has a comprehensive, flexible ecosystem of tools, libraries and community resources that lets researchers push the state-of-the-art in ML and developers easily build and deploy ML powered applications.

Why Tensorflow?

- Tensorflow bundles together on machine learning and deep learning models and algorithms and make them useful by way of common betterment.
- Google use machine learning in all of its products to improve our search engine the translation image captioning or recommendations. For example, Google users can experience a faster and more refined search with artificial intelligence. If the user types of keyword in search bar, Google provides the recommendation what could be the next one.
- Tensorflow used by a lot of companies in industries to name a few, start with Airbnb the leading global online marketplace and hospitality service. The Airbnb ingenious and data science team applied machine learning using Tensorflow to classify the images and detect object as key in helping to improve the guest experience.
- The healthcare industry using Tensorflow, GE healthcare is training a neural network to identify specific anatomic during the brain MRI exam to help improve speed and reliability.
- PayPal is using Tensorflow to stay at the cutting edge of fraud detection.
- Using Tensorflow, PayPal has been able to recognize complex fraud patterns to increase decline accuracy while improving the experience of legitimate users through increased precision in identification.

6.4.2 PyTorch

- PyTorch is an optimized tensor library primarily used for Deep Learning applications using GPUs and CPUs. It is an open-source machine learning library for Python, mainly developed by the Facebook AI Research team.
- PyTorch is the premier open-source deep learning framework developed and maintained by Facebook. At its core, PyTorch is a mathematical library that allows you to perform efficient computation and automatic differentiation on graph-based models.
- Tensorflow is much better for production models and scalability. It was built to be production ready. Whereas, PyTorch is easier to learn and lighter to work with, and hence, is relatively better for passion projects and building rapid prototypes.
- PyTorch is used for computer vision and Natural Language Processing (NLP) applications. Google Colab was developed by Google to help the masses access powerful GPU resources to run deep learning experiments. It offers GPU (Graphics Processing Units) support and integration with Google Drive for storage.

- PyTorch defines a class called Tensor (torch.Tensor) to store and operate on homogeneous multidimensional rectangular arrays of numbers.

PyTorch provides two high-level features:

- Tensor computing (like NumPy) with strong acceleration via Graphics Processing Units (GPU).
- Deep neural networks built on a type-based automatic differentiation system.

6.4.3 Keras

- One of the most powerful and easy-to-use Python libraries for developing and evaluating deep learning models is Keras; It wraps the efficient numerical computation libraries Theano and TensorFlow. The advantage of this is mainly that you can get started with neural networks in an easy and fun way.
- Keras is a deep learning API written in Python, running on top of the machine learning platform TensorFlow. It was developed with a focus on enabling fast experimentation and doing good research.

Features of Keras:

1. **Simple:** But not simplistic. Keras reduces developer cognitive load to free you to focus on the parts of the problem that really matter.
2. **Flexible:** Keras adopts the principle of progressive disclosure of complexity. Simple workflows should be quick and easy, while arbitrarily advanced workflows should be possible via a clear path that builds upon what you've already learned.
3. **Powerful:** Keras provides industry-strength performance and scalability. It is used by organizations and companies including NASA, YouTube, or Waymo.

Keras was developed and maintained by François Chollet, a Google engineer using four guiding principles:

1. **Modularity:** A model can be understood as a sequence or a graph alone. All the concerns of a deep learning model are discrete components that can be combined in arbitrary ways.
2. **Minimalism:** The library provides just enough to achieve an outcome, no frills and maximizing readability.
3. **Extensibility:** New components are intentionally easy to add and use within the framework, intended for researchers to trial and explore new ideas.
4. **Python:** No separate model files with custom file formats. Everything is native Python.

6.4.4 Keras Vs Tensorflow Vs Pytorch

- Keras is a neural network library while TensorFlow is the open-source library for a number of various tasks in machine learning.
- TensorFlow provides both high-level and low-level APIs while Keras provides only high-level APIs. Pytorch, on the other hand, is a lower-level API focused on direct work with array expressions.

In keras, there is usually very less frequent need to debug simple networks. But in case of Tensorflow, it is quite difficult to perform debugging. Pytorch on the other hand has better debugging capabilities as compared to the other two.

Keras has a simple architecture. It is more readable and concise. Tensorflow on the other hand is not very easy to use even though it provides Keras as a framework that makes work easier. PyTorch has a complex architecture and the readability is less when compared to Keras.

Keras is usually used for small datasets as it is comparatively slower. On the other hand, TensorFlow and PyTorch are used for high performance models and large datasets that require fast execution.

With the increasing demand in the field of Data Science, there has been an enormous growth of Deep learning technology in the industry. With this, all the three frameworks have gained quite a lot of popularity. Keras tops the list followed by TensorFlow and PyTorch. It has gained immense popularity due to its simplicity when compared to the other two.

6.5 NATURAL LANGUAGE PROCESSING (NLP) - NLTK, SpaCy, Gensim

- Natural Language Processing (NLP) is a process of manipulating or understanding the text or speech by any software or machine. An analogy is that humans interact and understand each other's views and respond with the appropriate answer. In NLP, this interaction, understanding, and response are made by a computer instead of a human.
- The following are the NLP libraries:
  1. NLTK
  2. SpaCy
  3. Gensim

6.5.1 NLTK (Natural Language Toolkit)

- This is one of the most usable and mother of all NLP libraries.
- The Natural Language Toolkit (NLTK) is a platform used for building Python programs that work with human language data for applying in statistical Natural Language Processing (NLP). It contains text processing libraries for tokenization, parsing, classification, stemming, tagging and semantic reasoning.
- The Natural Language Toolkit is an open source library for the Python programming language originally written by Steven Bird, Edward Loper and Ewan Klein for use in development and education.
- It comes with a hands-on guide that introduces topics in computational linguistics as well as programming fundamentals for Python which makes it suitable for linguists who have no deep knowledge in programming, engineers and researchers that need to explore into computational linguistics, students and educators.

Advantages:

1. The most well-known and full NPL library.
2. Many third-party extensions.
3. Plenty of approaches to each NLP task.
4. Fast sentence tokenization.
5. Supports the largest number of langauges compared to other libraries.

Disadvantages:

1. Complicated to learn and use.
2. Quite slow.
3. In sentence tokenization, NLTK only splits text by sentences, without analyzing the semantic structure.
4. Processes strings which is not very typical for object-oriented language python.
5. Does not provide neural network models.
6. No integrated word vectors.

6.5.2 SpaCy

- This is a completely optimized and highly accurate library widely used in deep learning.
- SpaCy is an open-source software library for advanced natural language processing, written in the programming languages Python and Cython. The library is developed by Matthew Honnibal and Ines Montani, the founders of the software company Explosion.
- SpaCy focuses on providing software for production usage rather than teaching and research.

Features:

1. Non-destructive tokenization.
2. Named entity recognition.
3. Support for 61+ languages.
4. 46 statistical models for 16 languages.
5. Pre-Trained word vectors
6. State-of-the-art speed.
7. Easy deep learning integration.
8. Labeled dependency parsing.
9. Syntax-driven sentence segmentation.
10. Export to NumPy data arrays.
11. Efficient binary serialization.
12. Easy model packaging and deployment.
13. Robust, rigorously evaluated accuracy.

Advantages:

1. The fastest NLP framework.
2. Easy to learn and use because it has one single highly optimized tool for each task.
3. Processes objects; more object-oriented, comparing to other libs.

4. Uses neural networks for training some models.
5. Provides built-in word vectors.
6. Active support and development.

Disadvantages:

1. Lacks flexibility, comparing to NLTK.
2. Sentence tokenization is slower than in NLTK.
3. Does not support many languages. There are models only for 7 languages and "multi-language" models.

6.5.3 Gensim

- Genism is a robust open source NLP library support in Python. This library is highly efficient and scalable.
- Gensim is a Python library for representing documents as semantic vectors, as efficiently (computer-wise) and painlessly (human-wise) as possible.
- Gensim is designed to process raw, unstructured digital texts (*"plain text"*) using unsupervised machine learning algorithms.
- We built Gensim from scratch for:
  - **Practicality**: As industry experts, we focus on proven, battle-hardened algorithms to solve real industry problems. More focus on engineering, less on academia.
  - **Memory independence**: There is no need for the whole training corpus to reside fully in RAM at any one time. Can process large, web-scale corpora using data streaming.
  - **Performance**: Highly optimized implementations of popular vector space algorithms using C, BLAS and memory-mapping.

Advantages:

1. Works with large datasets and processes data streams.
2. Provides tf-idf vectorization, word2vec, document2vec, latent semantic analysis, latent Dirichlet allocation.
3. Support deep learning.

Disadvantages:

1. Designed primarily for unsupervised text modeling.
2. Does not have enough tools to provide full NLP pipeline, so should be used with some other library (SpaCy or NLTK).

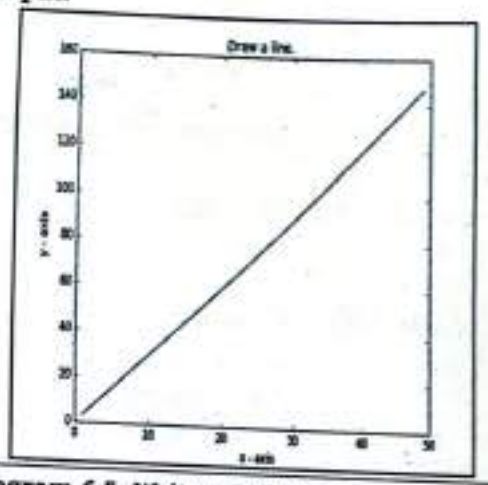
Sample Programs:

Program 6.4: Write a Python program to draw a line with suitable label in the x axis, y axis and a title.

```
import matplotlib.pyplot as plt
X = range(1, 50)
Y = [value * 3 for value in X]
```

```
print("Values of X:")
print(*range(1,50))
print("Values of Y (thrice of X):")
print(Y)
# Plot lines and/or markers to the Axes.
plt.plot(X, Y)
# Set the x axis label of the current axis.
plt.xlabel('x - axis')
# Set the y axis label of the current axis.
plt.ylabel('y - axis')
# Set a title
plt.title('Draw a line.')
# Display the figure.
plt.show()
```

Output:



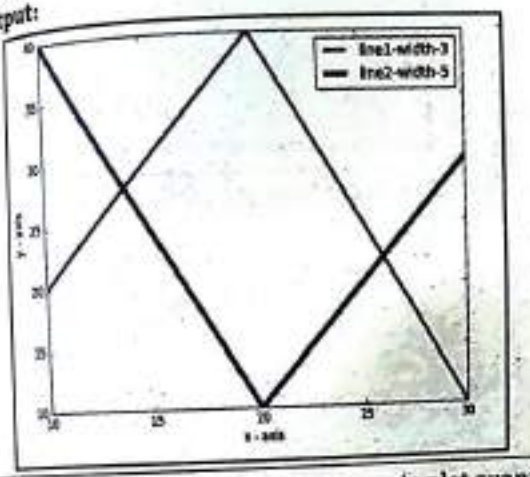
Program 6.5: Write a Python program to plot two or more lines with legends, different widths and colors.

```
import matplotlib.pyplot as plt
# line 1 points
x1 =[10,20,30]
y1 =[20,40,10]
# line 2 points
x2 =[10,20,30]
y2 =[40,10,30]
# Set the x axis label of the current axis.
plt.xlabel('x - axis')
# Set the y axis label of the current axis.
plt.ylabel('y - axis')
```



```
# Set a title
plt.title('Two or more lines with different widths and colors with suitable
legends ')
# Display the figure.
plt.plot(x1,y1, color='blue', linewidth =3, label ='line1-width-3')
plt.plot(x2,y2, color='red', linewidth =5, label ='line2-width-5')
# show a legend on the plot
plt.legend()
plt.show()
```

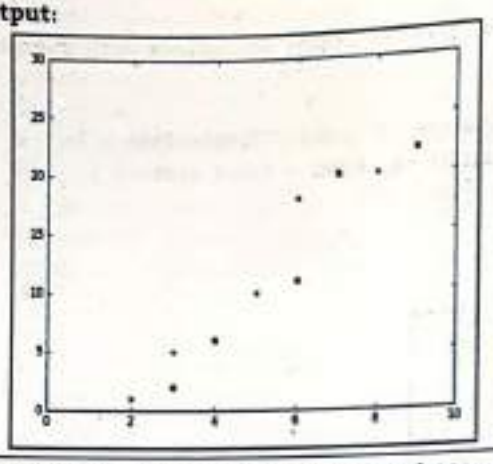
Output:



Program 6.6: Write a Python program to plot quantities which have an x and y position.

```
import NumPy as np
import pylab as pl
# Make an array of x values
x1 =[2,3,5,6,8]
# Make an array of y values for each x value
y1 =[1,5,10,18,20]
# Make an array of x values
x2 =[3,4,6,7,9]
# Make an array of y values for each x value
y2 =[2,6,11,20,22]
# set new axes limits
pl.axis([0,10,0,30])
# use pylab to plot x and y as red circles
pl.plot(x1, y1,'b*', x2, y2,'ro')
# show the plot on the screen
pl.show()
```

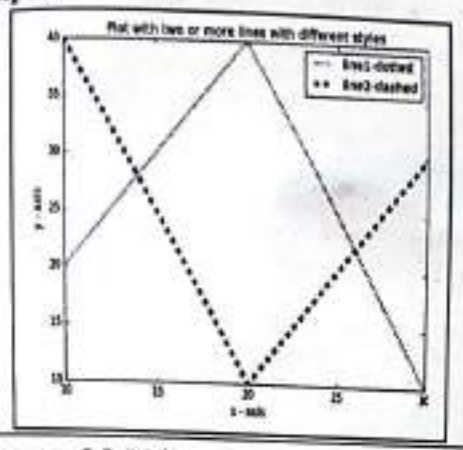
Output:



Program 6.7: Write a Python program to plot two or more lines with different styles.

```
import matplotlib.pyplot as plt
# line 1 points
x1 =[10,20,30]
y1 =[20,40,10]
# line 2 points
x2 =[10,20,30]
y2 =[40,10,30]
# Set the x axis label of the current axis.
plt.xlabel('x - axis')
# Set the y axis label of the current axis.
plt.ylabel('y - axis')
# Plot lines and/or markers to the Axes.
plt.plot(x1,y1, color='blue', linewidth =3,
                                label ='line1-dotted',linestyle='dotted')
plt.plot(x2,y2, color='red', linewidth =5,
                                label ='line2-dashed', linestyle='dashed')
# Set a title
plt.title("Plot with two or more lines with different styles")
# show a legend on the plot
plt.legend()
# function to show the plot
plt.show()
```

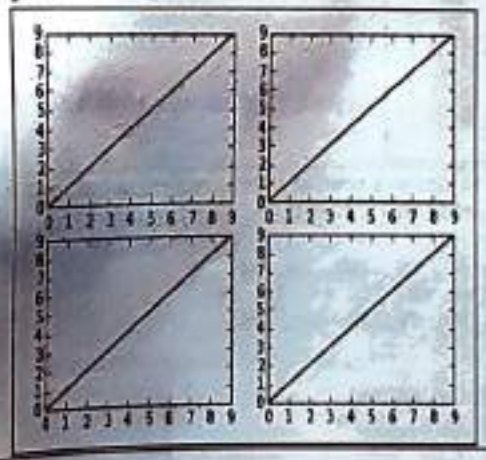

Output:



Program 6.8: Write a Python program to draw line or graph on subplots.

```
import matplotlib.pyplot as plt
x = range(10)
y = range(10)
fig = plt.figure()
plt.subplot(2, 2, 1)
plt.plot(x, y)
plt.subplot(2, 2, 2)
plt.plot(x, y)
plt.subplot(2, 2, 3)
plt.plot(x, y)
plt.subplot(2, 2, 4)
plt.plot(x, y)
plt.show()
```

Output:

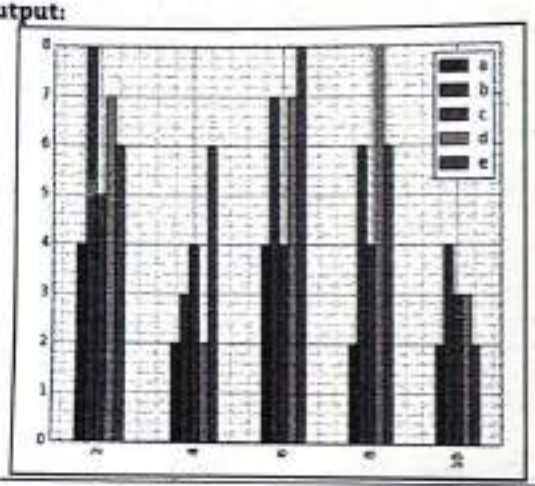


Program 6.9: Write a Python program to create bar plot from a DataFrame.

Sample Data Frame:

```
   a  b  c  d  e
24,8,5,7,6
42,3,4,2,6
64,7,4,7,8
82,6,4,8,6
102,4,3,3,2
from pandas import DataFrame
import matplotlib.pyplot as plt
import NumPy as np

a=np.array([[4,8,5,7,6],[2,3,4,2,6],[4,7,4,7,8],[2,6,4,8,6],[2,4,3,3,2]])
df=DataFrame(a, columns=['a','b','c','d','e'], index=[2,4,6,8,10])

df.plot(kind='bar')
# Turn on the grid
plt.minorticks_on()
plt.grid(which='major', linestyle='-', linewidth='0.5', color='green')
plt.grid(which='minor', linestyle=':', linewidth='0.5', color='black')
plt.show()
:
```

Output:



Program 6.10: Write a Python program to draw a scatter plot comparing two subject marks of Mathematics and Science. Use marks of 10 students.

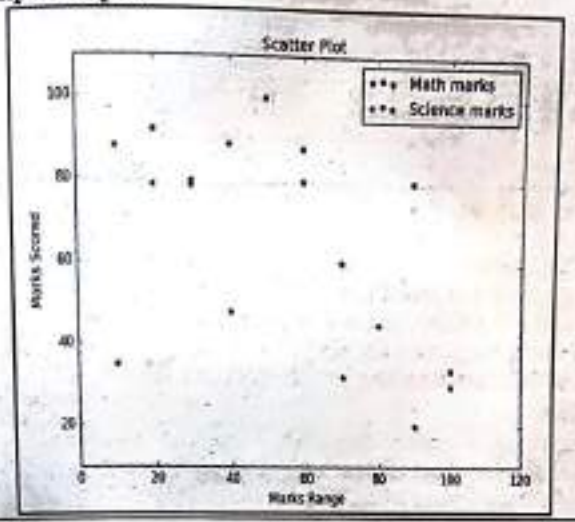
Test Data:

```
math_marks = [88, 92, 80, 89, 100, 80, 60, 100, 80, 34]
science_marks = [35, 79, 79, 48, 100, 88, 32, 45, 20, 30]
marks_range = [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
import matplotlib.pyplot as plt
import pandas as pd
math_marks =[88,92,80,89,100,80,60,100,80,34]
science_marks =[35,79,79,48,100,88,32,45,20,30]
```



```
marks_range =[10,20,30,40,50,60,70,80,90,100]
plt.scatter(marks_range, math_marks, label='Math marks', color='r')
plt.scatter(marks_range, science_marks, label='Science marks', color='g')
plt.title('Scatter Plot')
plt.xlabel('Marks Range')
plt.ylabel('Marks Scored')
plt.legend()
plt.show()
```

Sample Output:



Check Your Understanding

1. David Cournapeau developed _____ as a Google summer of code project.
   (a) Matlab (b) Scilab
   (c) Scikit learn (d) None
2. _____ must be installed before you use Scikit-learn.
   (a) Matlab (b) Scilab
   (c) Scipy (d) None
3. The more popular NLP library is _____.
   (a) NLTK (b) SpaCy
   (c) Gensim (d) Scikit
4. XGBoost stands for _____.
   (a) Extreme Google Boost (b) Extreme Gradient Boost.
   (c) Extended Gradient Boost (d) External Gradient Boost
5. _____ is an amazing visualization library in Python for 2D plots of arrays.
   (a) Scilearn (b) matplotlib
   (c) matlab (c) scilab
6. Matplotlib has a module named _____ makes things easy for plotting.
   (a) Scilearn (b). pyplot
   (c) matlab (c) scilab

7. _____ function plots y versus x as line.
   (a) line() (b) draw()
   (c) plot() (d) subplot()
8. _____ function adds a subplot to the current figure.
   (a) line() (b) draw()
   (c) plot() (d) subplot()
9. _____ function read an image from a file into an array.
   (a) Loadimg() (b) openimg()
   (c) imread() (d) read()

Answers

1. (c)	2. (c)	3. (a)	4. (b)	5. (b)	6. (b)	7. (c)	8. (d)	9. (c)

Practice Questions

Q.I Answer the following Questions in short.

1. What is Scikit-learn?
2. What are the features of Scikit learn?
3. What are the guiding principles behind Scikit-learn API?
4. What are the basic steps in using the Scikit-Learn estimator API?
5. What are the advantages of using Matplotlib library?
6. What are the different output formats supported by Matplotlib library?

Q.II Answer the following Questions.

1. What are the advantages and disadvantage of Gensim?
2. What are the advantages and disadvantage of SpaCy?
3. What are the advantages and disadvantage of NLTK?
4. Compare NLTK and SpaCy.
5. Compare Keras, Tensorflow and Pytorch
6. Write the four principal of keras.
7. State the guiding principles for designing Scikit-Learn API.
8. State the features of Scikit-learn.
9. What are the Advantages of Plotly over other visualization tools?
10. State the advantages of Matplotlib.
11. What is Seaborn?
12. What is Scikit-learn?
13. List out the plots that can plot with Seaborn.
14. State the use of Keras.
15. State the use of Pytorch.
16. State the uses of Tensorflow.

Q.III Define the terms.

1. Seaborn
2. Data Visualization
3. Data Modelling
4. XGBoost
5. SpaCy