SYLLABUS

Programming with 'Python' (22616)

Teaching Scheme			Credit (L+T+P)	Examination Scheme											
				Theory						Practical					
L	T	P		Paper Hrs.	ESE		PA		Total		ESE		PA		Total	
					Max	Min	Max	Min	Max	Min	Max	Min	Max	Min	Max	Min
3	-	2	5	3	70	28	30*	00	100	40	25@	10	25	10	50	20

Unit	Unit Outcomes (UOs) (in cognitive domain)	Topics and Sub - topics
Unit - I **Introduction and Syntax of Python Program**	1a. Identify the given Variables, Keywords and constants in Python. 1b. Use indentation, comments in the given program. 1c. Install the given Python IDE and editor. 1d. Develop the python program to display the given text.	1.1 Features of Python - Interactive, Object - oriented, Interpreted, Platform independent. l.2 Python building blocks - Identifiers, Keywords, Indention, Variables, Comments. 1.3 Python environment setup - Installation and working of IDE 1.4 Running Simple Python scripts to display 'welcome' message. 1.5 Python Data Types : Numbers, String, Tuples, Lists, Dictionary. Declaration and use of data types.
Unit - II **Python Operators and Control Flow Statements**	2a. Write simple Python program for the given arithmetic expressions. 2b. Use different types of operators for writing the the arithmetic expressions. 2c. Write a 'Python' program using decision making structure for two-way branching to solve the given problem. 2d. Write a 'Python' program using decision making structure for multi-way branching to solve the given problem.	2.1 Basic Operators : Arithmetic, Comparison / Relational, Assignment, Logical, Bitwise, Membership, Identity operators, Python Operator Precedence. 2.2 Control Flow : 2.3 Conditional Statements (if, if . . . else, nested if) 2.4 Looping in python (while loop, for loop, nested loops) 2.5 Loop manipulation using continue, pass, break, else.

Unit - III **Data Structures in Python**	3a. Write python program to use and manipulate lists for the given problem. 3b. Write python program to use and manipulate Tuples for the given problem. 3c. Write python program to use and manipulate Sets for the given problem. 3d. Write python program to use and manipulate Dictionaries for the given problem.	3.1 Lists : a) Defining lists, accessing values in list, deleting values in list, updating lists. b) Basic List Operations c) Built - in List functions 3.2 Tuples : a) Accessing values in Tuples, deleting values in Tuples, and updating Tuples. b) Basic Tuple operations. c) Built - in Tuple functions. 3.3 Sets : a) Accessing values in Set, deleting values in Set and updating Sets. b) Basic Set operations. c) Built - in Set functions 3.4 Dictionaries : a) Accessing values in Dictionary, deleting values in Dictionary and updating Dictionary. b) Basic Dictionary operations. c) Built- in Dictionaries functions.
Unit - IV **Python Functions, modules, and Packages**	4a. Use the Python standard functions for the given problem. 4b. Develop relevant user defined functions for the given problem using Python code. 4c. Write Python module for the given problem. 4d. Write Python package for the given problem.	4.1 Use of Python built - in functions (e.g. type / data conversion functions, math functions etc.) 4.2 User defined functions : Function definition, Function calling, function arguments and parameter passing, Return statement, Scope of Variables : Global variable and Local Variable. 4.3 Modules : Writing modules, importing modules, importing objects from modules, Python built - in modules (e.g. Numeric and mathematical module, Functional Programming Module) Namespace and Scoping. 4.4 Python Packages : Introduction, Writing Python packages, Using standard (e.g. math, scipy, Numpy, matplotlib, pandas etc.) and user defined packages.

Unit - V **Object Oriented Programming in Python**	5a. Create classes and objects to solve the given problem. 5b. Write Python code for data hiding for the given problem. 5c. Write Python code using data abstraction for the given problem. 5d. Write Python program using inheritance for the given problem.	5.1 Creating Classes and Objects. 5.2 Method Overloading and Overriding. 5.3 Data Hiding. 5.4 Data abstraction 5.5 Inheritance and composition classes 5.6 Customization via inheritance specializing inherited method.
Unit VI **File I/O Handling and Exception Handling**	6a. Write Python code for the given reading values from keyboard. 6b. Read data from the given file. 6c. Write the given data to a file. 6d. Handle the given exceptions through Python program.	6.1 I/O Operations : Reading keyboard input, Printing to screen. 6.2 File Handling : Opening file in different modes, accessing file contents using standard library functions, Reading and writing files, closing a file, Renaming and deleting files, Directories in Python, File and directory related standard functions. 6.3 Exception Handling : Introduction, Exception handling - 'try : except :' statement, 'raise' statement, User defined exceptions

Table of Contents

Unit - I

Unit - II

Unit - III

Unit - IV

Unit - V

Unit - VI

UNIT - I

1 Introduction and Syntax of Python Program

1.1 Features of Python Program

Following are features of python :

1. Python is **free and open source** programming language. The Python software can be downloaded and used freely.
2. It is **high level programming language.**
3. It is **simple and easy** to learn.
4. It is **portable**. That means python programs can be executed on various platforms without altering them.
5. The Python programs **do not require compilation,** rather they are **interpreted**. Internally, Python converts the source code into an intermediate form called **bytecodes** and then translates this into the native language of your computer and then runs it.
6. It is an **object oriented programming** language.
7. It can be **embedded** within your C or C++ programs.
8. It is **rich set of functionality** available in its huge standard library.
9. Python has a powerful set of **built-in data types** and **easy-to-use control constructs.**
10. It has **rich and supportive community** to help the developer while creating new applications.
11. Using **few lines of code** many useful applications can be developed using Python.

Uses of Python in Industry

- Top companies are using Python as their business application development. Even the Central Intelligence Agency (CIA) is using Python to maintain their websites.
- **Google's** first search engine was written in Python. It was developed in the late 90s.
- **Facebook** uses the Python language in their Production Engineering.
- **NASA** uses Workflow Automation Tool which is written in Python.
- **Nokia** which is a popular telecommunication company who uses Python for its platform such as S60.
- The entire stack of **Dropbox** was written in Python.
- **Quora** is a popular social commenting websites which is also written in Python.
- **Instagram** is another website which uses Python for its front-end.
- **YouTube** also uses scripted Python for their websites.

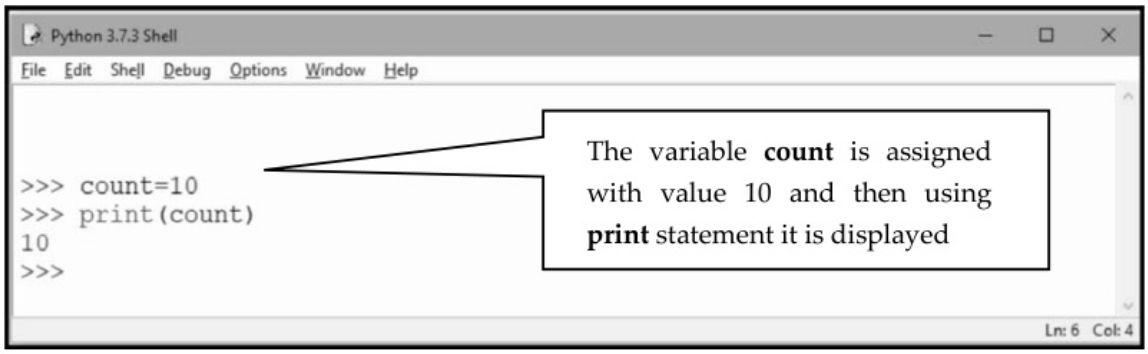
Review Question

1. *Enlist different features of Python.*

1.2 Python Building Blocks

1.2.1 Identifiers and Variables

- **Definition :** A variable is nothing but a reserved memory location to store values.
- Variable is an entity to which the programmer can assign some values. Programmer choose the variable name which is meaningful. For example :
- In the following example we have declared the variable count to which value 10 is assigned with. This value is displayed by passing the variable name to **print** statement.



We can re-declare the variables for assigning different values. For example

```
>>> count =10
>>> print(count)
10
>>> count = 1000
>>> print(count)
1000
```

The variable count is initially assigned with 10 and then re-assigned with the value 1000

Rules for variable names

- Variable names must be meaningful. The variable name normally should denote its purpose. For example - if the variable name is **even_count,** then clearly total number of even elements is stored in this variable.
- The variable names can be arbitrarily long. They contain both letters and digits but they cannot start with a number.
- Normally variable name should start with lower case letter.
- The underscore character is allowed in the variable name. For example – int total_elements

1.2.2 Keywords

The keywords are special words reserved for some purpose. The python3 has following list of keywords

False	class	finally	Is	return
None	continue	for	lambda	try
True	def	from	nonlocal	while
and	del	global	not	with

as	elif	if	Or	yield
assert	else	import	pass	
break	except	in	raise	

The names of keywords can be not be used as variable name. For instance a variable name can not be **from** because it is a keyword.

1.2.3 Indentation

- Leading white space at the beginning of the logical line is called indentation.
- Python programs get structured through indentation, i.e. code blocks are defined by their indentation.
- All statements with the same distance to the right belong to the same block of code, i.e. the statements within a block line up vertically.
- If a block has to be more deeply nested, it is simply indented further to the right.
- For example –

```
age = 1;
if age <= 2:
   print(' Infant')
```

Here the line is indented that means this statement will execute only if the if statement is true

1.2.4 Comments

- Comments are the kind of statements that are written in the program for program understanding purpose.
- By the comment statements it is possible to understand what exactly the program is doing.
- In Python, we use the hash (#) symbol to start writing a comment.
- It extends up to the newline character.
- Python Interpreter ignores comment.
- For example

```
# This is comment line
print("I love my country")
```

- If we have comments that extend multiple lines, one way of doing it is to use hash (#) in the beginning of each line. For example

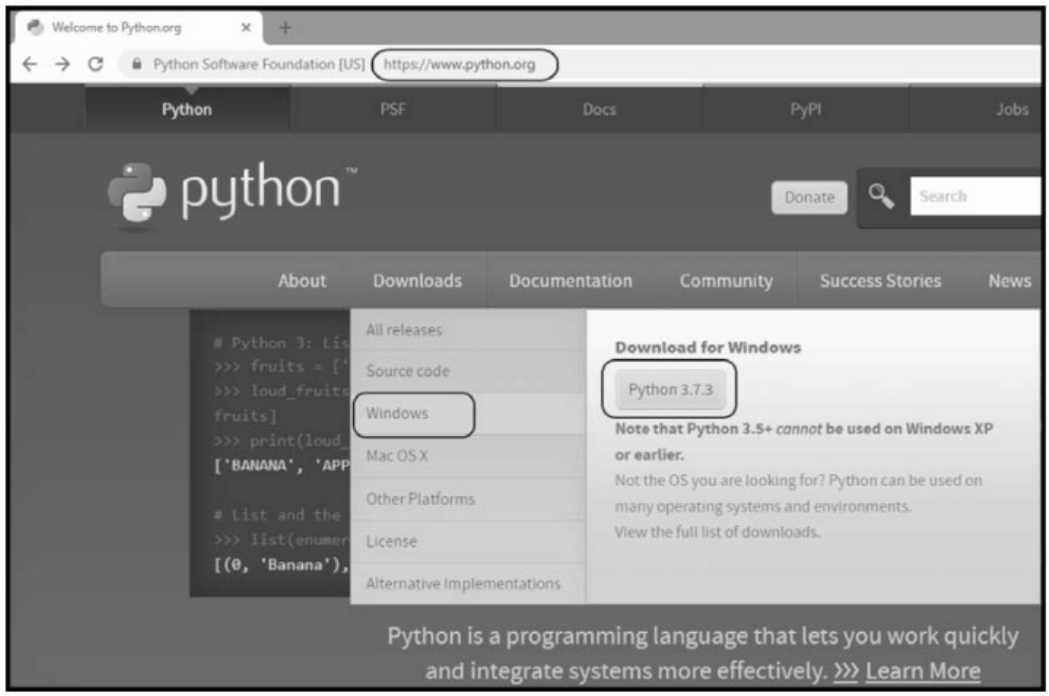
```
#This is
#another example
#of comment statement
```

Review Questions

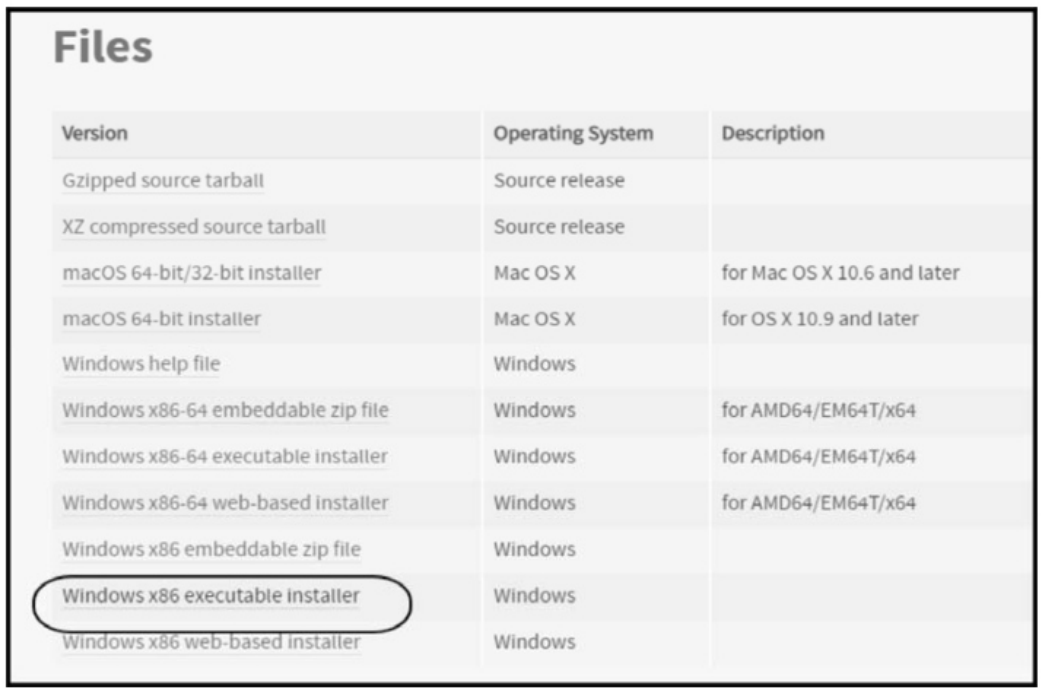
1. *What is the significance of having indentation in Python ?*
2. *How to write comment statement in Python ?*

1.3 Python Environmental Setup

First, download the latest version of Python from the official website. I have downloaded python 3.7 version. For that purpose open the web site **https://www.python.org**

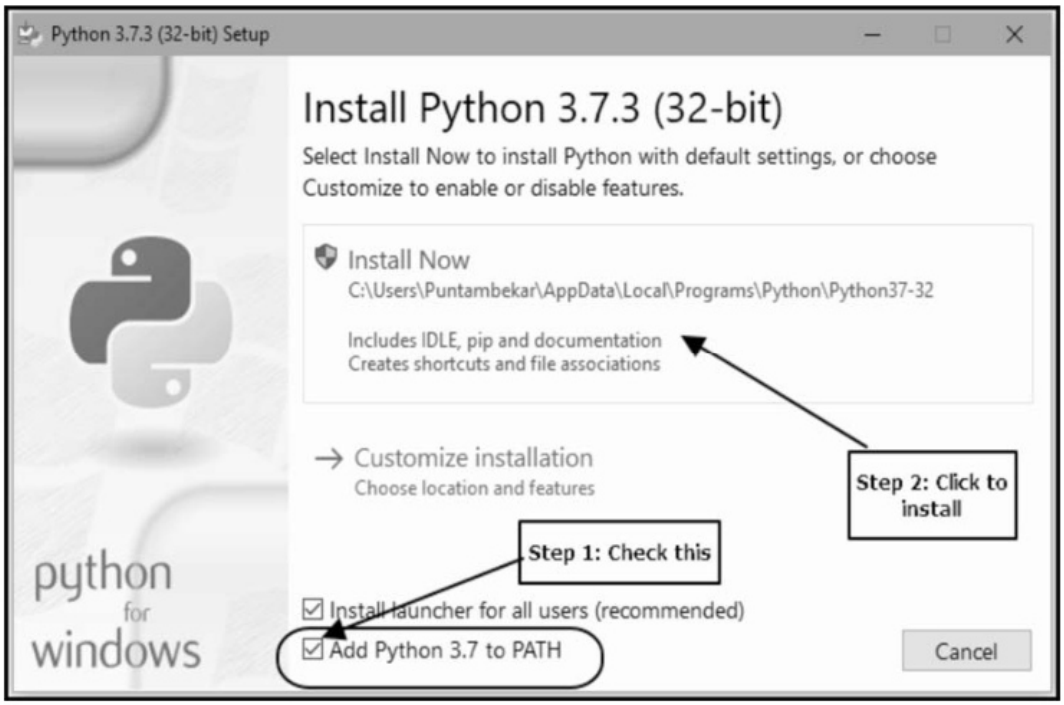


Now Select the file based on architecture of your PC. For Windows it can be 32 bit or 64 bit. If it is 32 bit then select the file as shown in the following screen shot

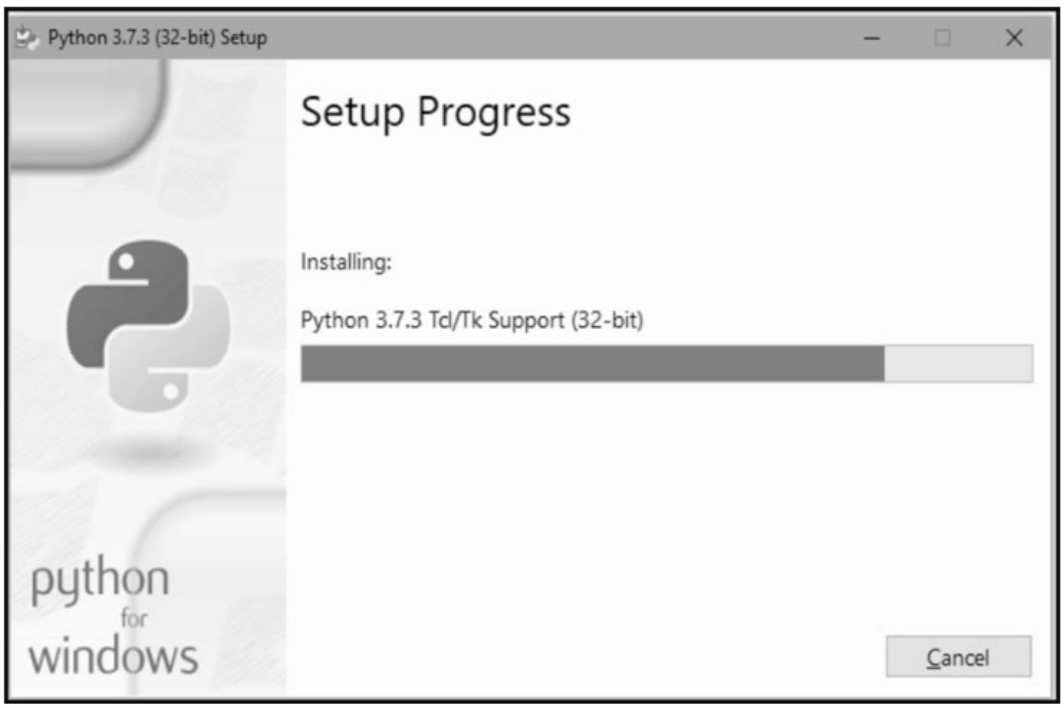


The executable file Python 3.7.3.exe will get downloaded on your PC. Now double click this executable file to install Python. For 64-bit architecture, select **Windows x86-64 executable installer**.

Then following screen will appear. Just check the Add path checkbox and click Install Now. The illustrative screenshot will help you.



The installation will progress.

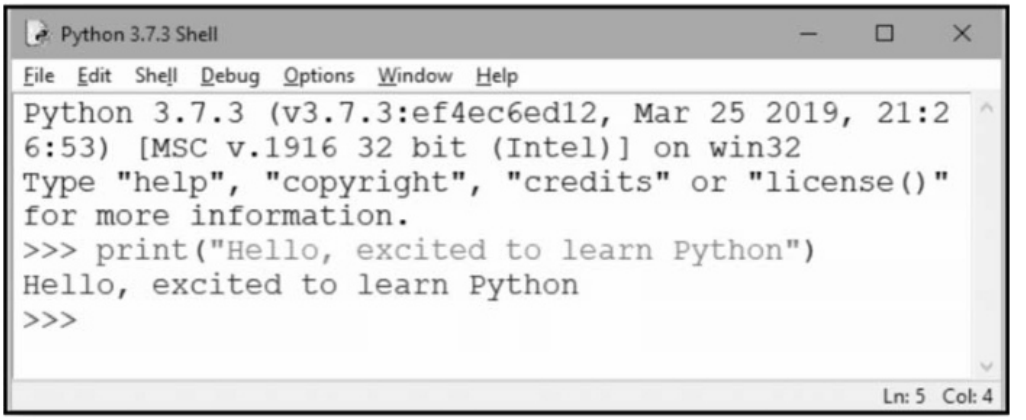


Finally you will get successful installation message.

Congratulations!!!. The Python is installed on your PC. Now you can write and execute the programs in Python using Python Shell.

Go to the List of Applications in your PC, Locate Python, either click on Python or IDLE to get the shell prompt. I use Python IDLE. The shell prompt appears as follows –

The python shell will be displayed and >>> prompt will be displayed. One can type different programming constructs and get the instant result of the command entered. **For example**



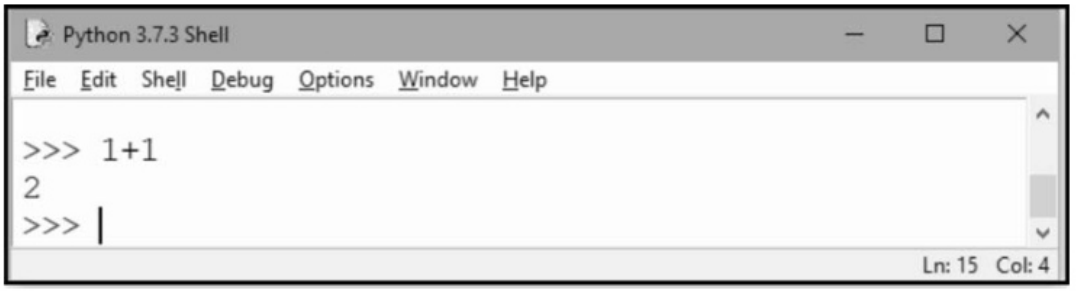
In above command we have used **print** command to display the desired message at the prompt.
The last line is a **prompt** that indicates that the interpreter is ready for you to enter code. If you type a line of code and hit Enter, the interpreter displays the result. For example

1.3.1 Modes of Working in Python

- Python has two basic modes : **normal** and **interactive**.

1) Interactive mode

- **Interactive mode** is a command line shell which gives immediate feedback for each statement, while running previously fed statements in active memory. As new lines are fed into the interpreter, the fed program is evaluated both in part and in whole.
- The >>> is Python's way of telling you that you are in interactive mode. In interactive mode what you type is immediately run. For example - If we type 1+1 on the interpreter the immediate result i.e. 2 will be displayed by interpreter.

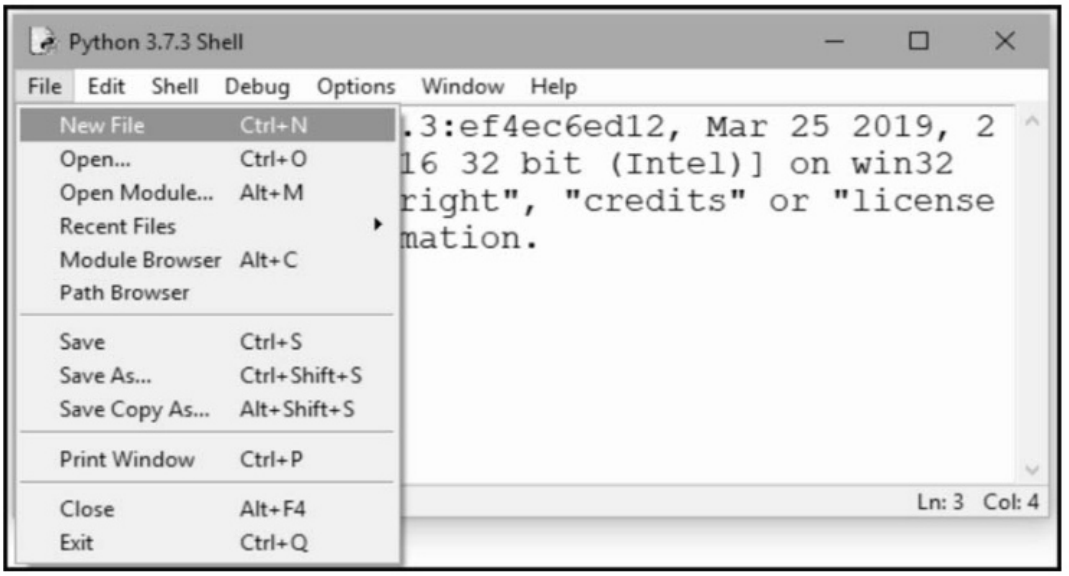


2) Script mode

- This is also called as **normal mode**. This is a mode in which the python commands are stored in a file and the file is saved using the extension **.py**
- For example : We can write a simple python program in script mode using following steps

Step 1 : Open python Shell by clicking the Python IDE.

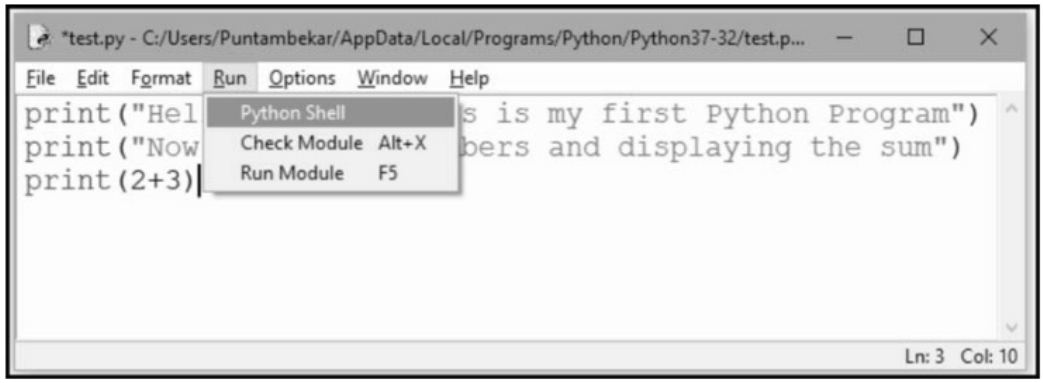
Step 2 : On File Menu Click on **New File** option.



Step 3 : Give some suitable file name with extension .py (I have created **test.py**).

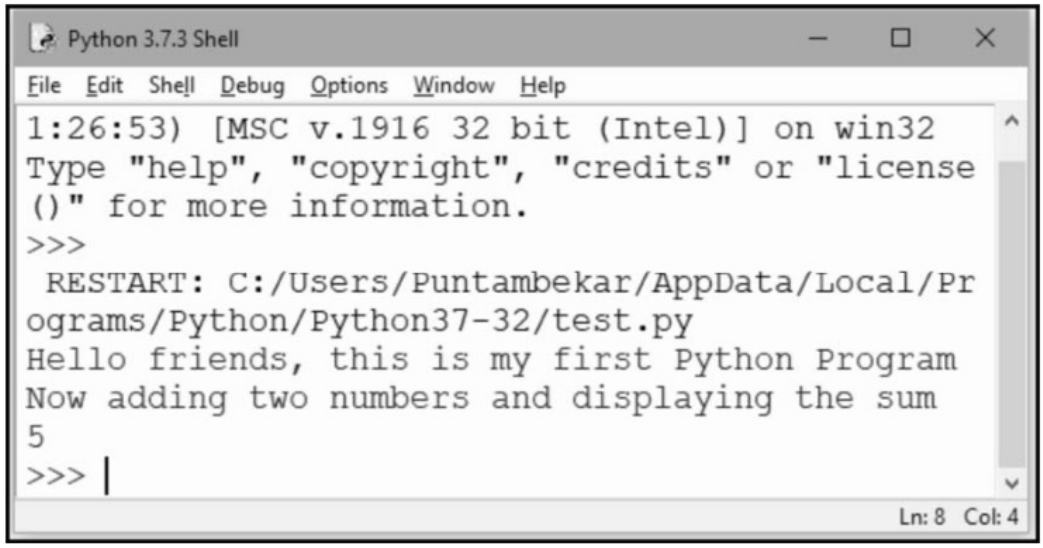
Step 4 : A file will get opened and the type some programming code. Sample file is as follows -

Step 5 : Now run your code by clicking on **Run ->Run Module** on Menu bar. Following screenshot illustrates it



For running the script we can also use F5 key.

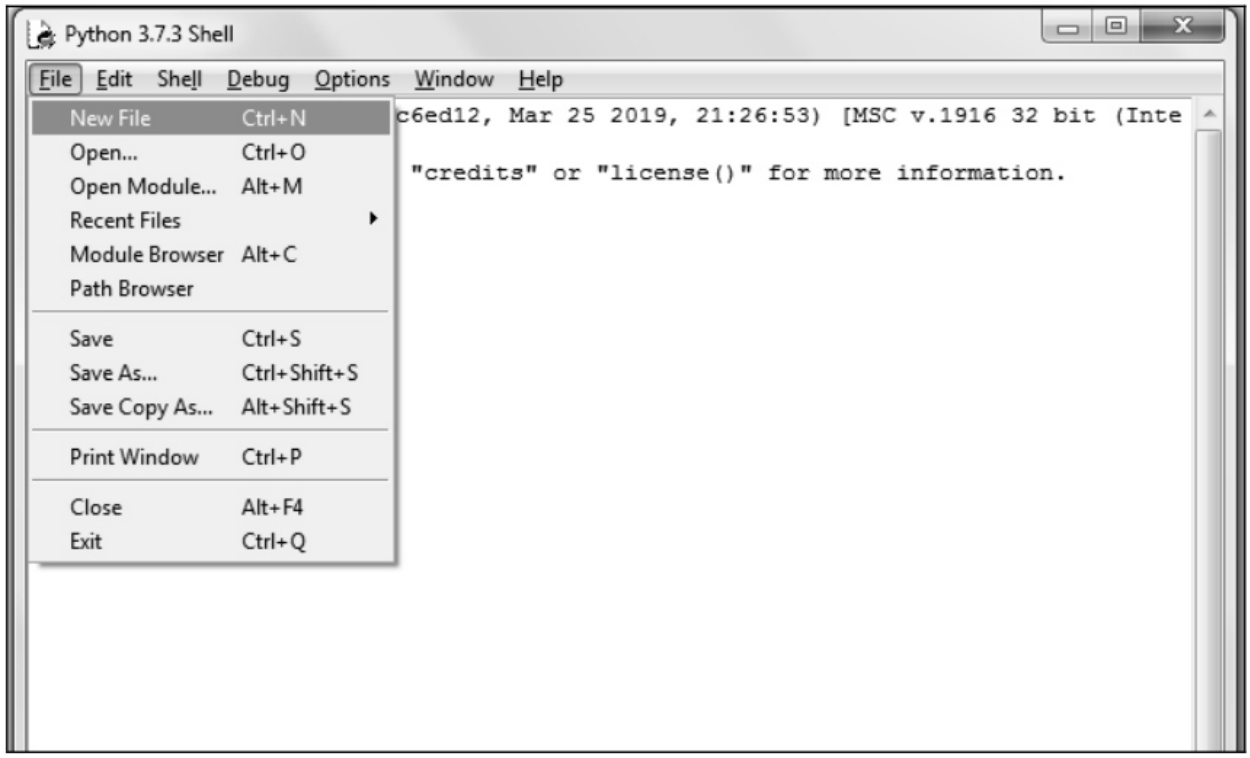
Step 6 : The output will be displayed on the python shell. It is as follows



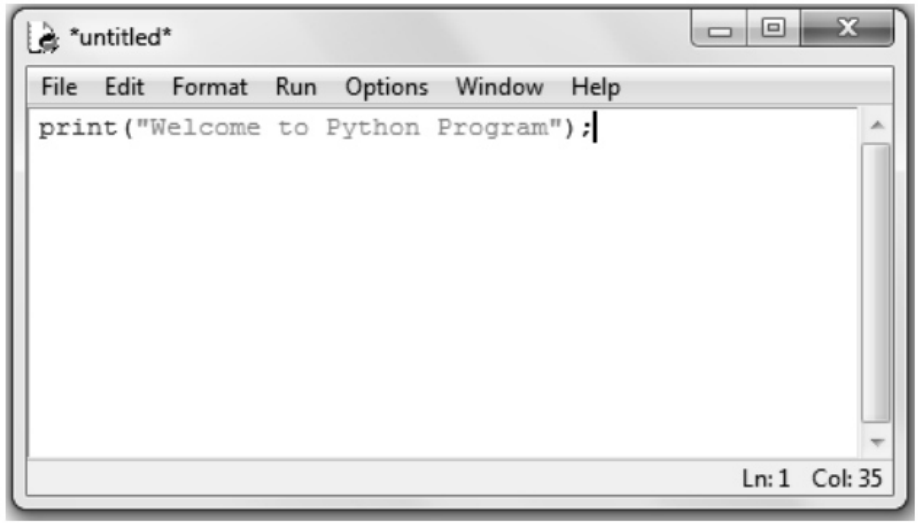
1.4 Running Simple Python Script to Display 'welcome' Message

Step 1 : Open python Shell by clicking the Python IDE.

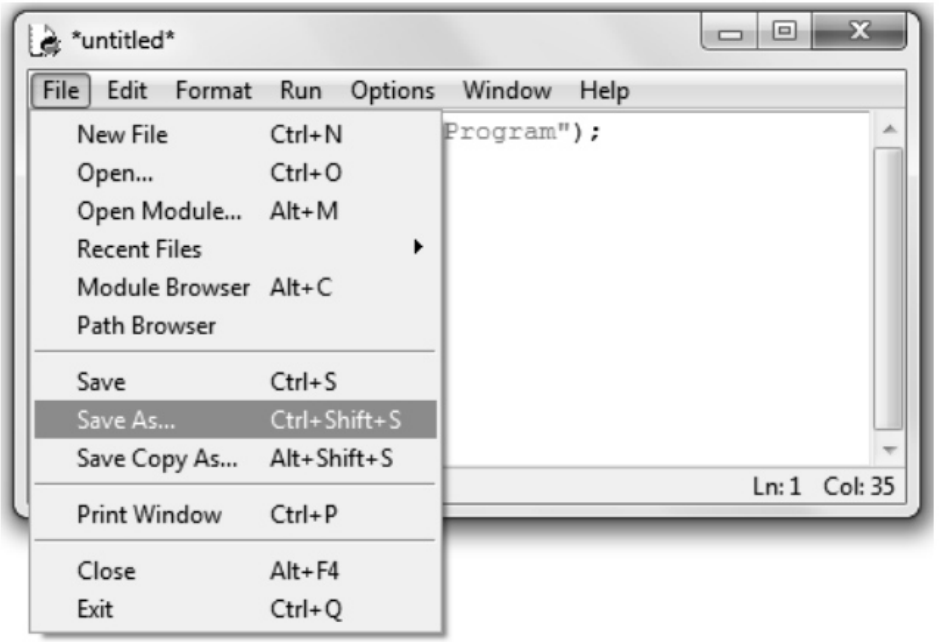
Step 2 : On **File** Menu then, Click on **New File** option.



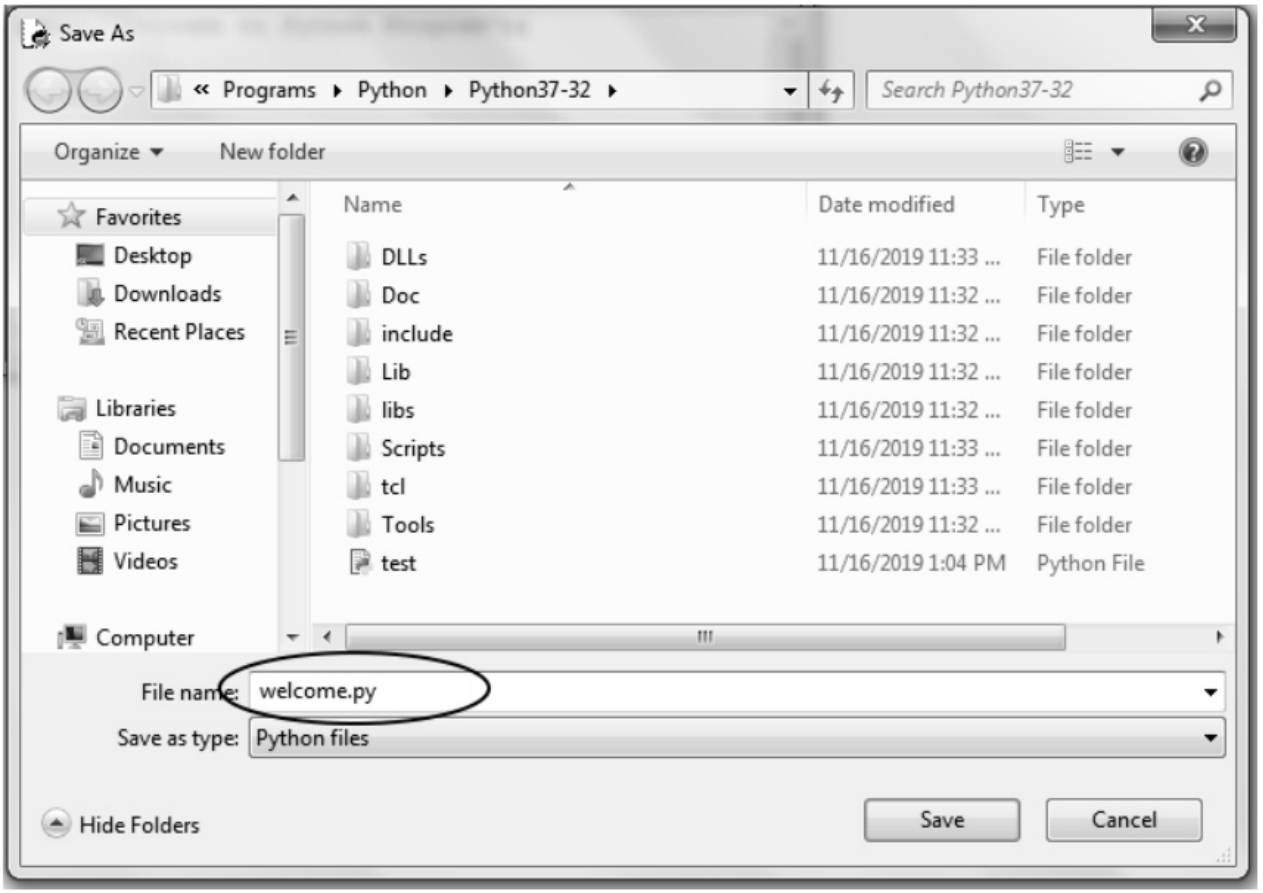
Step 3 : A file will get opened and the type some programming code. The illustrated code is –



Step 4 : Now select File -> Save As Option



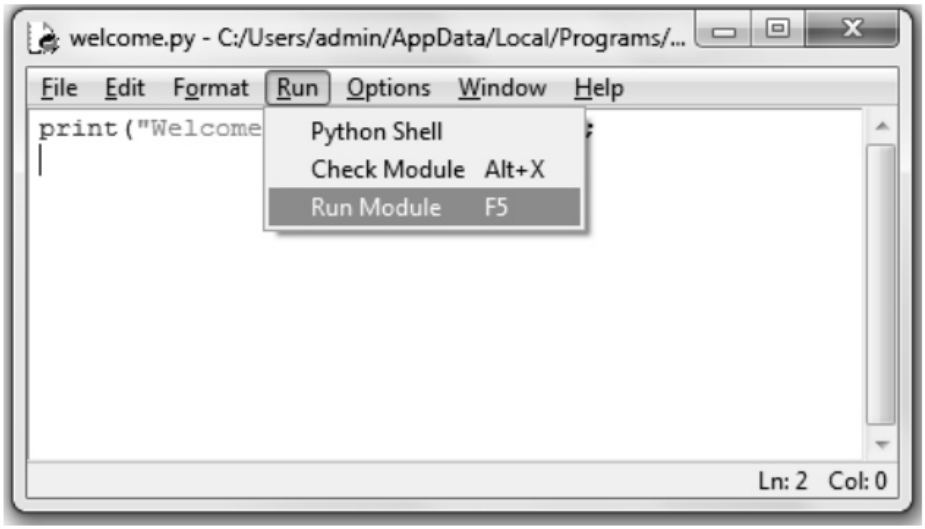
Give some suitable file name with extension .py (I have created **welcome.py**).



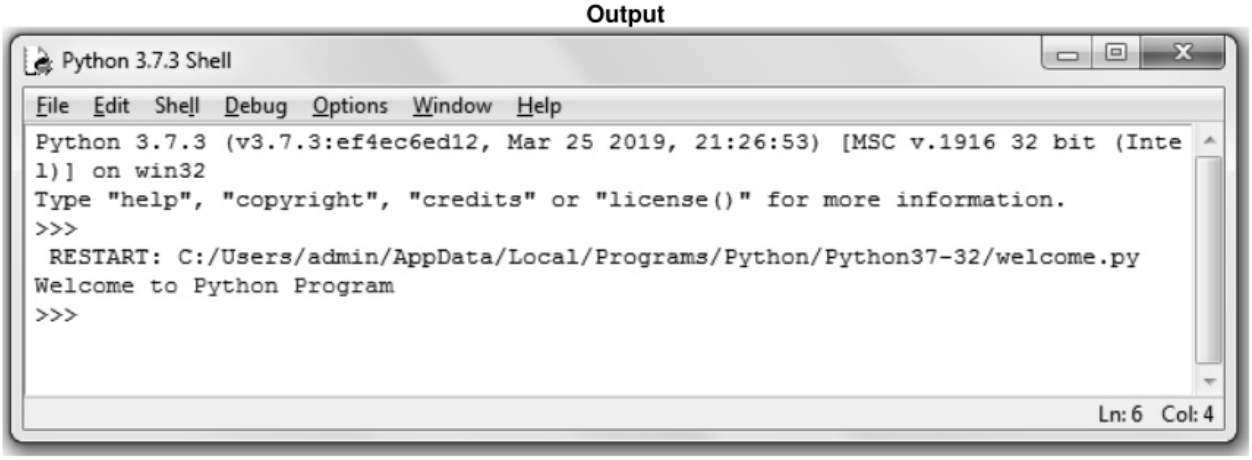
welcome.py

```
print("Welcome to Python Program");
```

Step 5 : From the File menu, click on **Run->Run Module.** Or you can press F5 button to run the above program.



Output



1.5 Python Data Types

- Data types are used to define type of variable.
- In Python there are five types of data type that are used commonly and those are –

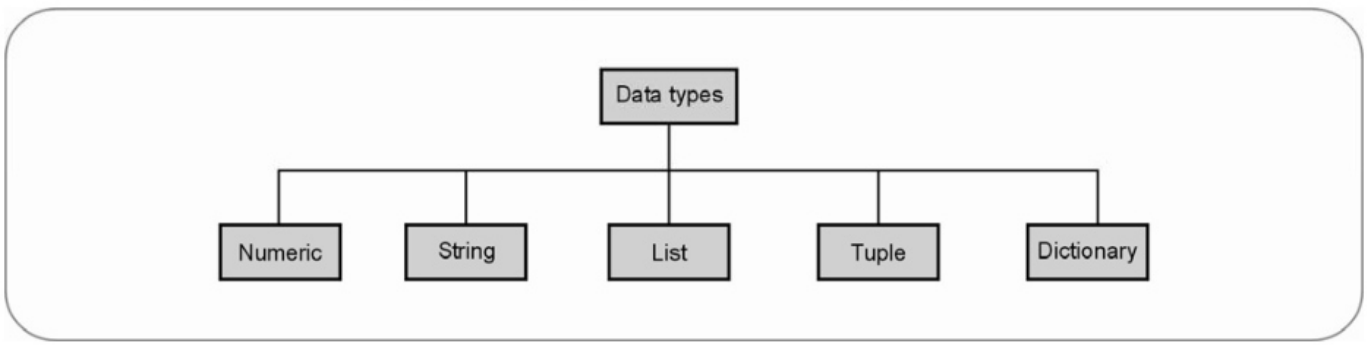
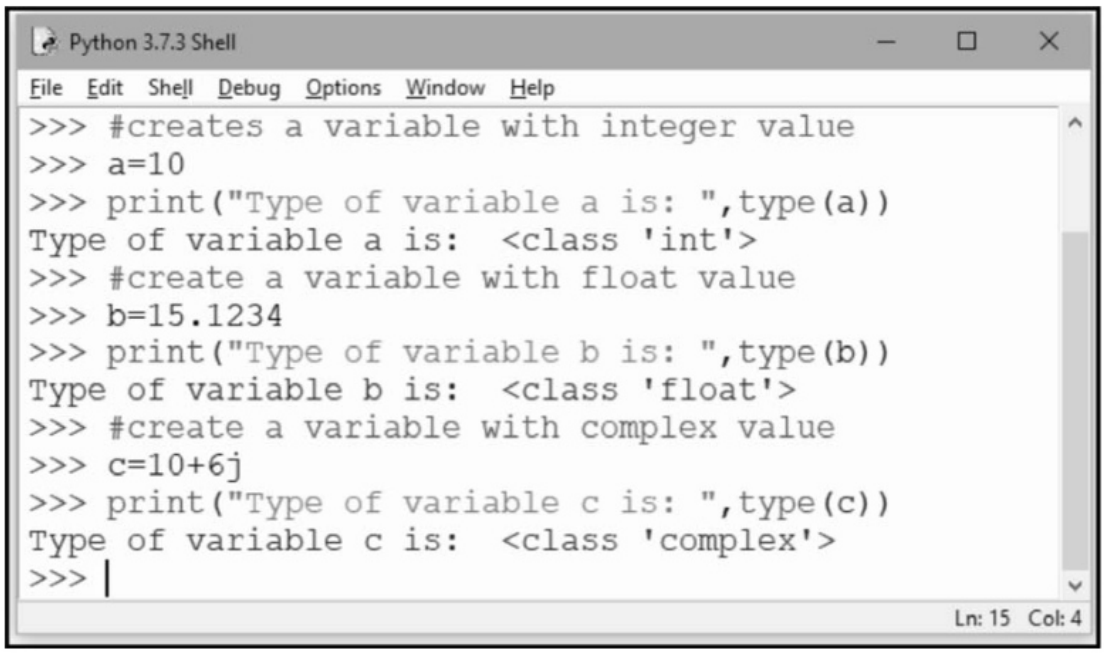


Fig. 1.5.1 Data types in Python

(1) Numeric : Python numeric data type is used to hold numeric values like;

- int – holds signed integers of non-limited length.
- long- holds long integers
- float- holds floating precision numbers and it's accurate upto 15 decimal places.
- complex- holds complex numbers.

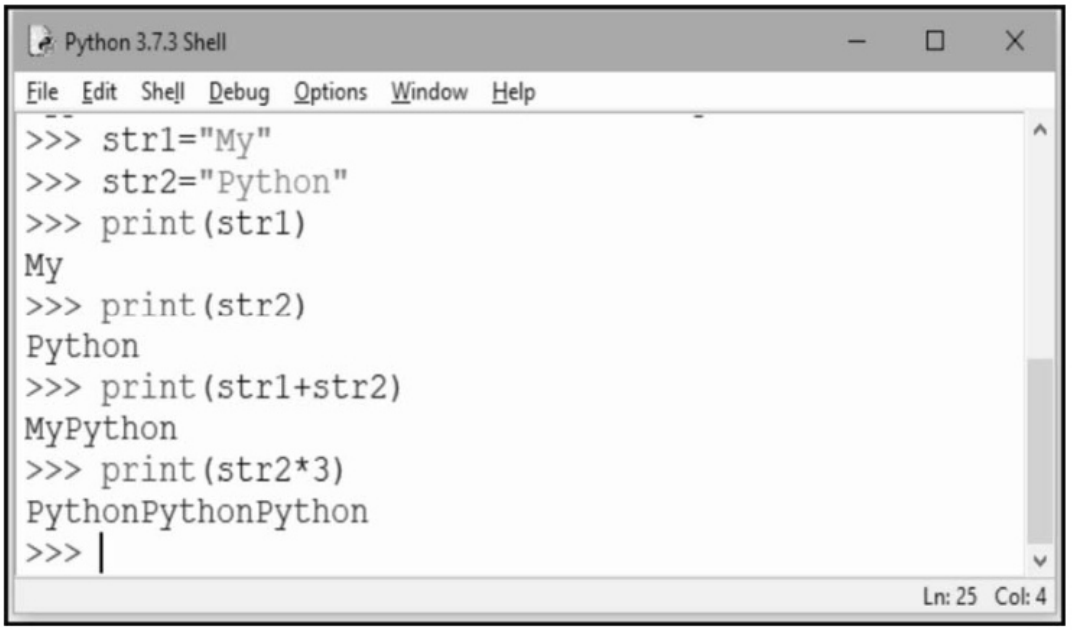
In Python we need not to declare datatype while declaring a variable like C or C++. We can simply just assign values in a variable. But if we want to see what type of numerical value is it holding right now, we can use **type()**. For example -



(2) String :

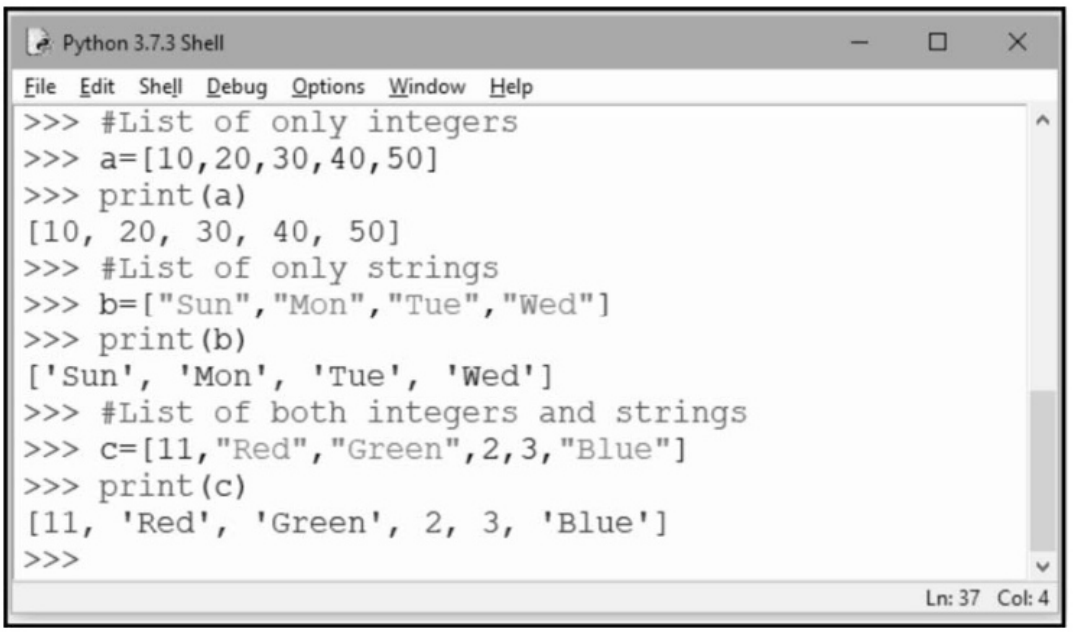
- String is a collection of characters.
- In Python, we can use single quote, double quote or triple quote to define a string.
- We can use two operators along with the string one is + and another is *.

- The + operator is used to concatenate the two strings. While * operator is used as a repetition operation. Following execution illustrates it -



(3) List :

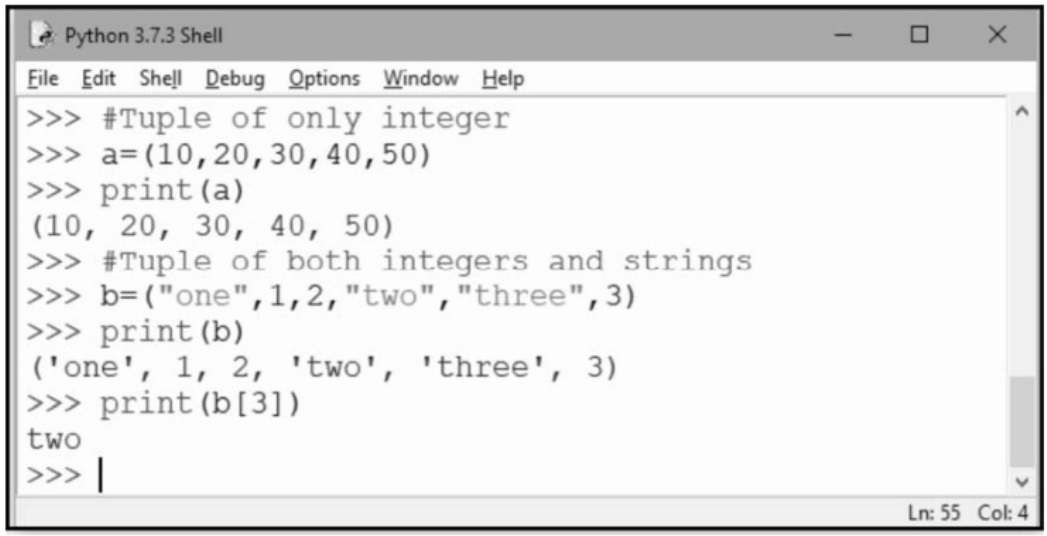
- It is similar to array in C or C++ but it can simultaneously hold different types of data in list.
- It is basically an ordered sequence of some data written using square brackets([]) and commas(,)
- For example -



(4) Tuple :

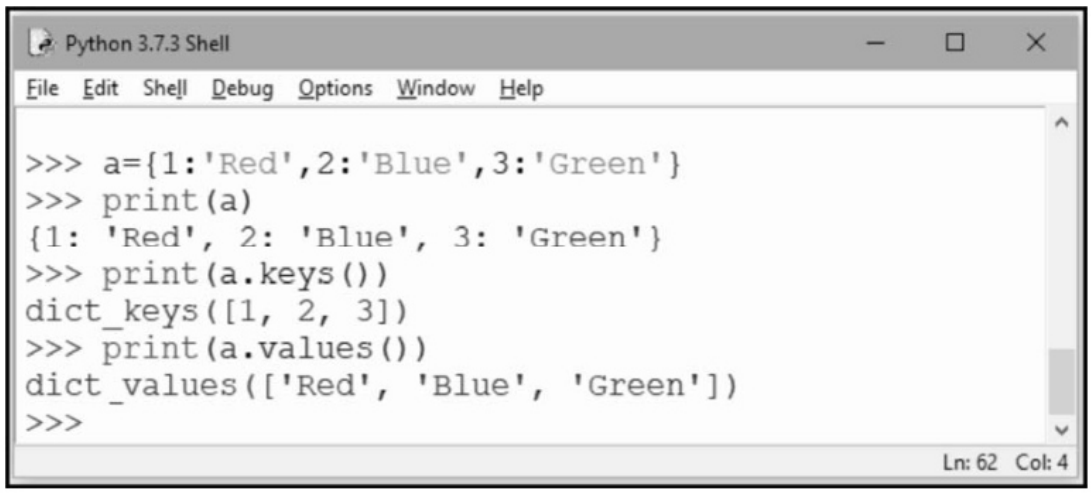
- Tuple is a collection of elements and it is similar to the List. But the items of the tuple are separated by comma and the elements are enclosed in ( ) parenthesis.

- Tuples are immutable or read-only. That means we can not modify the size of tuple or we cannot change the value of items of the tuple.
- Here are examples of tuples -



(5) Dictionary

- Dictionary is a collection of elements in the form of **key:value** pair.
- The elements of dictionary are present in the curly brackets.
- For example –



Review Question

1. *List and Explain different data types used in Python.*

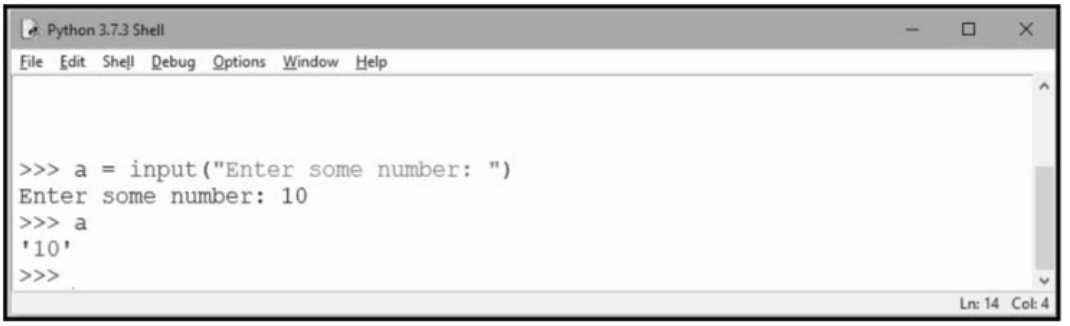
1.6 Input through Keyboard

- In python it is possible to input the data using keyboard.
- For that purpose, the function **input()** is used.
- **Syntax**

```
input([prompt])
```

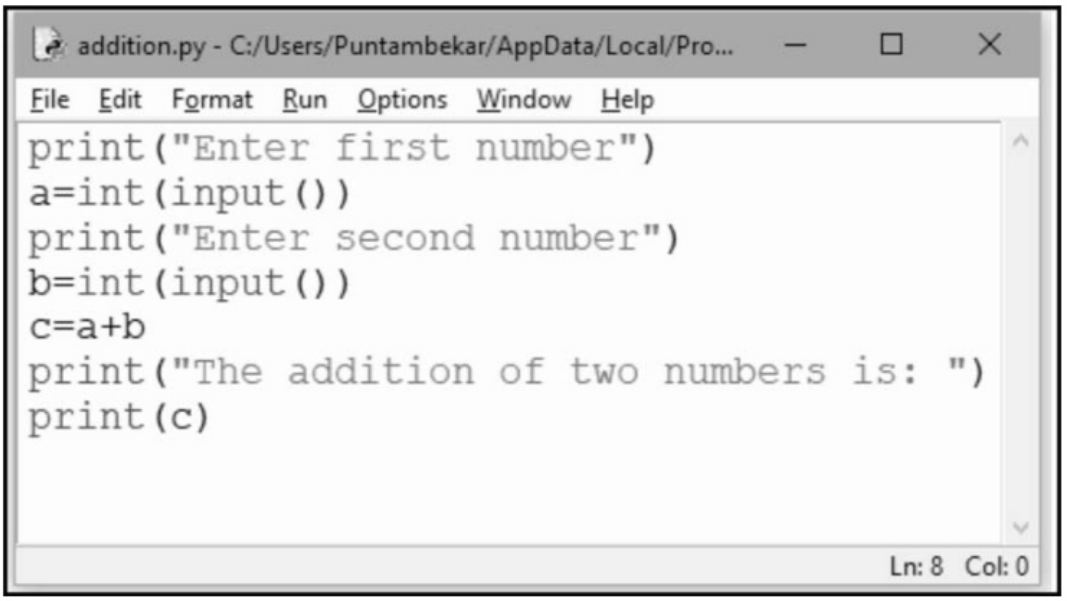
where prompt is the string we wish to display on the screen. It is optional.

Example 1.6.1 : *In the following screenshot, the input method is used to get the data*



Example 1.6.2 : *Write a python program to perform addition of two numbers. Accept the two numbers using keyboard*
Solution :

addition.py



Output

For getting the output click on **Run-> Run Module** or press F5 key, following shell window will appear -

Program Explanation :

- In above program, we have used **input()** function to get the input through keyboard. But this input will be accepted in the form of string.
- For performing addition of two numbers we need numerical values and not the strings. Hence we use **int()** function to which the **input()** function is passed as parameter. Due to which whatever we accept through keyboard will be converted to integer.
- Finally the addition of two numbers as a result will be displayed.
- The above program is run using F5 and on the shell window the messages for entering first and second numbers will be displayed so that user can enter the numbers.

Example 1.6.3 : *Write a Python program to find the square root of a given number*
Solution:

SqrtDemo.py

```
print("Enter the number:")
num=float(input())
result=num**0.5
print("The sqaure root of ",num," is ",result)
```

Output

```
Enter the number:
25
The sqaure root of  25.0  is  5.0
>>>
```

Example 1.6.4 : *Write a program in Python to obtain principle amount, rate of interest and time from user and compute simple interest.*

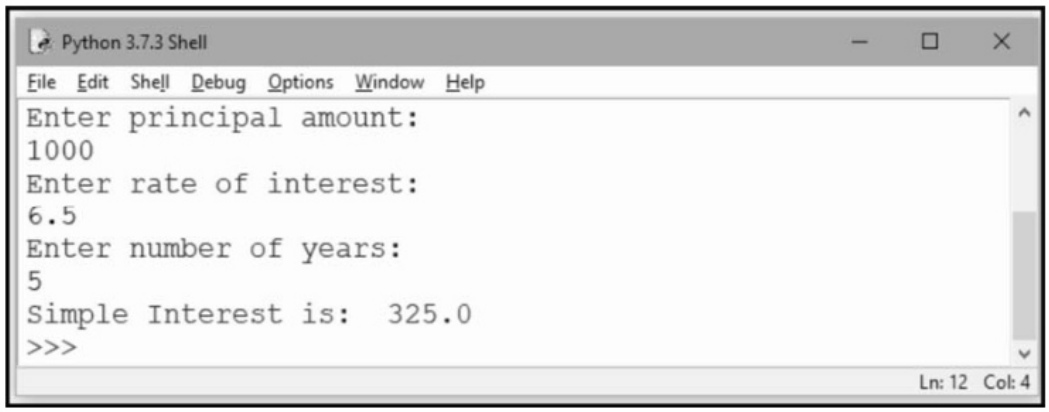
Solution :

Interest.py

```
print("Enter principal amount: ")
p=float(input())
print("Enter rate of interest: ")
r=float(input())
print("Enter number of years: ")
n=float(input())
I=(p*n*r)/100
print("Simple Interest is: ",I)    # Output will be displayed on console
```

Here we are reading the values through keyboard. Note we are reading the values as float

Output



❑❑❑

2 Python Operators and Control Flow Statements

2.1 Basic Operators

- Operands are special symbols that are used in computations. For example +, -, * and / are used for performing arithmetic operations.
- The values that operator uses for performing computation are called operands.

Various operators used in Python are described as follows -

2.1.1 Arithmetic Operators

These operators are used for performing arithmetic operations.

Operator	Meaning	Example
+	Addition operator is used for performing addition of two numbers.	10 + 20 = 30
–	Subtraction operator is used for performing subtraction.	20 – 10 = 10
*	Multiplication operator is used for performing multiplication.	10 * 10 = 100
/	Division operator is used for performing division of two numbers.	10/2 = 5
%	Mod operator returns the remainder value.	10%2 = 0
**	This is an exponentiation operator(power).	2**3 = 8
//	This is floor division operator. In this operation the result is the quotient in which the digits after the decimal point are removed.	10//3 = 3

For example

```
>>> 10+20
30
>>> 20-10
10
>>> 10/2
5.0
>>> 5*10
50
>>> 2**3
8
>>> 9//2
4
>>>
```

2.1.2 Comparison / Relational Operators

The operators compare the values and establish the relationship among them.

Operator	Meaning
==	If two values are equal then condition becomes true
!=	If two operands are not equal then the condition becomes true
<>	This is similar to !=. That means if two values are not equal then it returns true.
<	This is less than operator. If left operand is less than the right operator then the return value is true
>	This is greater than operator. If left operand is greater than the right operator then the return value is true
<=	This is less than equal to operator. If left operand is less than the right operator or equal to the right operator then the return value is true
>=	This is greater than equal to operator. If left operand is less than the right operator or equal to the right operator then the return value is true

For example

```
>>> 10<20
True
>>> 10<=10
True
>>> 20>10
True
>>> 20>=10
True
>>>
```

2.1.3 Assignment Operators

The assignment operator is used to assign the values to variables. Following is a list of assignment operators.

Operator	Example and Meaning
=	This is an operator using which values is assigned to a variable a = 5
+=	a+=5 means a=a+5
–=	a–=5 means a=a–5

Similarly *=, /= operators are used for performing arithmetic multiplication and division operation.

2.1.4 Logical Operators

There are three types of logical operators and, or, not

Operator	Meaning	Example
and	If both the operands are true then the entire expression is true.	a and b
or	If either first or second operand is true.	a or b
not	If the operand is false then the entire expression is true.	not a

For example

```
>>> a=True
>>> b=False
>>> a and b
False
>>> a or b
True
>>> not a
False
>>>
```

2.1.5 Bitwise Operator

Bitwise operators work on the bits of the given value. These bits are binary numbers i.e. 0 or 1.

For example : The number 2 is 010, 3 is 011.

Operator	Meaning	Example If a=010 , b=011
&	This is bitwise and operator.	a&b=010
|	This is bitwise or operator.	a|b=011
~	This is bitwise not operator.	~a=101
^	This is bitwise XOR operator. The binary XOR operation will always produce a 1 output if either of its inputs is 1 and will produce a 0 output if both of its inputs are 0 or 1.	a xor b = 001
<<	The left shift operator	a<<1 = 010<<1 means make left shift by one positions and add a tailing zero 010 100 =decimal 4
>>	The right shift operator	a>>1= 0101>>1 means make right shift by one position and add leading zero. 010 001 = decimal 1

2.1.6 Membership Operator

There are two types of membership operators – in and not in

These operators are used to find out whether a value is a member of a sequence such as string or list.

Operator	Description
in	It returns True if a sequence with specified value is present in the object.
not in	It returns true if a sequence with specified value is not present in the object.

Following screenshot of Python shell shows the use of in and not in operator.

Explanation :

1) In example, we have created a list of colors.
2) The members of color list are "red","blue" and "green"
3) As "blue" is member of color list, it returns **True,** while testing with membership operator **in** operator.
4) As "yellow" is not a member of color list, it return **False** for **in** operator and **True** for **not in** operator.

2.1.7 Identity Operator

The **'is'** operator returns true if both the operand point to same memory location. Similarly **'is not'** operator returns true if both the operand point to different memory location.

Similarly,

```
>>> print(color1 is not color2)
True
```

Explanation : In above demonstration,

1) We have created two lists color1 and color2.
2) Although the contents of the lists are exactly the same, their memory locations are different. Hence color1 is color2 becomes False.

3) As color3 is a new variable to which we assign color1, then they point to same memory location. Hence color1 is color3 returns True.

2.1.8 Modulus and Floor Division Operator

The % operator is a modulo operator that gives the remainder from the division of first argument by second.

For example

```
>>> 10%3
1
>>> 10.10%3.3
0.20000000000000018
>>>
```

The **operator // is used for floor division**. This division returns the integral part of the quotient.

For example

```
>>> 10//3.5
2.0
```

2.1.9 Python Operator Precedence

- When more than one operator appears in an expression, the order of evaluation depends on the rules of precedence.
- Python follows the same precedence rules for its mathematical operators that mathematics does.
- The acronym **PEMDAS** is a useful way to remember the order of operations.
    1. **P :** Parentheses have the highest precedence and can be used to force an expression to evaluate in the order you want. Since expressions in parentheses are evaluated first, 1 * (10-5) is 5
    2. **E :** Exponentiation has the next highest precedence, so 2**3 is 8.
    3. **MDAS : M**ultiplication and **D**ivision have the same precedence, which is higher than **A**ddition and **S**ubtraction, which also have the same precedence. So 2+3*4 yields 14 rather than 20.
    4. Operators with the same precedence are evaluated from left to right. So in the expression 3-2+1 will result 2. As subtraction is performed first and then addition will be performed.

2.1.10 Modulus Operator

The % operator is a modulo operator that gives the remainder from the division of first argument by second.

For example

```
>>> 10%3
1
>>> 10.10%3.3
0.20000000000000018
>>>
```

The **operator // is used for floor division**. This division returns the integral part of the quotient.

For example

```
>>> 10//3.5
2.0
```

2.1.11 Programs using Operators

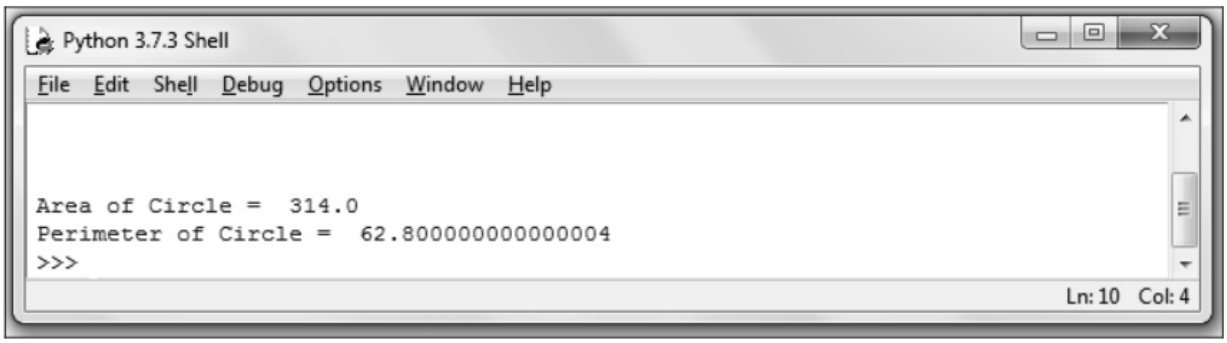
Example 2.1.1 : Write a program in Python to print area and perimeter of a circle.

Solution :

areaDemo.py

```
r=10
PI = 3.14
area= PI*r*r
perimeter= 2.0*PI*r
print("Area of Circle = ",area);
print("Perimeter of Circle = ",perimeter)
```

Output



Example 2.1.2 : Write a program to convert Fahrenheit to Celsius.

Solution : We will use the following formula for conversion

Celsius = (Fahrenheit – 32)/1.8

Temp.py

```
f=95.5
c= (f-32)/1.8
print("Celsius = ",c)
```

Output

```
Celsius =  35.2777
```

Review Questions

1. *Explain arithmetic and logical operators in Python.*
2. *Explain bitwise and relational operators in Python.*
3. *What is modulus operator ?*

2.2 Control Flow

The control flow statements are of three types –

(1) Conditional Statements (2) Loop Statements and (3) Loop Manipulation Statement

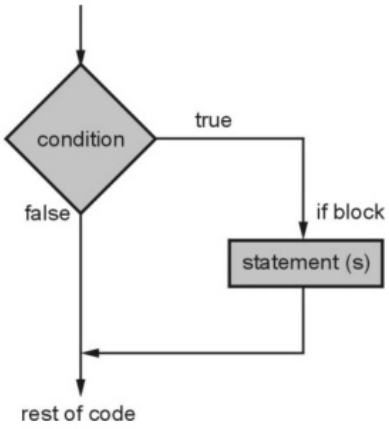
2.3 Conditional Statements

Various types of conditional statements used in Python are -

1. if statement
2. if-else statement
3. Nested if statement
4. If-elif-else statement

1. If Statement

The if statement is used to test particular condition. If the condition is true then it executes the block of statements which is called as **if block.** The if statement is the simplest form of the conditional statement.



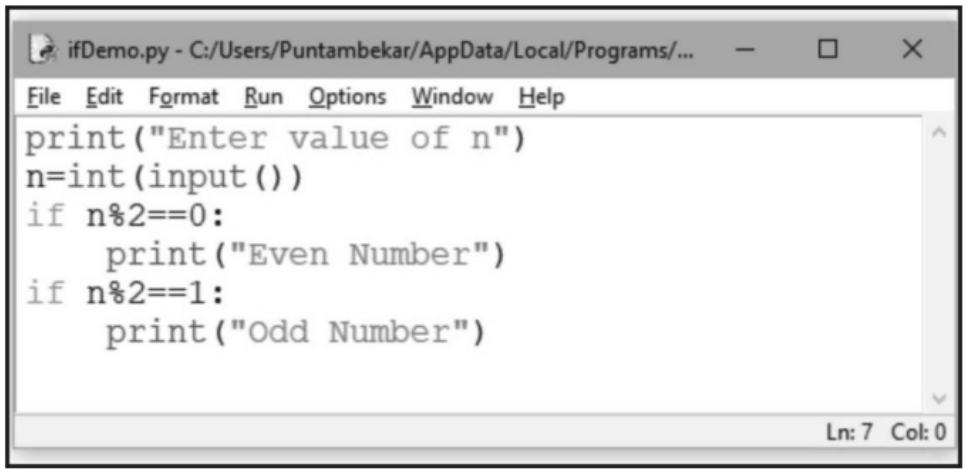
Syntax :

```
if condition:
        statement
```

Example:

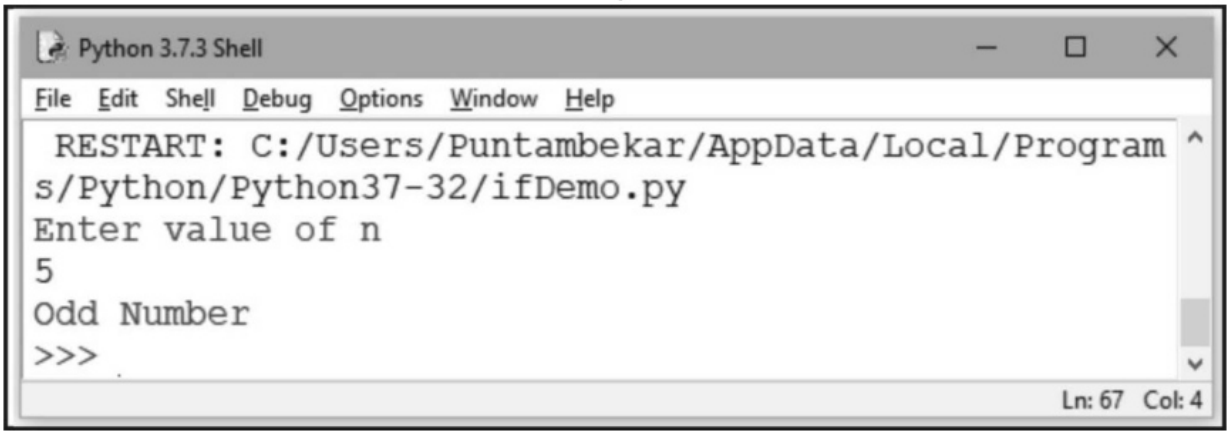
```
if a<10:
    print("The number is less than 10")
```

Example 2.3.1 : Write a Python program to check whether given number is even or odd.
Solution :



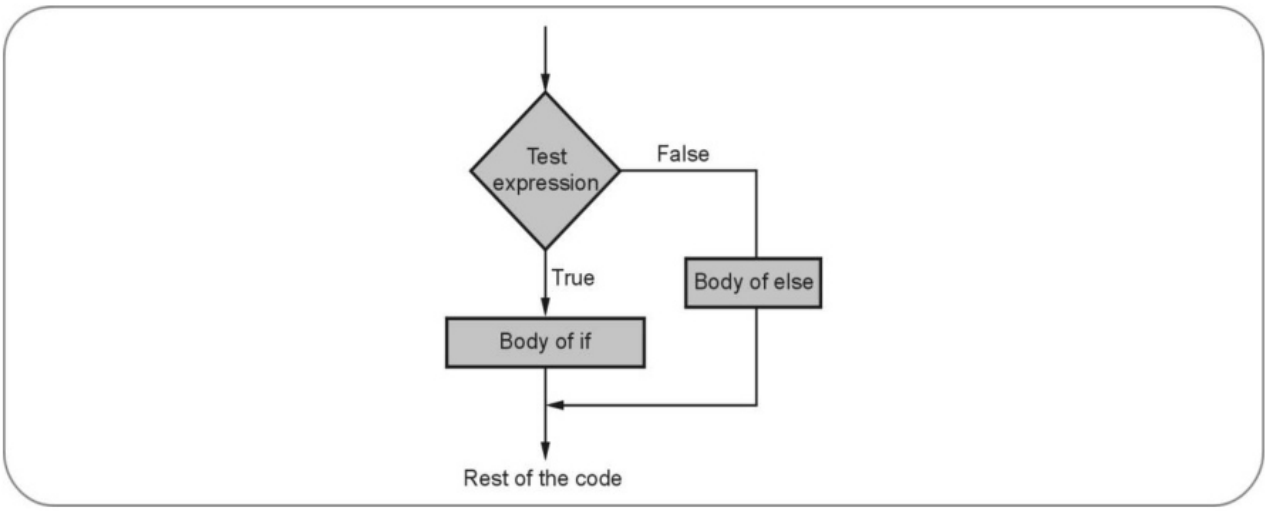

```
print("Enter value of n")
n=int(input())
if n%2 == 0:
    print("Even Number")
if n%2= 1:
    print("Odd Number")
```

Output



2. If-else

- The if-else statement provides an else block combined with the if statement which is executed in the false case of the condition. The flowchart for if-else is



Syntax

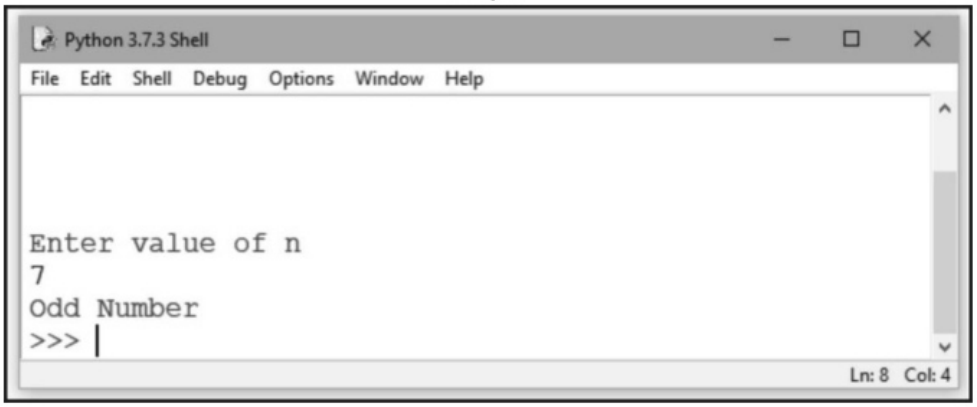
```
if condition :
        statement
else :
        statement
```

- If the condition is true, then the if-block is executed. Otherwise, the else-block is executed.

ifelseDemo.py

```
print("Enter value of n")
n=int(input())
if n%2==0:
    print("Even Number")
else:
    print("Odd Number")
```

Output



3. Nested if

When one if condition is present inside another if then it is called nested if conditions. Any number of these statements can be nested inside one another. Indentation is the only way to figure out the level of nesting.

Example 2.3.2 : Write a python program to compare two numbers using nested condition.
Solution :

NestedIfDemo.py

```
print("Enter value of a")
a=int(input())
print("Enter value of b")
b=int(input())
if a==b:
    print("Both the numbers are equal")
else:
    if a<b:
    print("a is less than b")
else:
    if a>b:
    print("a is greater than b")
```

Nested if-else

Output

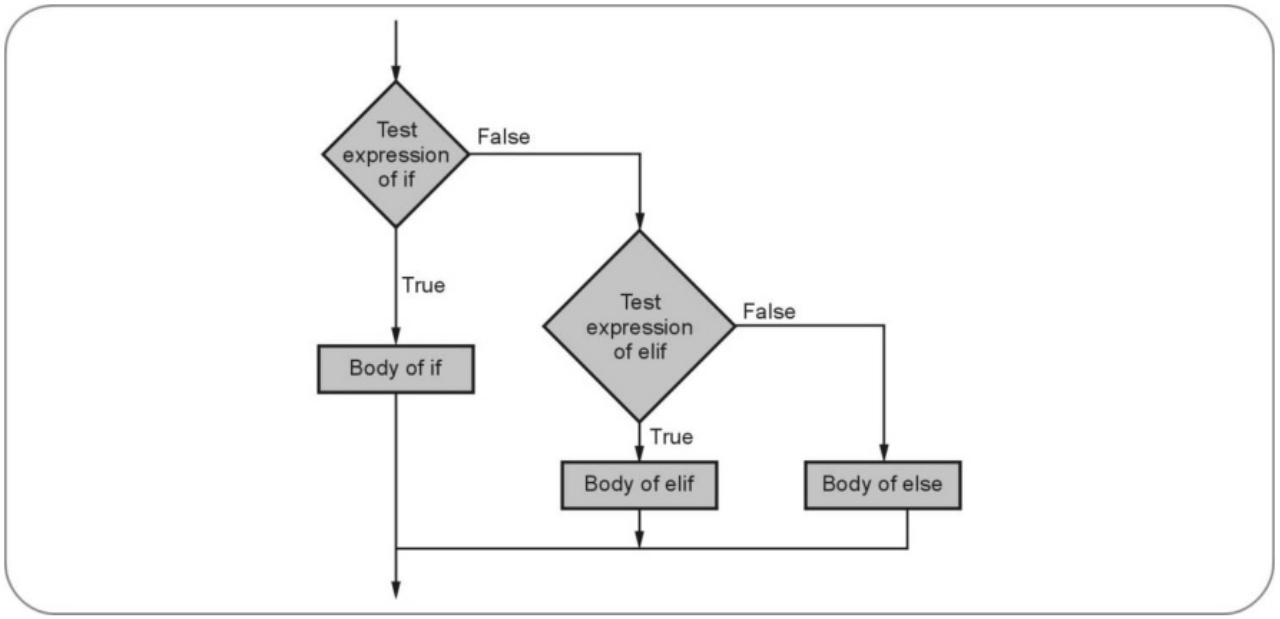
```
Enter value of a
20
Enter value of b
10
a is greater than b
>>>
```

4. If-elif-else Statement

- Sometimes there are more than two possibilities. These possibilities can be expressed using chained conditions. The syntax for this is as follows.

```
if condition:
    Statement
elif condition:
    Statement
...
else:
    Statement
```

- The chained conditional execution will be such that each condition is checked in order.
- The **elif** is basically abbreviation of **else if.**
- There is no limit on the number of elif statements.
- If there is **else** clause then it should be at the end.



- In chained execution, each condition is checked in order and if one of the condition is true then corresponding branch runs and then the statement ends. In this case if there are any remaining conditions then those condition won't be tested.

Example 2.3.3 : Write a python program to display the result such as distinction, first class, second class, pass or fail based on the marks entered by the user.

Solution :

```
print("Enter your marks")
m=int(input())
if m >= 75:
    print("Grade : Distinction")
elif m >= 60:
    print("Grade : First Class")
elif m >= 50:
    print("Grade : Second Class")
elif m >= 40:
    print("Grade : Pass Class")
else:
    print("Grade : Fail")
```

Output

Enter your marks

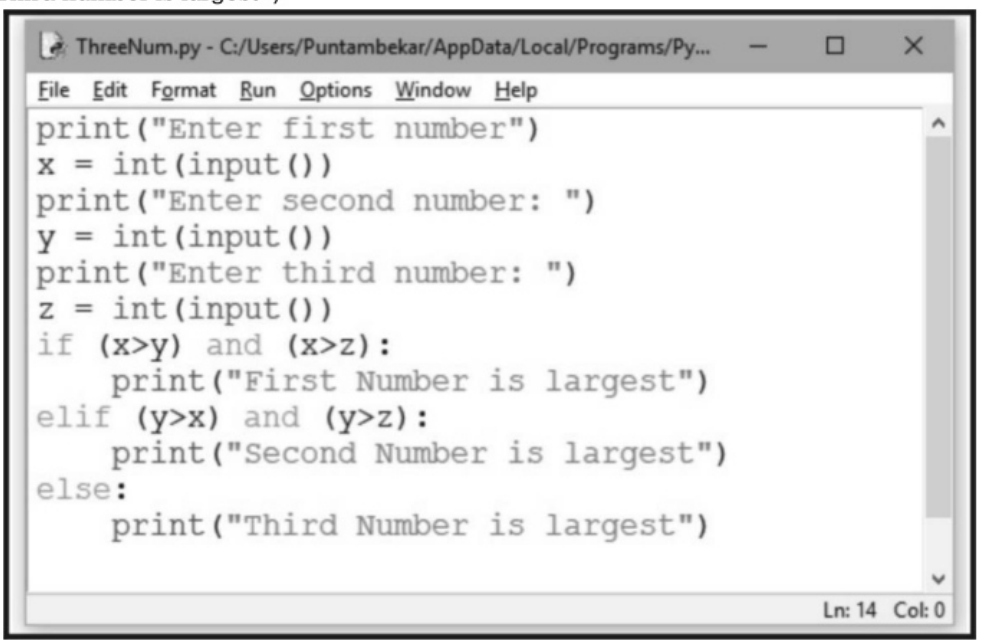
45

Grade : Pass Class

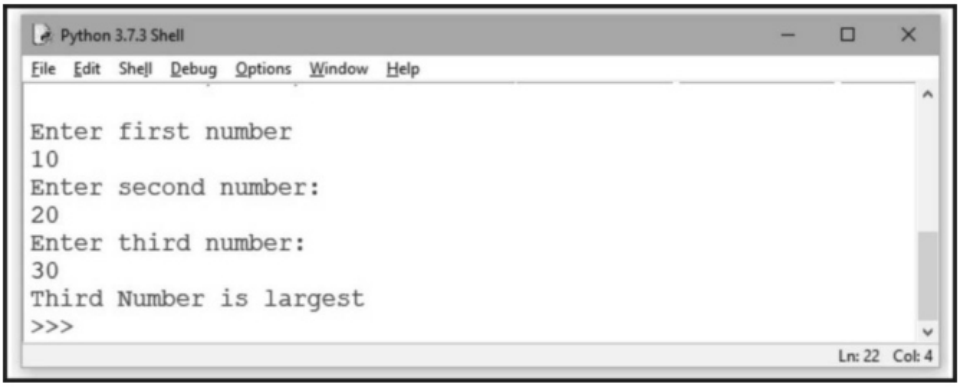
Example 2.3.4 : Write a python program to find the largest among the three numbers.

Solution :

```
print("Enter first number")
x = int(input())
print("Enter second number:")
y = int(input())
print("Enter third number:")
z=int(input())
if (x>y) and (x>z):
    print("First number is largest")
elif (y>x) and (y>z):
    print("Second number is largest")
else:
    print("Third number is largest")
```



Output



Review Question

1. *Explain selection statements in python with suitable examples.*

2.4 Looping in Python

- Looping is a technique that allows to execute a block of statements **repeatedly.**
- **Definition :** Repeated execution of a set of statements is called looping or iteration.
- The programming constructs used for iteration are while , for, break, continue and so on.
- Let us discuss the iteration techniques with the help of illustrative examples.

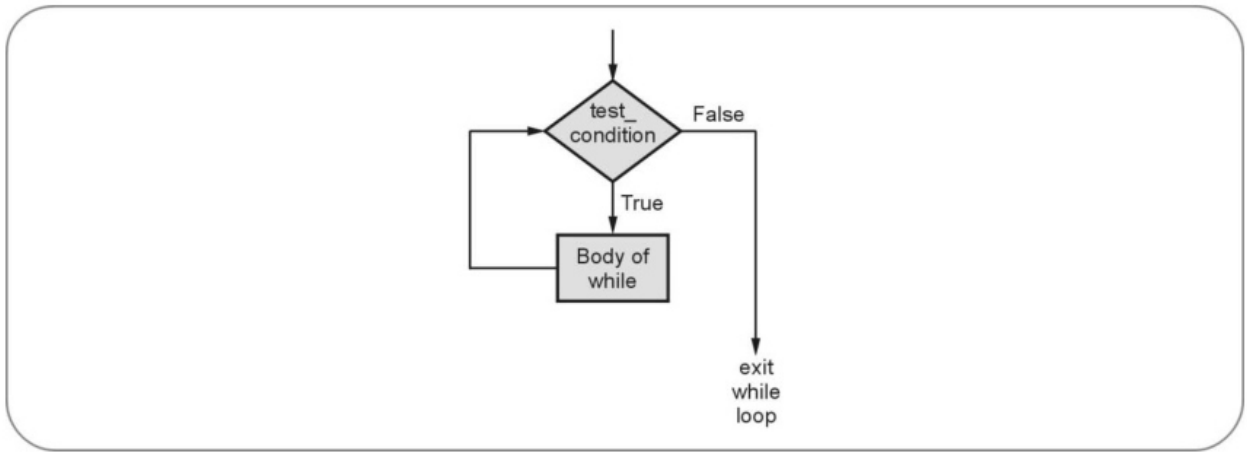
2.4.1 While Loop

The while statement is popularly used for representing iteration.

Syntax

```
while test_condition:
body of while
```

Flowchart for while statement is as given below



The flow of execution is specified as follows -

1. Using the condition determine if the given expression is true or false.
2. If the expression is false then exit the while statement
3. If the expression is true then execute the body of the while and go back to step 1 in which again the condition is checked.

For example

```
while i<=10:
    i=i+1
```

- The body of a while contains the statement which will change the value of the variable used in the test condition. Hence finally after performing definite number of iterations, the test condition gets false and the control exits the while loop.
- If the condition never gets false, then the while body executes for infinite times. Then in this case, such while loop is called **infinite loop.**

while...else

- There is another version of while statement in which **else** is used.

Syntax

```
while test_condition:
        body of while
else:
        statement
```

Example

```
while i<=10:
        i=i+1
else:
        print("Invalid value of i")
```

Programming examples on while

Example 2.4.1 : Write a python program for computing the sum of n natural numbers and finding out the average of it.

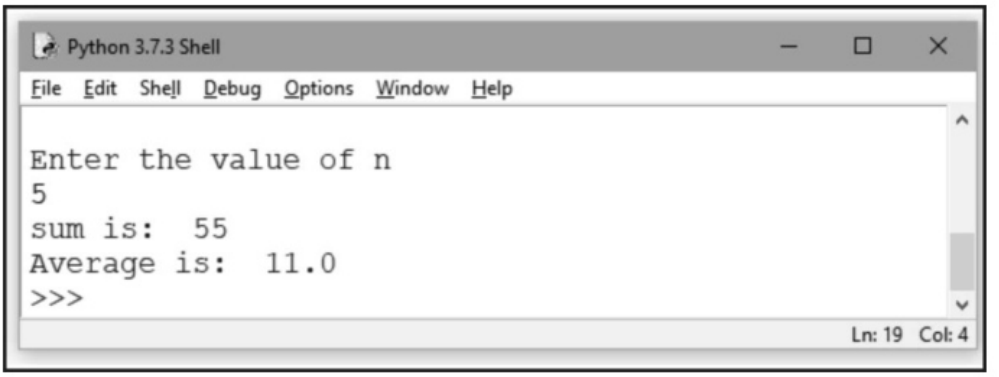
Solution : The python script is as given

whileDemo.py

```
print("Enter the value of n")
n=int(input())
sum = 0
avg = 0.0
i=1
while i<=10:
   sum = sum+i
   i=i+1

print("sum is: ",sum)
avg = sum/n
print("Average is: ",avg)
```

Output

This program can be run by pressing F5 key and following output can be obtained.



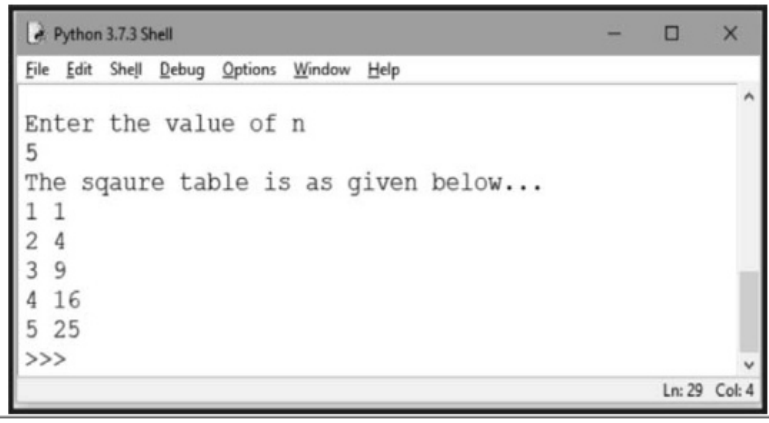
Example 2.4.2 : Write a python program to display square of n numbers using while loop.

Solution :

Sqtable.py

```
print("Enter the value of n")
n=int(input())
i=1
print("The sqaure table is as given below...")
while i<=n:
    print(i,i*i)
    i=i+1
```

Output



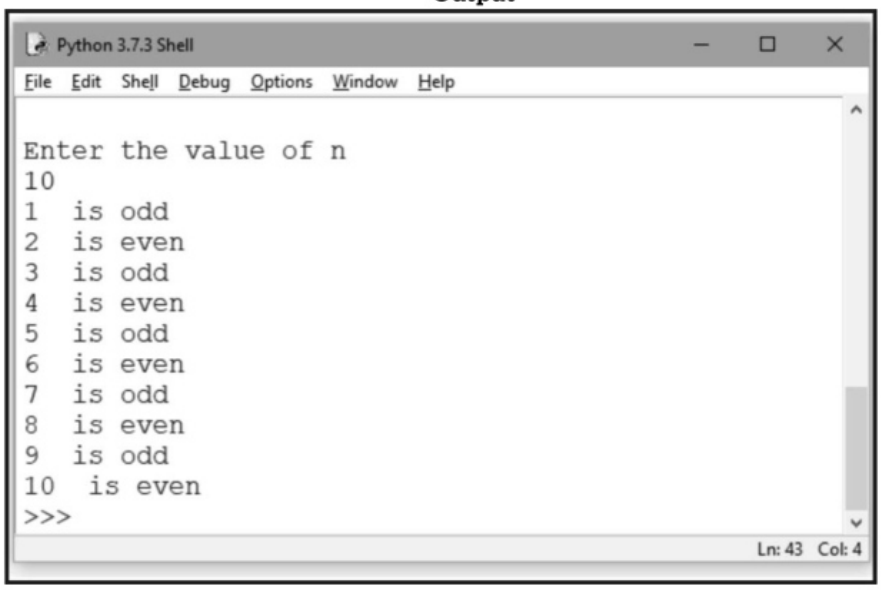
Example 2.4.3 : Write a python program for displaying even or odd numbers between 1 to n.

Solution :

oddevenDemo.py

```
print("Enter the value of n")
n=int(input())
i=1
j=1
while j<=n:
      if i%2==0:
      print(i," is even")
else:
      print(i," is odd")
j=j+1
i=j
```

Output

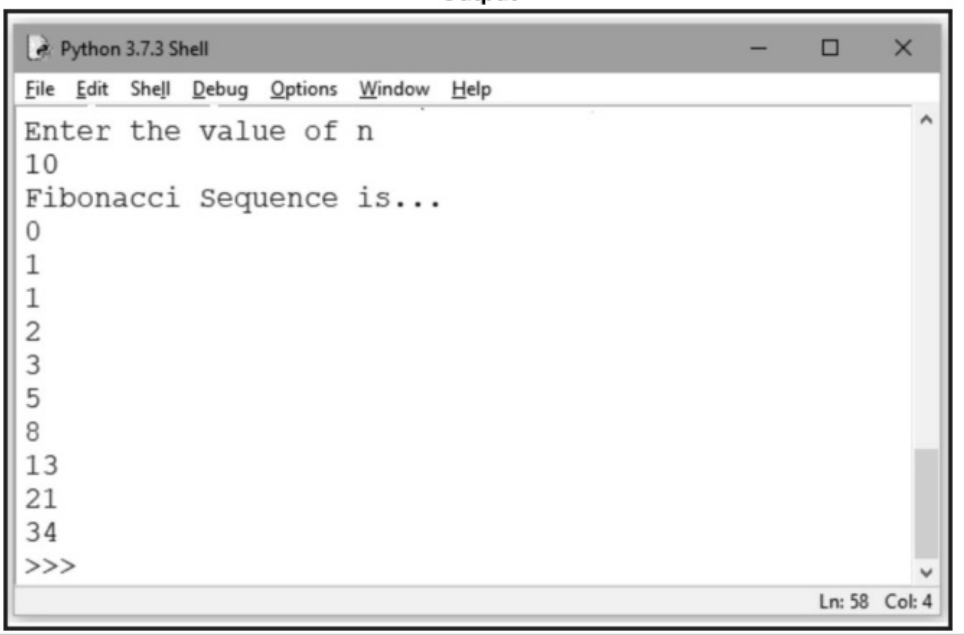


Example 2.4.4 : Write a python program to display fibonacci numbers for the sequence of length n.

Solution :

```
print("Enter the value of n")
n=int(input())
a=0
b=1
i=0
print("Fibonacci Sequence is...")
while i<n:
      print(a)
      c=a+b
      a=b
      b=c
      i=i+1
```

Output



Example 2.4.5 : To accept student's five courses marks and compute his/her result. Student is passing if he/she scores marks equal to and above 40 in each course. If student scores aggregate greater than 75 %, then the grade is distinction. If aggregate is 60 >= and < 75 then the grade is first division. If aggregate is 50 >= and <60 then the grade is second division. If aggregate is 40 >= and < 50 then the grade is third division.

Solution :

```
print("Enter the marks in English: ")
marks1=int(input())
print("Enter the marks in Mathematics: ")
marks2=int(input())
print("Enter the marks in Science: ")
marks3=int(input())
print("Enter the marks in Social Studies: ")
marks4=int(input())
print("Enter the marks in Computer Science: ")
marks5=int(input())
if(marks1<40 or marks2<40 or marks3<40 or marks4<40 or marks5<40 ):
    print("Fail")#minimum requirement for passing
else:
    total=marks1+marks2+marks3+marks4+marks5
    avg=float(total)/5 #computing aggregate marks
    print("The aggregate marks are: ",avg)
    if(avg>=75):
        print("Grade:Distinction")
    elif(avg >=60 and avg<75):
        print("Grade:First Division")
    elif(avg>=50 and avg<60):
        print("Grade:Second Division")
    elif(avg>=40 and avg<50):
        print("Grade:Third Division(pass)")
    else:
        print("Fail")
```

Output(Run1)

```
Enter the marks in English:
33
Enter the marks in Mathematics:
66
Enter the marks in Science:
55
Enter the marks in Social Studies:
66
Enter the marks in Computer Science:
44
Fail
>>>
```

Output(Run2)

```
Enter the marks in English:
67
Enter the marks in Mathematics:
89
Enter the marks in Science:
86
Enter the marks in Social Studies:
79
Enter the marks in Computer Science:
90
The aggregate marks are:  82.2
Grade:Distinction
>>>
```

Example 2.4.6 : To check whether input number is Armstrong number of not. An Armstrong number is an integer with three digits such that the sum of the cubes of its digits is equal to the number itself. For example : 371.

Solution :

```
print("Enter some number:")
num = int(input())
# initialize sum
sum = 0
# find the sum of the cube of each digit
temp = num
while (temp >0):
 digit = temp % 10
 sum = sum + (digit ** 3)
 temp //= 10
```

```
# display the result
if (num == sum):
 print(num,"is an Armstrong number")
else:
 print(num,"is not an Armstrong number")
```

Output(Run1)

```
Enter some number:
371
371 is an Armstrong number
>>>
```

Output(Run2)

```
Enter some number:
456
456 is not an Armstrong number
>>>
```

2.4.2 The for Loop

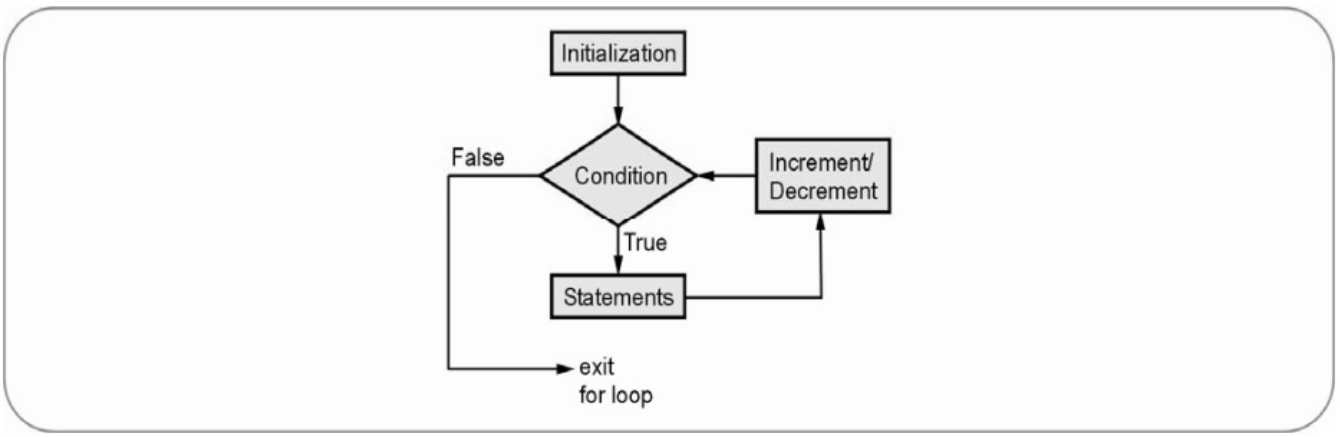
- The for loop is another popular way of using iteration. The syntax of for loop is
- **Syntax**

  ```
  for variable in sequence:
  Body of for loop
  ```

- **Example**

  ```
  for val in numbers:
  val=val+1
  ```

- The variable takes the value of the item inside the sequence on each iteration.
- Loop continues until we reach the last item in the sequence. The body of for loop is separated from the rest of the code using indentation.
- The flowchart is as shown below –



Programming examples on Loop

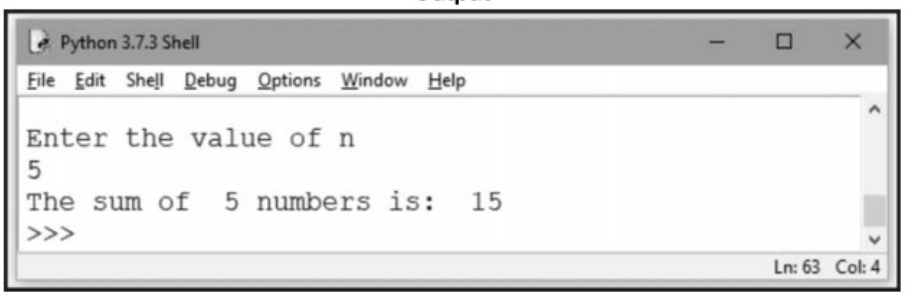
Example 2.4.7 : Write a python program to find the sum of 1 to 10 numbers.

Solution :

forDemo.py

```
print("Enter the value of n")
n=int(input())
sum=0
for i in range(1,n+1):
   sum=sum+i
print("The sum of ",n,"numbers is: ",sum)
```

Output

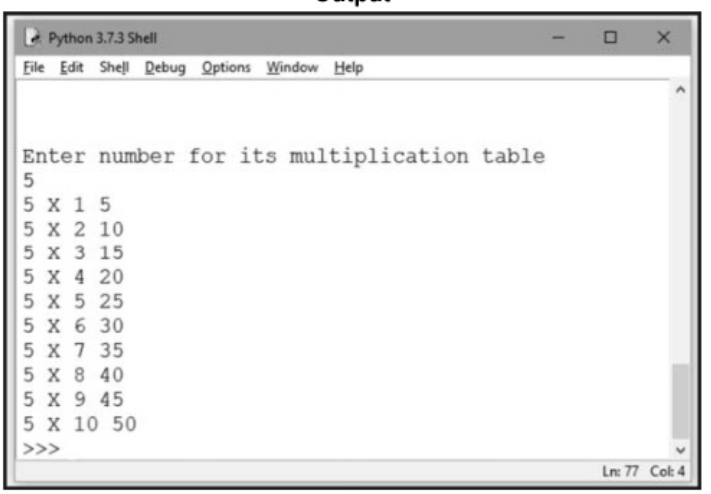


Example 2.4.8 : Write a python program to display the multiplication table.

Solution:

```
print("Enter number for its multiplication table")
n=int(input())
for i in range(1,11):
   print(n,"X",i,i*n)
```

Output



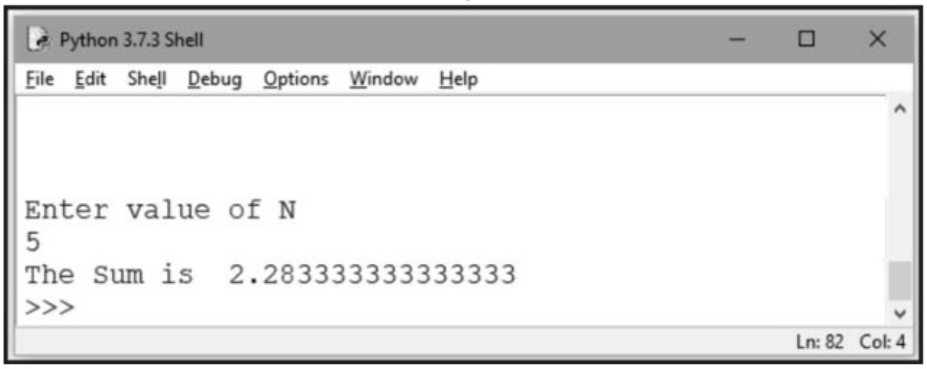
Example 2.4.9 : Write a program to print 1+1/2+1/3+1/4+ ..... +1/N series.

Solution :

SeriesDemo.py

```
print("Enter value of N")
n=int(input())
sum=0
for i in range(1,n+1):
   sum=sum+1/i
print("The Sum is ",sum)
```

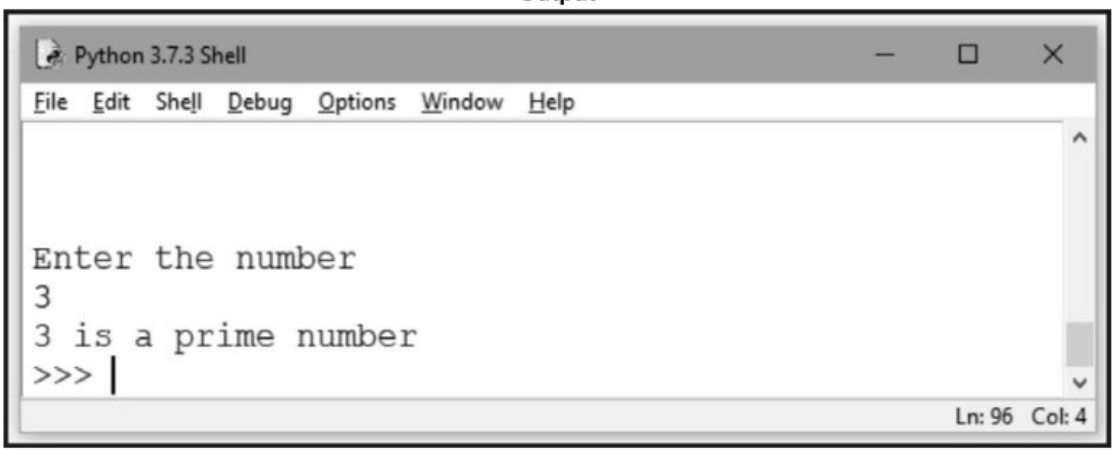
Output



Example 2.4.10 : Write a python program to check whether given number is prime or not.
Solution :

```
# User can enter the value thru keyboard
print("Enter the number");
num = int(input())

# 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



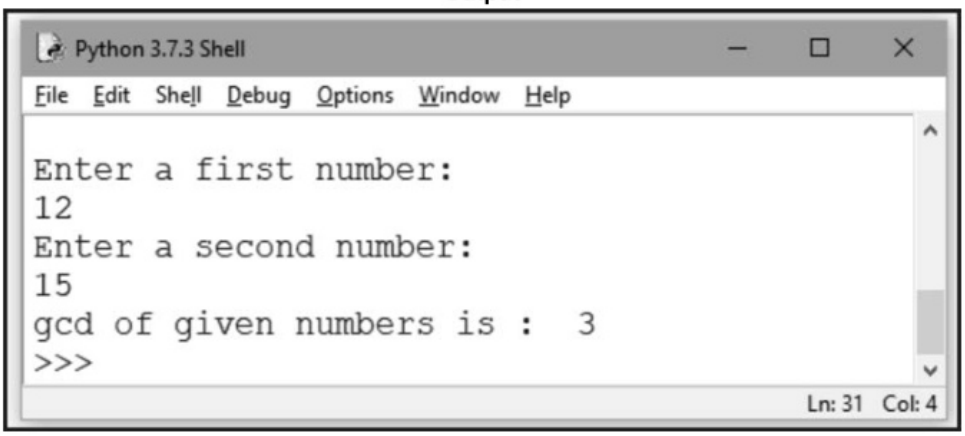
Example 2.4.11 : Write a python program for finding the GCD of two numbers.
Solution :

GCDDemo.py

```
print("Enter a first number:")
a=int(input())
```

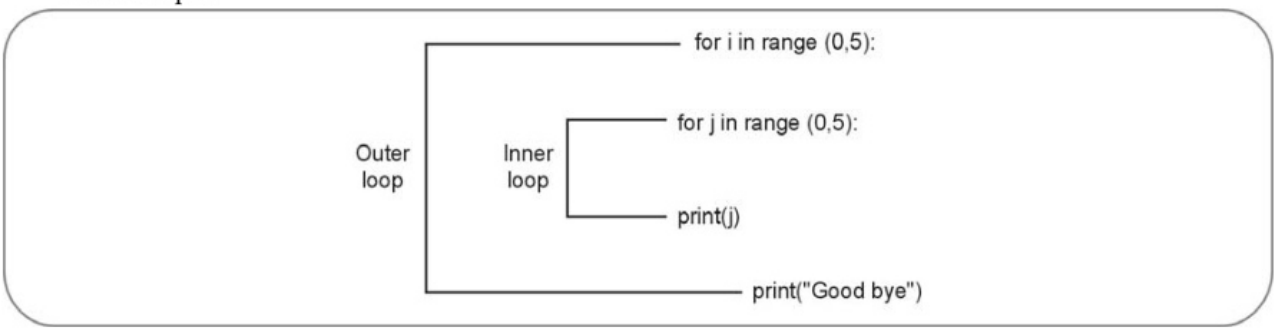
```
print("Enter a second number:")
b=int(input())
rem=a%b
while rem!=0:
   a=b
   b=rem
   rem=a%b
print ("gcd of given numbers is : ",b)
```

Output



2.4.3 Nested Loop

- Nested loops are the loop structure in which one loop is present inside the other loop.
- There can be nested for loops. That means one for loop is present inside the other for loop. In this case, the control first enters inside the outer loop then enters inside the inner for loop, it then executes inner loop completely and then switch back to outer for loop.
- For example -



- Following are some examples that shows use of nested loops.

Example 2.4.12 : Write a python program to display the star pattern as

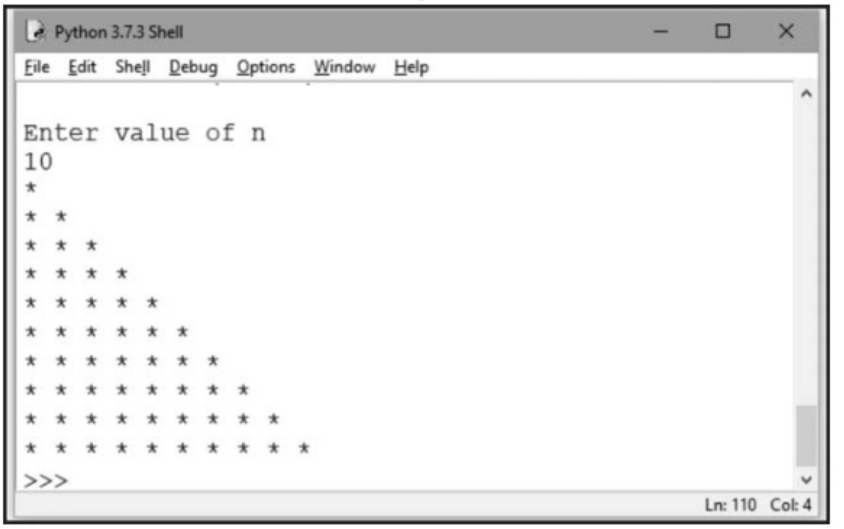
```
*
* *
* * *
* * * *
* * * * *
...
...
```

User should enter the value of N .

Solution :

```
print("Enter value of n")
n=int(input())
for i in range(0,n):
    for j in range(0,i+1):
        print('* ',end="")
    print("")
```

Output



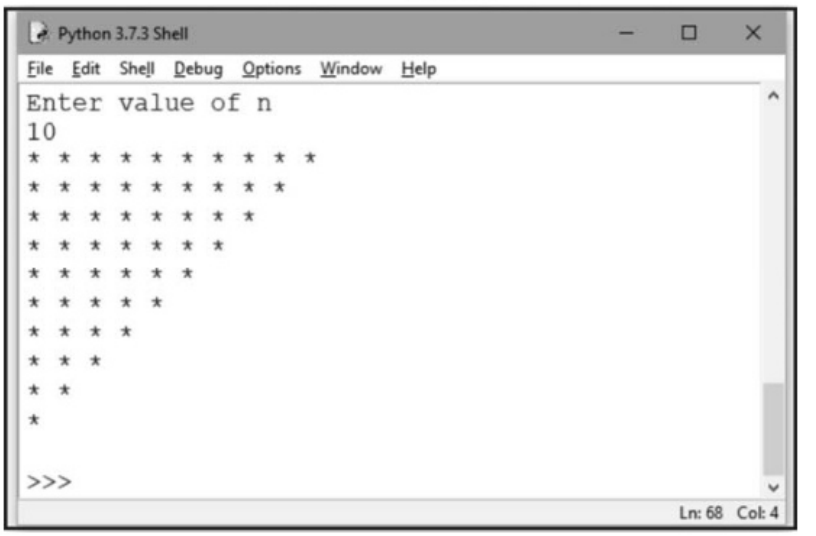
Example 2.4.13 : Write a python program to display a star pattern as

```
* * * * * * * * * *
* * * * * * * * *
* * * * * * * *
...
...
...
```

Solution :

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

Output



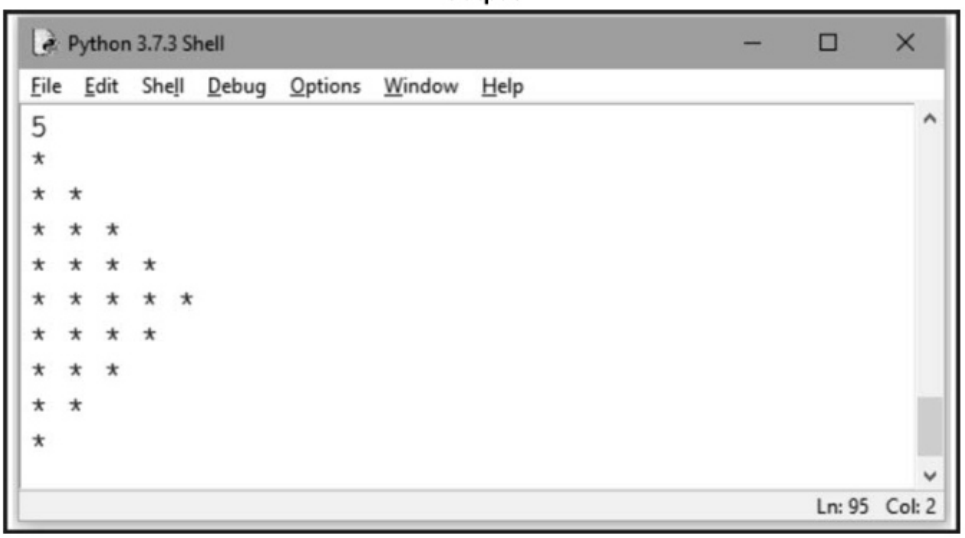
Example 2.4.14 : Write a python program to display the star pattern as

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

Solution :

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

Output



Example 2.4.15 : Write a python program to display the number pattern as follows

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

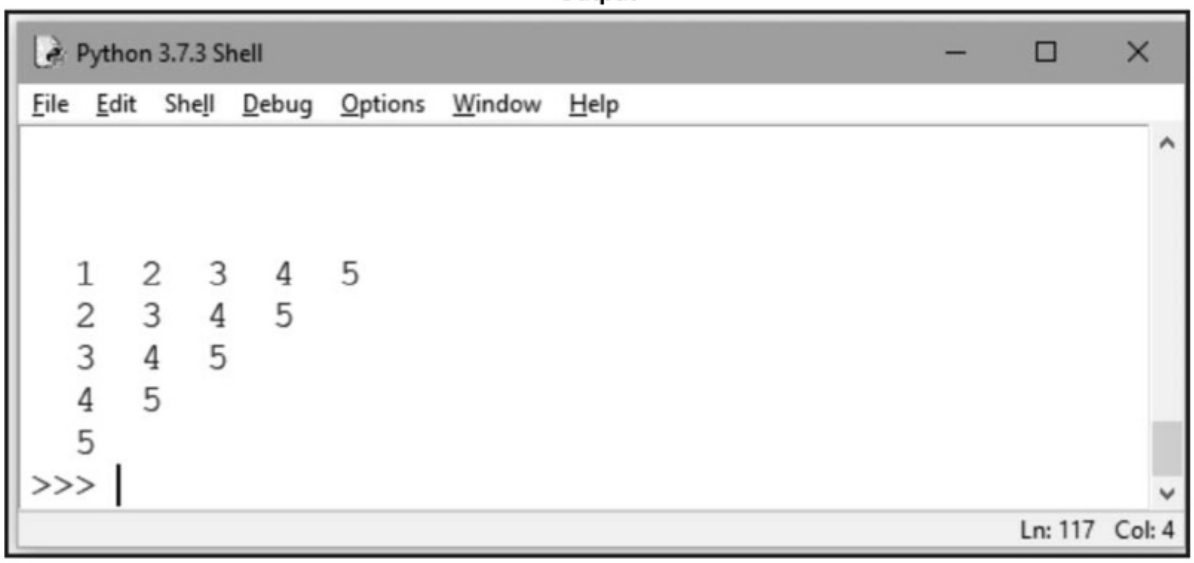
Solution :

Pattern2.py

```
for i in range (1,6):
    for k in range(1,i):
        print(end="")
```

```
for j in range(i,6):
    print(" ",j,end="")
print()
```

Output



Example 2.4.16 : Write a python program to print the pattern as given below

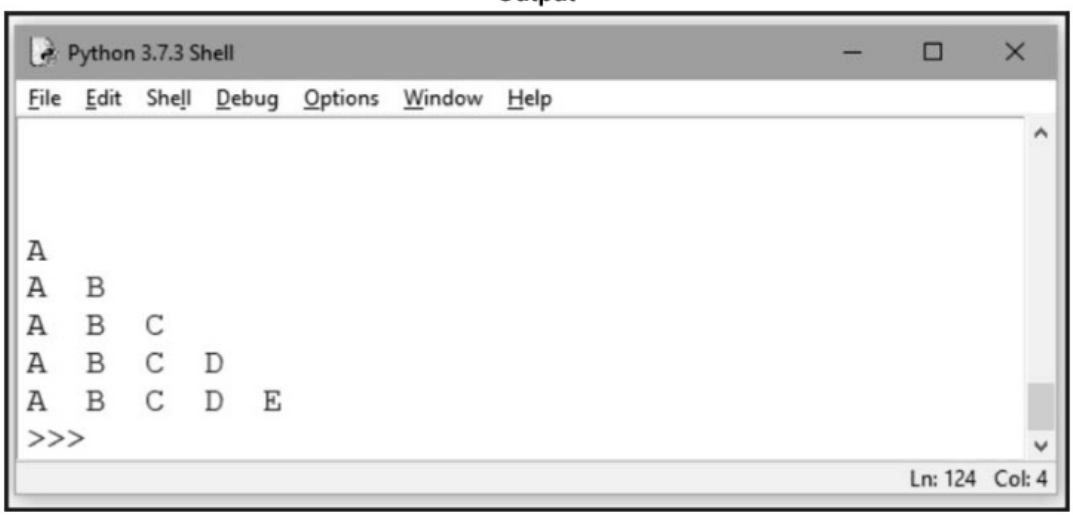
A
A B
A B C
A B C D
A B C D E

Solution :

Pattern3.py

```
for i in range(1, 6):
   for j in range(65, 65+i):
      a = chr(j)
      print(a," ",end="")
   print()
```

Output



Example 2.4.17 : Write a python program to print the pattern for alphabets.

```
A A A A A
B B B B
C C C
D D
E
```

Solution :

Pattern4.py

```
num=65 #ASCII Value for A
for i in range(0,5):
    for j in range(i,5):
        ch=chr(num+i) # Converting ASCII to character
        print(ch," ",end="")
    print();
```

Review Question

1. *Explain the nested loop with suitable example.*

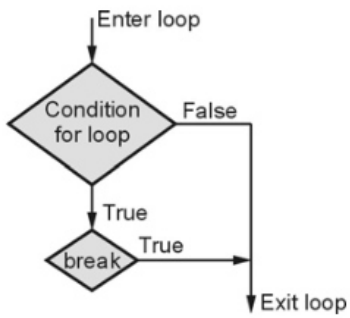
2.5 Loop Manipulation

In this section we will discuss

1. break	2. continue	3. Pass
4. Else with for loop		5. Else with while loop

(1) break

- The break statement is used to transfer the control to the end of the loop.
- When break statement is applied then loop gets terminates and the control goes to the next line pointing after loop body.
- The flowchart for break statement is



- **Syntax**

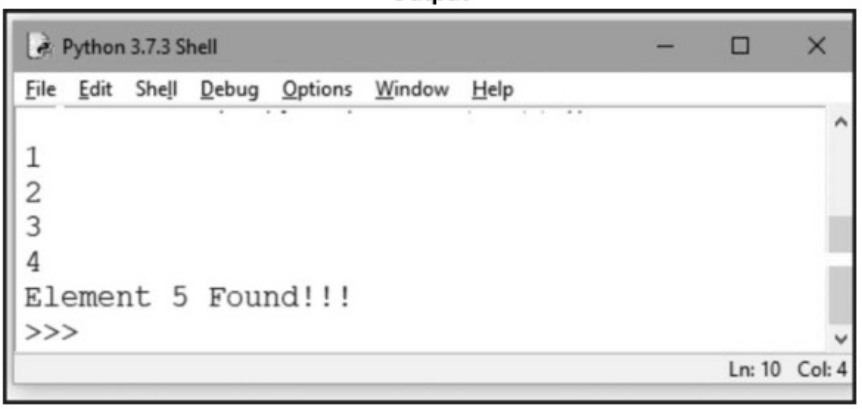
  break

- **For example**

```
for i in range(1,11):
    if i==5:
```

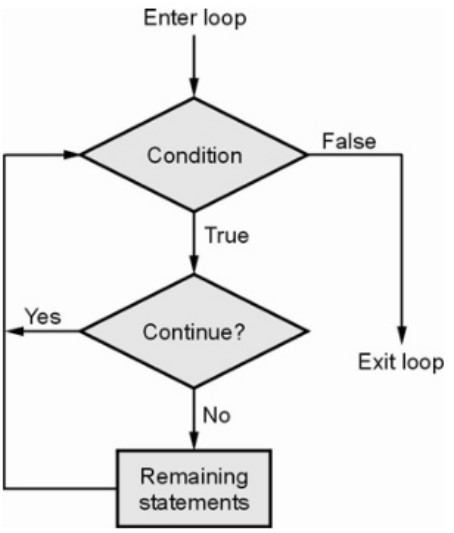
```
    print("Element {} Found!!!".format(i))
    break
print(i)
```

Output



(2) continue

- The continue statement is used to skip some statements inside the loop. The continue statement is used with decision making statement such as if...else.
- The continue statement forces to execute the next iteration of the loop to execute.
- The flowchart for continue statement is



- **Syntax**

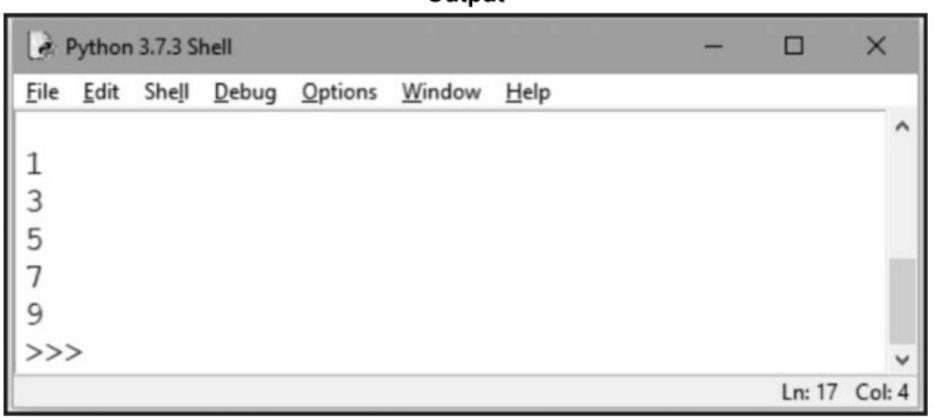
  continue

- **Example**

In while loop the continue passes the program control to conditional test. Following example illustrates the idea of continue

```
i=0
while i<10:
   i=i+1
   if i%2 ==0:
      continue
   print(i)
```

Output



In above program, if we find any even number(i.e. i%2 = 0) from 1 to 10 then just continue the loop that means simply increment i otherwise (i.e. if i%2 = 0 is false) then it should print the value of i. Thus we get all the odd numbers printed on the output screen.

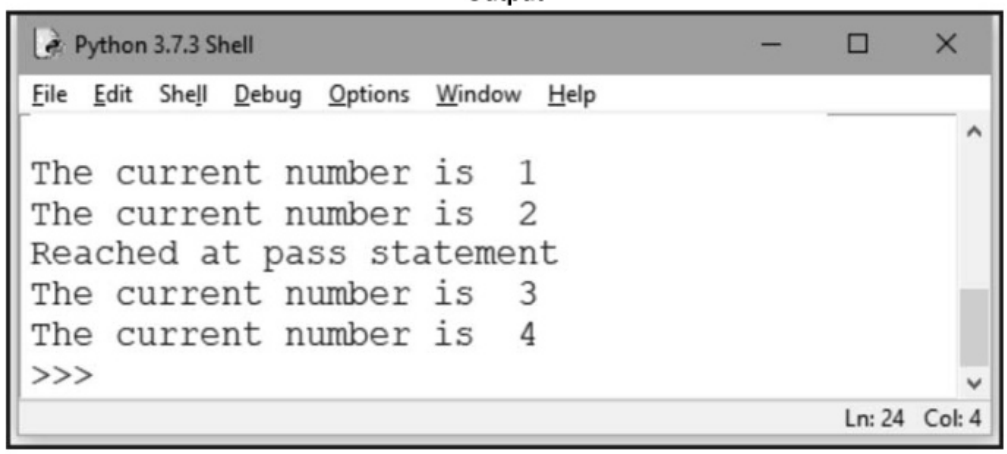
(3) pass

- The **pass** statement is used when we do not want to execute any statement. Thus the desired statements can be bypassed.
- **Syntax**

  Pass
- **Example**

```
for i in range(1,5):
   if i==3:
      pass
      print("Reached at pass statement")
   print("The current number is ",i)
```

Output



(4) else statement with loops

The else block is placed just after the for or while loop. It is executed only when the loop is not terminated by a break statement.

Else statement with for loop

The for loop can be written using else clause.

Syntax

```
for variable in sequence:
 Body of for loop
else:
 Statement
```

Programming example

Following are two scenarios in which the loop may end. The first one is in which break is encountered not encountered and the second scenario is that the loop ends with encountering a break statement.

```
for i in range(1,10):
 print(i)
else:
 print("There is no break statement")
```

Output

```
1
2
3
4
5
6
7
8
9
There is no break statement
```

```
for i in range(1,10):
 print(i)
 break
else:
 print("There exists break statement")
```

Output

```
1
```

In above case, In first case, else part is executed when the for loop condition turns out to be false. But in second column, we encounter a break statement in for loop and there by for loop terminates but the else part does not get executed here.

Else statement with while loop

- Similar to for loop we can have else block after while loop.
- The else part is executed if the condition in the while loop evaluates to false.
- The while loop can be terminated with a break statement. In such case, the else part is ignored. Hence, a while loop's else part runs if no break occurs and the condition is false.
- Here is an example to illustrate this.

Programming example

```
i=0
while i<3:
    print(i)
     i=i+1
else:
print("There is no break statement")
```

Output

```
0
1
2
There is no break statement
```

```
i=0
while i<3:
      print(i)
      i=i+1
      break
else:
 print("There is a break statement")
```

Output

```
0
```

Difference between break and continue

Sr.No.	break	continue
1.	This statement terminates the execution of remaining iteration of the loop.	It terminates only the current iteration of the loop.
2.	It causes early termination of the entire loop.	It causes early execution of the next iteration.

Review Questions

1. Explain loop statements for and while with illustrative example.

2. What is the difference between condition controlled and counter controlled loops ?

3. Explain the use of break and continue statements in Python.

❑❑❑

3 Data Structures in Python

3.1 List

3.1.1 Introduction to Lists

3.1.1.1 Definition of List

List is a sequence of values written in a square bracket and separated by commas. For example

How to crate List ?

```
>>>a=['AAA','BBB','CCC']
>>>b=[10,20,30,40]
>>>
```

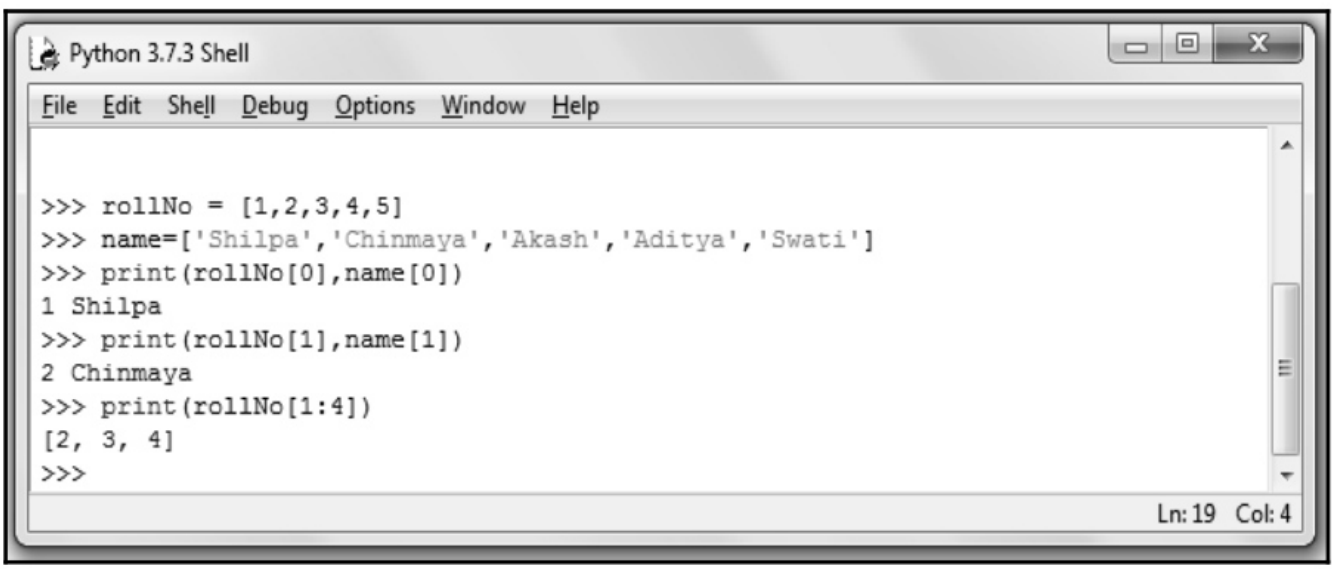
Here a and b are the lists data structures.

3.1.1.2 Accessing Values in List

Individual element of the list can be accessed using index of the list. For example -

```
>>> rollNo = [1,2,3,4,5]
>>> name=['Shilpa','Chinmaya','Akash','Aditya','Swati']
>>> print(rollNo[0],name[0])
1 Shilpa
>>> print(rollNo[1],name[1])
2 Chinmaya
>>> print(rollNo[1:4])
[2, 3, 4]
>>>
```

The above code can be illustrated on IDE as follows -



Using the **in** operator we can check whether particular element belongs to the list or not. If the given element is present in the list it returns true otherwise false. For example

```
>>> a = ['AAA', 'XXX', 'CCC']
>>> 'XXX' in a
True
>>> 'BBB' in a
False
>>>
```

3.1.1.3 Deleting Values in List

- The deletion of any element from the list is carried out using various functions like **pop, remove, del.**
- **The pop Function :** If we know the index of the element to be deleted then just pass that index as an argument to **pop** function.
  - For example

```
>>> a=['u','v','w','x','y','z']
>>> val=a.pop(1)  #the element at index 1 is v, it is deleted
>>> a
['u', 'w', 'x', 'y', 'z']      #list after deletion
>>> val               #deleted element is present in variable val
'v'
>>>
```

- If we do not provide any argument to the **pop function** then the **last element** of the list will be deleted.
  - For example

```
>>> a =['u','v','w','x','y','z']
>>> val=a.pop()
>>> a
['u', 'v', 'w', 'x', 'y']
>>> val
'z'
>>>
```

- **The remove function :** If we know the value of the element to be deleted then the **remove** function is used. That means the parameter passed to the remove function is the actual value that is to be removed from the list. Unlike, pop function the remove function does not return any value.
  - The execution of remove function is shown by following illustration -

```
>>> a=['a','b','c','d','e']
>>> a.remove('c')
>>> a
['a', 'b', 'd', 'e']
>>>
```

- **The del function :** In python, it is possible to remove more than one element at a time using **del function.**
  - For example

```
>>> a=['a','b','c','d','e']
>>> del a[2:4]
>>> a
['a', 'b', 'e']
>>>
```

3.1.1.4 Updating List

- Lists are **mutable**. That means it is possible to change the values of list.
- If the bracket operator is present on the left hand side, then that element is identified and the list element is modified accordingly.
- For example

```
>>> a=['AAA','BBB','CCC']
>>> a[1]='XXX'
>>> a
['AAA', 'XXX', 'CCC']
>>>
```

- Using the in operator we can check whether particular element belongs to the list or not. If the given element is present in the list it returns true otherwise false.
- For example

```
>>> a = ['AAA', 'XXX', 'CCC']
>>> 'XXX' in a
True
>>> 'BBB' in a
False
>>>
```

3.1.2 Basic List Operations

(1) Traversing a List

The loop is used in list for traversing purpose. The **for** loop is used to traverse the list elements.

Syntax

```
for VARIABLE in LIST :
BODY
```

Example

```
>>> a=['a','b','c','d','e'] # List a is created
>>> for i in a:
        print(i)
will result into
a
b
c
d
e
>>>
```

There are some useful functions using which the we can traverse the list. These are discussed below -

(i) Using the range() function

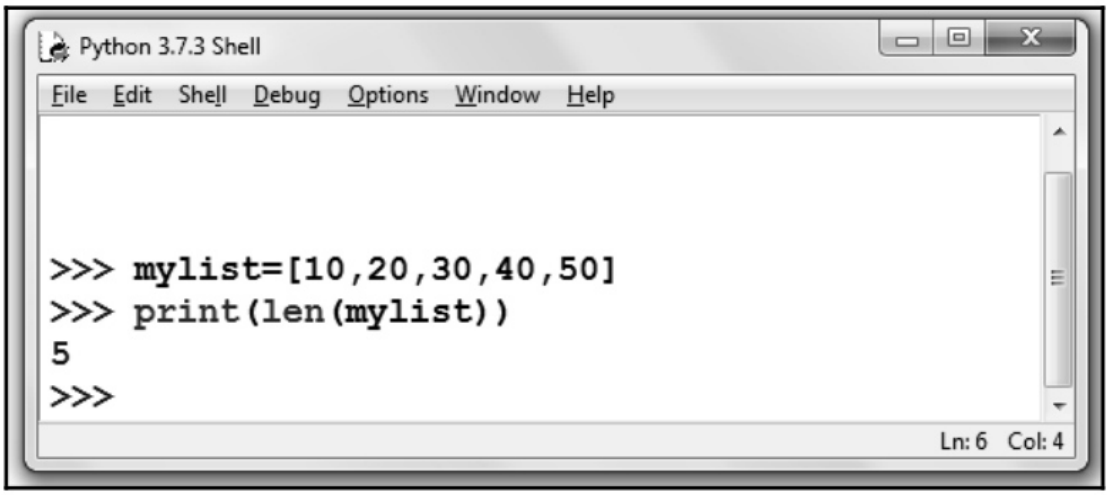
Using **range()** function we can access each element of the list using index of a list.

If we want to increment each element of the list by one, then we must pass index as argument to for loop. This can be done using **range()** function as follows -

```
>>> a=[10,20,30,40]
>>> for i in range(len(a)):
```

(4) Finding Length of a list

The **len()** function is used to find the number of elements present in the list. For example



(5) Membership

- Using the in operator we can check whether particular element belongs to the list or not. If the given element is present in the list it returns true otherwise false.
- For example

```
>>> a = ['AAA', 'XXX', 'CCC']
>>> 'XXX' in a
True
>>> 'BBB' in a
False
>>>
```

(6) List Slices

- The : Operator used within the square bracket that it is a list slice and not the index of the list.

```
>>> a=[10,20,30,40,50,60]
>>> a[1:4]
[20, 30, 40]
>>> a[:5]
[10, 20, 30, 40, 50]
>>> a[4:]
[50, 60]
>>> a[:]
[10, 20, 30, 40, 50, 60]
>>>
```

- If we omit the first index then the list is considered from the beginning. And if we omit the last second index then slice goes to end.
- If we omit both the first and second index then the list will be displayed from the beginning to end.
- Lists are mutable. That means we can change the elements of the list. Hence it is always better to make a copy of the list before performing any operation.

For example

```
>>> a=[10,20,30,40,50]
>>> a[2:4]=[111,222,333]
>>> a
[10, 20, 111, 222, 333, 50]
>>>
```

(6) List Methods

- There are various methods that can work on the list. Let us understand these method with the help of illustrative examples

(i) append

The append method adds the element at the end of the list. For example

```
>>> a=[10,20,30]
>>> a.append(40)   #adding element 40 at the end
>>> a
[10, 20, 30, 40]
>>> b=['A','B','C']
>>> b.append('D')   #adding element D at the end
>>> b
['A', 'B', 'C', 'D']
>>>
```

(ii) extend

The extend function takes the list as an argument and appends this list at the end of old list. For example

```
>>> a=[10,20,30]
>>> b=['a','b','c']
>>> a.extend(b)
>>> a
[10, 20, 30, 'a', 'b', 'c']
>>>
```

(iii) sort

The sort method arranges the elements in increasing order. For example

```
>>> a=['x','z','u','v','y','w']
>>> a.sort()
>>> a
['u', 'v', 'w', 'x', 'y', 'z']
>>>
```

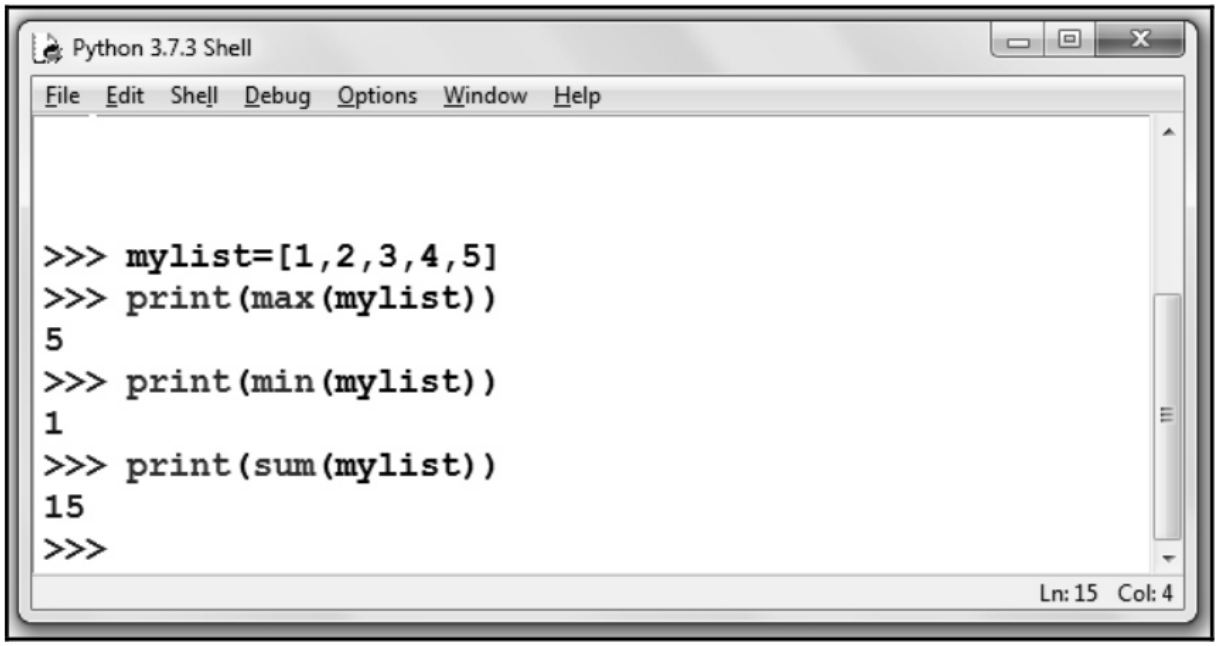
The methods append, extend and sort does not return any value. These methods simply modify the list. These are void methods.

3.1.3 Built in List Functions

- There are various built in functions in python for supporting the list operations. Following table shows these functions

Function	Purpose
all()	If all the elements of the list are true or if the list is empty then this function returns true.
any()	If the list contains any element true or if the list is empty then this function returns true.
len()	This function returns the length of the string.
max()	This function returns maximum element present in the list.
min()	This function returns minimum element present in the list.
sum()	This function returns the sum of all the elements in the list.
sorted()	This function returns a list which is sorted one.

Following screenshot illustrates the use of some built-in functions of list



Functions related to Strings

(1) List function

- String is a sequence of characters and list is sequence of values.
- But list of characters is not the string.
- We can convert the string to list of characters.

For example -

```
>>> str='hello'
>>> myList=list(str)
>>> print(myList)
['h', 'e', 'l', 'l', 'o']
>>>
```

- Note that we have used **list** function to split the characters of the string which ultimately forms the list. The **list** is a built in function.

(2) Split function

- If the string contains multiple words then we need to use the built in function split to split the words into the list.

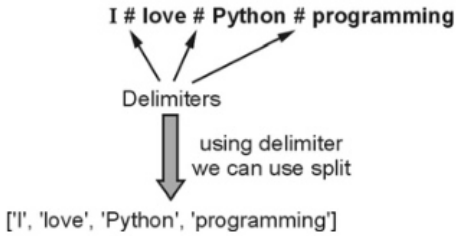
For example -

```
>>> msg="I love Python Programming very much"
>>> myList=msg.split()
>>> print(myList)
['I', 'love', 'Python', 'Programming', 'very', 'much']
>>>
```

- We can pass argument to the **split** function as some delimiter. For instance : If we have following string.

For example -

```
>>> msg="I#love#Python#programming"
>>> myList=msg.split('#')
>>> print(myList)
['I', 'love', 'Python', 'programming']
>>>
```



(3) Join function

- The **join** function is exactly reverse to the **split** function. That means the **join** function takes the list of strings and concatenate to form a string.
- The **join** is basically a string method.
- The use of **join** method is as shown below -

```
>>> msg=['I','love','Python','programming']
>>> ch='#'
>>> ch.join(msg)
'I#love#Python#programming'
>>>
```

Note that the string is joined used the delimiting character #

List Comprehensions

- List comprehension is an elegant way to create and define new lists using existing lists.
- This is mainly useful to make new list where each element is obtained **by applying some operations to each member** of another sequence.

Syntax

```
List=[expression for item in the list]
```

Example 3.1.2 : Write a python program to create a list of even numbers from 0 to 10.

Solution :

```
even = [] #creating empty list
for i in range(11):
   if i % 2 ==0:
      even.append(i)
print("Even Numbers List: ",even)
      even = [] #creating empty list
for i in range(11):
   if i % 2 ==0:
      even.append(i)
print("Even Numbers List: ",even)
```

Output

```
Even Numbers List:  [0, 2, 4, 6, 8, 10]
```

Program explanation : In above program,

(1) We have created an empty list first.

(2) Then using the range of numbers from 0 to 11 we append the empty list with even numbers. The even number is test using the if condition. i.e. if I %2= =0. If so then that even number is appended in the list.

(3) Finally the comprehended list will be displayed using print statement.

Example 3.1.3 : Write a python program to combine and print two lists using list comprehension.

Solution :

```
print([(x,y)for x in['a','b'] for y in ['b','d'] if x!=y])
```

Output

```
[('a', 'b'), ('a', 'd'), ('b', 'd')]
```

Example 3.1.4 : To accept N numbers from user. Compute and display maximum in list, minimum in list, sum and average of numbers.

Solution :

```
mylist = [] # start an empty list
print("Enter the value of N: ")
N = int(input()) # read number of element in the list
for i in range(N):
   print("Enter the element: ")
   new_element = int(input()) # read next element
   mylist.append(new_element) # add it to the list

print("The list is: ")
print(mylist)
print("The maximum element from list is: ")
max_element=mylist[0]
for i in range(N):  #iterating throu list
   if(mylist[i]>max_element):
      max_element=mylist[i] #finding the maximum element

print(max_element)#printing the maximum element

print("The minimum element from list is: ")
min_element=mylist[0]
for i in range(N):  #iterating throu list
   if(mylist[i]<min_element):
      min_element=mylist[i] #finding the minimum element

print(min_element)#printing the minimum element
print("The sum of all the numbers in the list is: ")
sum=0
for i in range(N):  #iterating throu list
   sum=sum+mylist[i] # finding the sum
print(sum)
avg=sum/N # computing the average
print("The Average of all the numbers in the list is: ",avg)
```

Output

```
Enter the value of N:
5
Enter the element:
33
Enter the element:
22
Enter the element:
55
Enter the element:
11
Enter the element:
44
The list is:
[33, 22, 55, 11, 44]
The maximum element from list is:
55
The minimum element from list is:
11
The sum of all the numbers in the list is:
165
The Average of all the numbers in the list is:  33.0
>>>
```

Review Questions

1. What is list ? Explain how to access the list elements.

2. Write a Python program to delete and update list elements.

3. List and explain any five built in functions in list.

3.2 Tuples

3.2.1 Introduction to Tuples

3.2.1.1 Definition Tuple

Tuple is a collection of elements which are enclosed within the parenthesis, and these elements are separated by commas.

How to Create Tuple ?

```
T1  =  (10, 20, 30, 40)
T2  =  ('a','b','c','d')
T3  =  ('A',10, 20)
T4  =  ('aaa','bbb',30, 40)
```

The empty tuple can be created as

```
T1  =  ()
```

How to write tuple ?

Tuple is written within the parenthesis. Even if the tuple contains a single value, the comma is used as a separator. For example -

```
T1  =  (10,)
```

Difference between Tuple and List

Tuple	List
Tuple use parenthesis	List use square brackets
Tuples can not be changed	Lists can be changed.

3.2.1.2 Accessing Values in Tuple

The element in Tuple can be accessed using the index. For example

```
>>> t1=(10,20,'AAA','BBB')
>>> print(t1[0])
        10
>>> print(t1[1:3])
(20, 'AAA')
>>>
```

3.2.1.3 Deleting Values in Tuple

- We can delete a tuple using **del** statement. The code for this is as given below -

```
T1=(10,20,30)
del T1
print(T1)
```

- As an output of the above program, the error message will be displayed as T1 gets deleted.

3.2.1.4 Updating Tuple

Tuples are immutable. That means - values in the tuple can not be changed. We can only extract the values of one tuple to create another tuple.

For example

```
T1 = (10,20,30)
T2 = (40,50)
T3 = T1+T2
print(T3)
```

Output

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

3.2.2 Basic Tuple Operations

(1) len : The len function is used to find the total number of elements in the tuple.

```
>>> t1=(10,20,30,40)
>>> print(len(t1))
4
>>>
```

(2) Concatenation : The operator + is used to concatenate two tuples. For example

```
>>> t1=(10,20,30)
>>> t2=(40,50,60)
>>> print(t1+t2)
(10, 20, 30, 40, 50, 60)
>>>
```

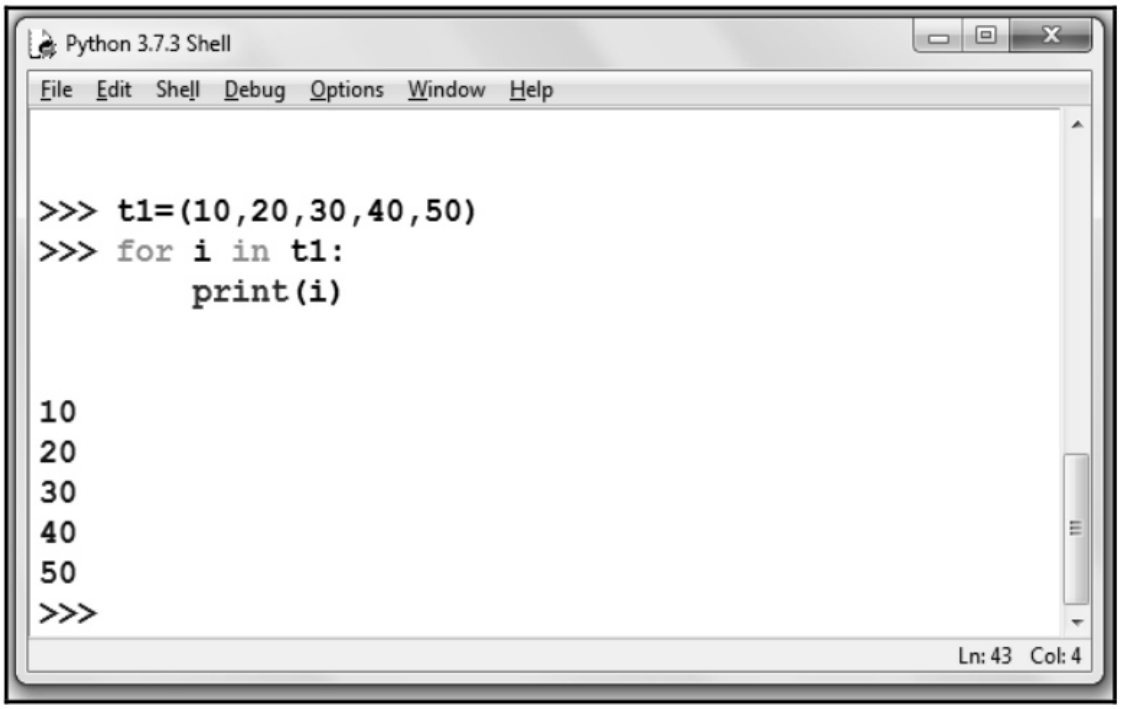
(3) Repetition : For repetition of the elements in the Tuple, we use * operator. For example

```
>>> t1=(10,20,30)
>>> print(t1*3)
(10, 20, 30, 10, 20, 30, 10, 20, 30)
>>>
```

(4) Membership : Membership means checking if the element is present in the tuple or not. The membership operation is performed using **in** operator.

```
>>> t1=(10,20,30,40,50)
>>> print(3 in t1)
False
>>> print(30 in t1)
True
>>>
```

(5) Iteration: Iteration means accessing individual element of tuple. With the help of **for loop** we can access the element of a tuple. Following screenshot illustrates this operation



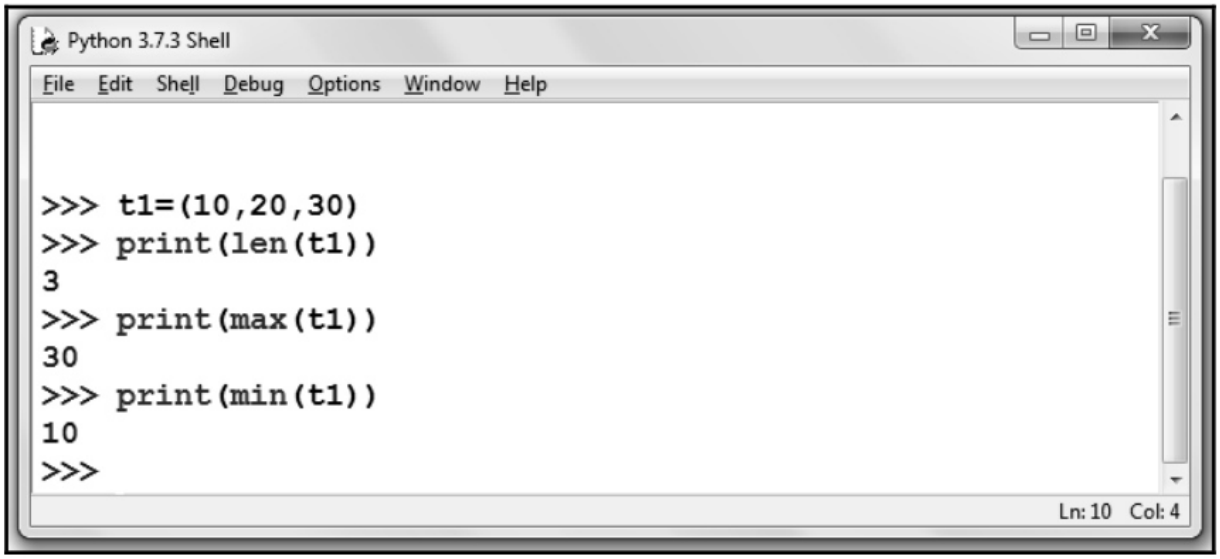
3.2.3 Built in Tuple Function

There are some useful built in Tuple functions. These functions are enlisted in the following table -

Function	Purpose
cmp(t1,t2)	This function compares tuple t1 with tuple t2. It will return either 1, 0 or -1, depending upon whether the two tuples being compared are similar or not. Here, cmp() will return negative (-1) if t1<t2, zero (0) if t1=t2, positive (1) if t1>t2.
len(t1)	This function gives the total length of the tuple.
max(t1)	This function returns maximum value from given tuple t1.
min(t1)	This function returns minimum value from given tuple t1.
tuple(sequence)	This function converts the list into tuple.

The illustration of above functions is as shown below –

```
>>>t1=(10,20)
>>>t2=(10,20,30)
>>>print(cmp(t1,t2)
>>>-1
```



Tuple function: There is built in function tuple which is used to create the tuple.

For example

```
>>> t1=tuple()
>>> print(t1)
()
>>>
```

Another example

```
>>> t1=tuple("hello")
>>> print(t1)
('h', 'e', 'l', 'l', 'o')
>>>
```

Review Questions

1. *Enlist built in functions in tuple.*
2. *What is tuple ? Give the difference between tuple and list.*
3. *What are the basic operations that can be performed on tuple ? Explain with suitable example.*

3.3 Sets

3.3.1 Introduction to Sets

Sets are data structures that, based on specific rules they define. The specific rules that are followed by sets are -

(1) Items in a set are **unique**. We can not have a set that contains two items that are equal.

(2) Items in a set are **not ordered**. Depending on the implementation, items may be sorted using the hash of the value stored, but when you use sets you need to assume items are ordered in a random manner.

How to create a Set ?

The set can be created using **set** function. For example -

```
>>>color=set(['red','green','blue'])
```

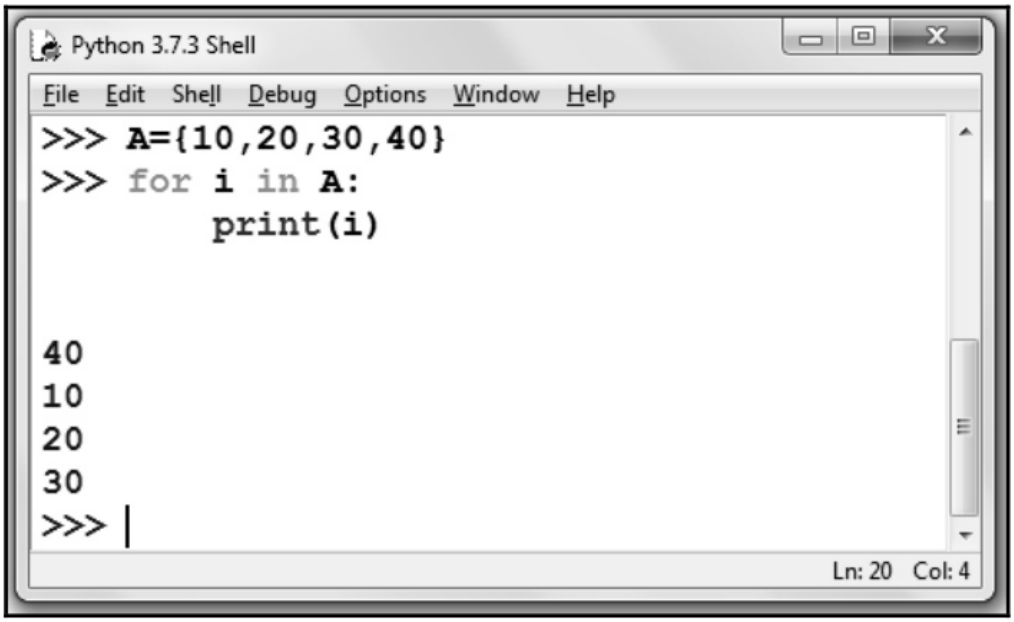
3.3.1.1 Accessing Values in Set

We cannot access the values in set using index as the set is unordered and **has no index.**

We can display the contents of set using following Python Code

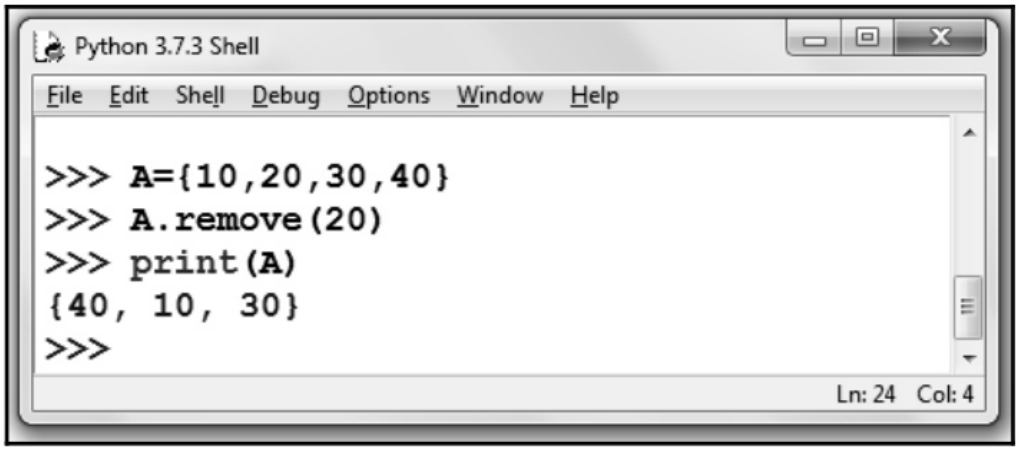
```
>>> A={10,20,30,40}
>>> print(A)
{40, 10, 20, 30}
```

It is also possible to iterate through each item of a set using for loop. The code is as follows

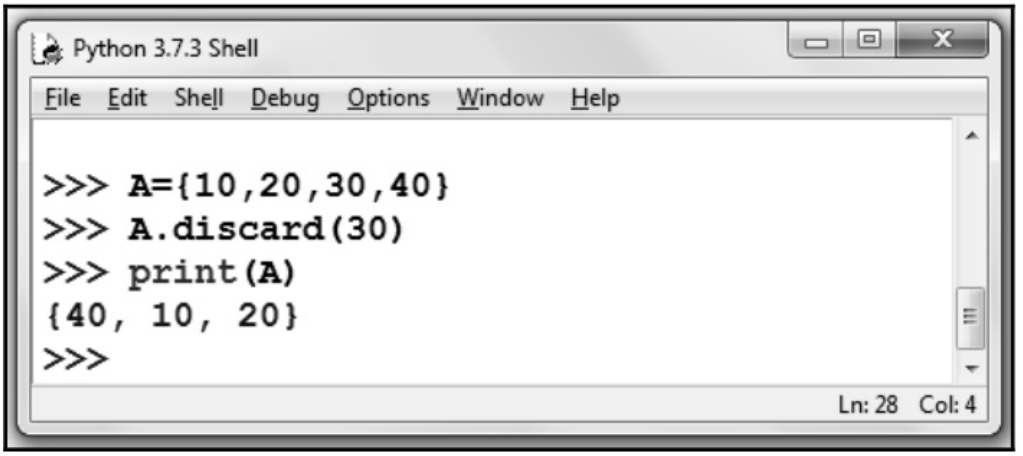


3.3.1.2 Deleting Values in Set

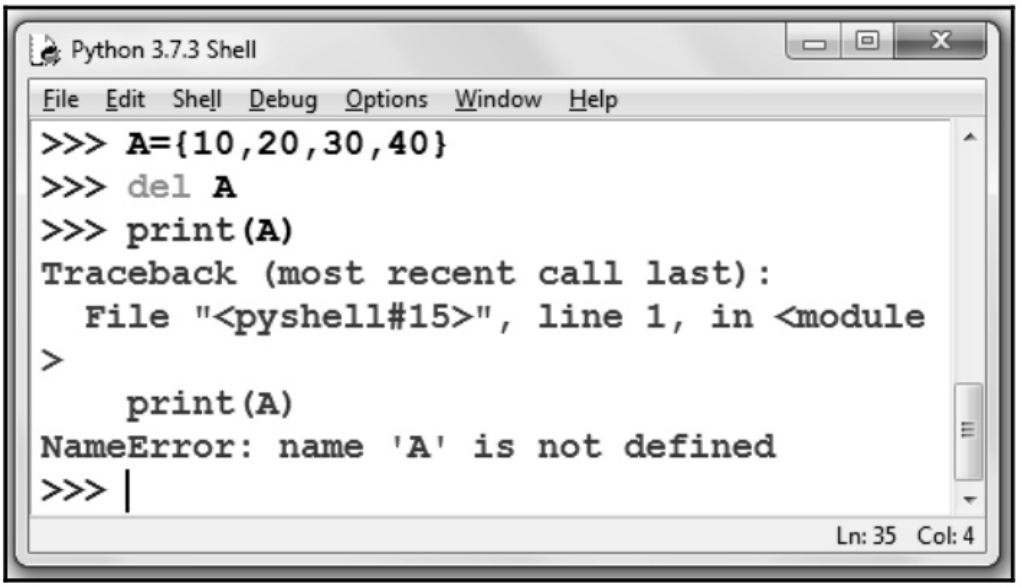
For removing the item from the set either remove or discard method is used. The remove() method is illustrated below -



Similarly we can use discard method to remove an element from the set.



The **del** keyword is used to delete the set completely. The illustration of this function is as follows -



3.3.1.3 Updating Values in Set

- Once the set is created, we cannot change the values in the set. But we can add the element to the set.
- Using the **add()** method we can add the element to the set.

```
>>> A={10,20,30,40}
>>> A.add(25)
>>> print(A)
{40, 10, 20, 25, 30}
>>>
```

- Using the **update()** method more than one elements can be added to the set.

```
>>> A={10,20,30,40}
>>> A.update([25,35,45])
>>> print(A)
{35, 40, 10, 45, 20, 25, 30}
>>>
```

3.3.2 Basic Set Operations

Various set type operators in Python are -

(1) Union : The union between two sets, results in a third set with all the elements from both sets. The union operation is performed using the operator |.

```
>>> a=set([1,2,3])
>>> b=set([2,3,4])
>>> c=a|b
>>> print(c)
{1, 2, 3, 4}
>>>
```

There exists a method named **union** for union of two sets. For example -

```
>>> x=set(['a','b','c'])
>>> y=set(['b','c','d'])
>>> c=x.union(y)
>>> print(c)
{'a', 'd', 'b', 'c'}
>>>
```

(2) Intersection : The intersection of two sets is a third set in which only common elements from both the sets are enlisted. Intersection is performed using & operator.

For example -

```
>>> a=set([10,20,30])
>>> b=set([20,30,40])
>>> c=a&b
>>> print(c)
{20, 30}
```

There exists a method named **intersection** for intersection of two sets for example

```
>>> a=set([10,20,30])
>>> b=set([20,30,40])
>>> c=a.intersection(b)
>>> print(c)
{20, 30}
```

(3) Difference : Difference of A and B i.e. (A - B) is a set of elements that are only in A but not in B. Similarly, B - A is a set of element in B but not in A. The operator - is used for difference. Similarly the method named **difference** is also used for specifying the difference. For example -

```
>>> a=set([10,20,30,40,50])
>>> b=set([40,50,60,70,80])
>>> c=a-b
>>> print(c)
{10, 20, 30}
>>>a.difference(b)
{10, 20, 30}
>>>
```

(4) Symmetric difference : Symmetric Difference of A and B is a set of elements in both A and B except those that are common in both.

Symmetric difference is performed using ^ operator. Same can be accomplished using the method **symmetric_difference().**For example -

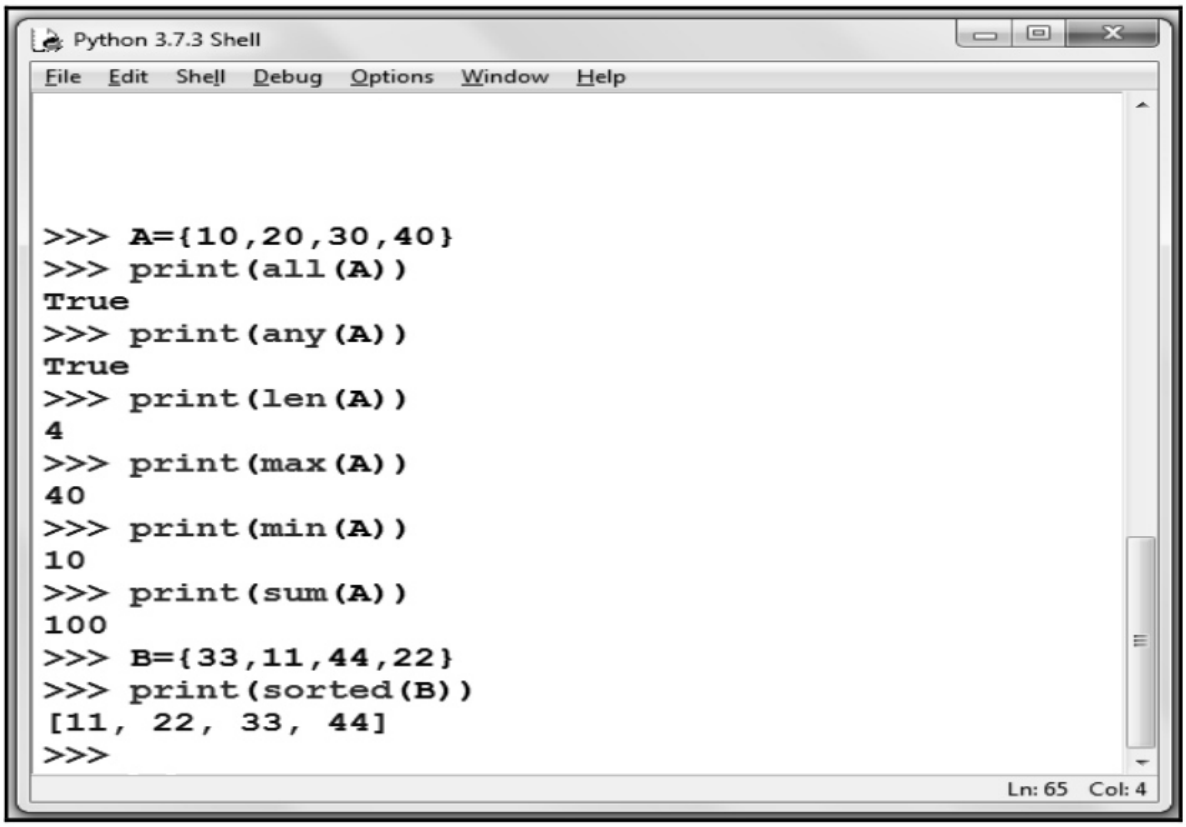
```
>>> a=set([10,20,30,40,50])
>>> b=set([40,50,60,70,80])
>>> c=a^b
>>> print(c)
{80, 20, 70, 10, 60, 30}← Note that 40 and 50 are the common elements which are not present in set c.
>>>a.symmetric_difference(b)
{80, 20, 70, 10, 60, 30}
>>>
```

3.3.3 Built in Set Function

Various built in set functions are enlisted in the following table.

Function	Purpose
all()	This function return True if all elements of the set are true. This function also returns a true value is the set is empty.
any()	This function return True if any element of the set is true. If the set is empty, return False.
enumerate()	This function return an enumerate object. It contains the index and value of all the items of set as a pair.
len()	This function return the length of the set. That means it returns number of elements present in the set.
max()	This function returns the maximum value present in the set.
min()	This function returns the minimum value present in the set.
sorted()	This function returns a new sorted list from the elements.
sum()	This function returns the sum of all elements in the set.

The illustration of above functions in represented by following screenshot -



Review Questions

1. *What is set ? How to create a set in python ?*
2. *What are the basic operations that can be performed on set in python ? Explain with suitable examples.*

3.4 Dictionaries

3.4.1 Introduction to Dictionary

Definition : In python, dictionary is unordered collection of items. These items are in the form key-value pairs.

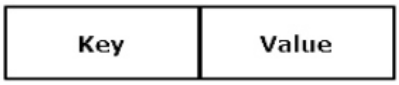


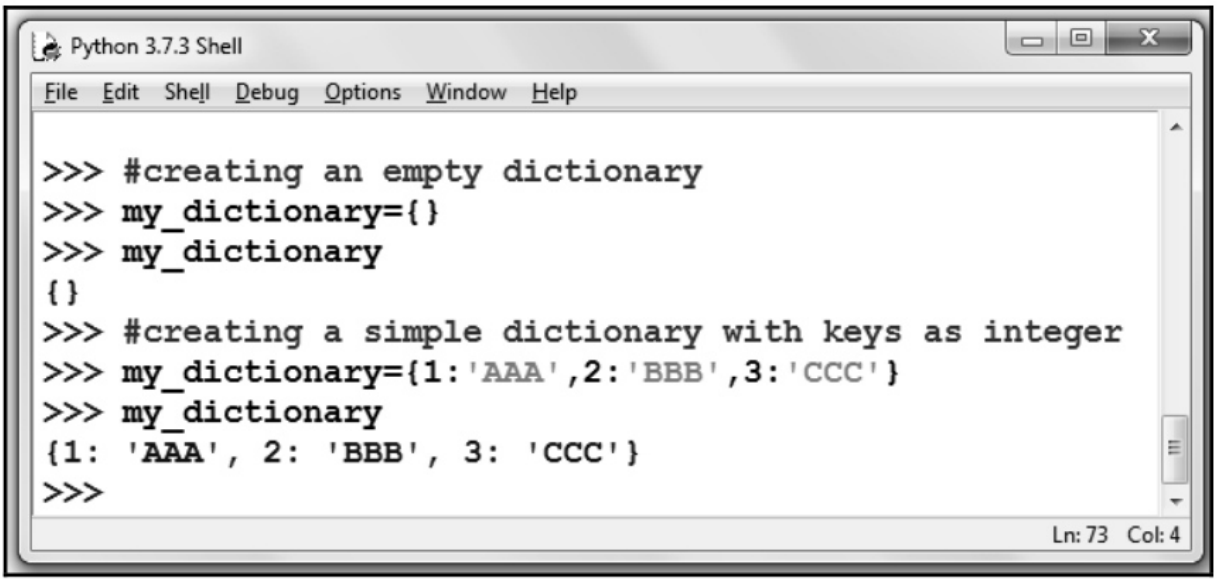
Fig. 3.4.1 Dictionary element

- The dictionary contains the collection of indices called **keys** and collection of **values**.
- Each key is associated with a single value.
- The association of keys with values is called **key-value pair or item**.
- Dictionary always represent the mappings of keys with values. Thus each key maps to a value.

How to create dictionary ?

- Items of the dictionary are written within the {} brackets and are separated by commas.
- The key value pair is represented using : operator. That is **key:value.**
- **For example**

  my_dictionary={1:'AAA',2:'BBB',3:'CCC'}
- The keys are unique and are of immutable types - such as string, number, tuple.
- Here is a screenshot that shows how to create a dictionary



- We can also create a dictionary with mixed data type as

  >>> my_dictionary= {'name':'AAA',2:89}
- We can also create a dictionary using the word **dict()**.

For example -

```
>>> my_dictionary=dict({0:'Red',1:'Green',2:'Blue'})
```

3.4.1.1 Accessing values in Dictionary

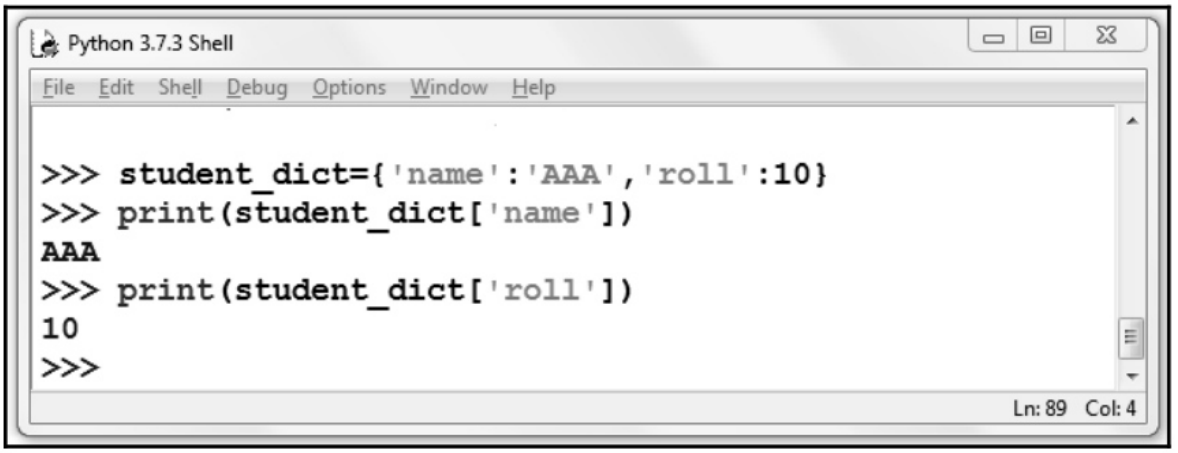
We can access the element in the dictionary using the keys. Following script illustrates it

DictionaryDemo.py

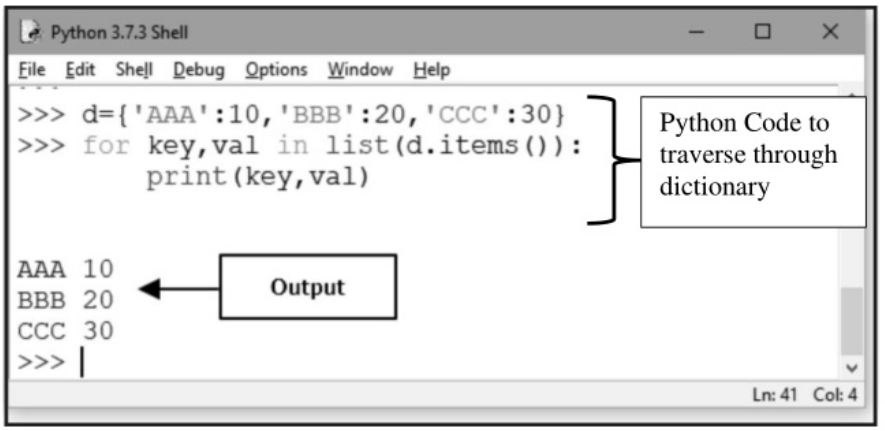
```
student_dict={'name':'AAA','roll':10}
print(student_dict['name'])
print(student_dict['roll'])
```

Output

```
AAA
10
```



- Now to traverse the dictionary items we need to consider two values at a time and these are the values for both **key** and **value**.
- At a time we can assign both of these values. This is called multiple assignment. **For** example **-** Following is a python code in which we are traversing the keys and values of a dictionary in a single loop.



Code explanation :

- Note that, the above loop has two iteration variables because items returns a list of tuples.

The key-value is a tuple assignment that successively iterates through each of key value pairs in the dictionary.

3.4.1.2 Deleting Values in Dictionary

For removing an item from the dictionary we use the keyword del.

For example

```
>>> del my_dictionary[2] #deleting the item from dictionary
>>> print(my_dictionary) #display of dictionary
{0: 'Red', 1: 'Green', 3: 'Yellow'}
>>>
```

3.4.1.3 Updating Values in Dictionary

We can update the value of the dictionary by directly assigning the value to corresponding key position.

For example -

```
>>> my_dictionary=dict({0:'Red',1:'Green',2:'Blue'}) #creation of dictionary
>>> print(my_dictionary) #display of dictionary
{0: 'Red', 1: 'Green', 2: 'Blue'}
>>> my_dictionary[1]='Yellow' #updating the value at particular index
>>> print(my_dictionary) #display of dictionary
{0: 'Red', 1: 'Yellow', 2: 'Blue'} #updation of value can be verified.
>>>
```

3.4.2 Basic Dictionary Operations

Various operations that can be performed on dictionary are :

1. Adding item to dictionary

We can add the item to the dictionary.

For example -

```
>>> my_dictionary=dict({0:'Red',1:'Green',2:'Blue'})
>>> print(my_dictionary)
{0: 'Red', 1: 'Green', 2: 'Blue'}
>>> my_dictionary[3]='Yellow' #adding the element to the dictioary
>>> print(my_dictionary)
{0: 'Red', 1: 'Green', 2: 'Blue', 3: 'Yellow'}
>>>
```

2. Checking length

The len() function gives the number of pairs in the dictionary.

For example

```
>>> my_dictionary=dict({0:'Red',1:'Green',2:'Blue'})#creation of dictionary
>>> len(my_dictionary)  # finding the length of dictionary
3                       #meaning – that there are 3 items in dictionary
>>>
```

3. Iterating through Dictionary

For iterating through the dictionary we can use the for loop.

The corresponding Key and Values present in the dictionary can be displayed using this for loop.

The illustrative program is as follows -

LoopDemo.py

```
d={'Red':10, 'Blue':42,'Orange':21}
for i in d:
   print(i,d[i])
```

Output

```
Red 10
Blue 42
Orange 21
```

3.4.3 Built in Dictionary Functions

Following are some commonly used methods in dictionary.

Method	Purpose
clear	Removes all items from dictionary.
copy( )	Returns a copy of dictionary.
fromkeys( )	Creates a new dictionary from given sequence of elements and values provided by user.
get(key[,d])	Return the value of key. If key doesnot exit, return d (defaults to None).
items( )	Return a new view of the dictionary's items (key, value).
keys( )	Return a new view of the dictionary's keys.
pop(key[,d])	Remove the item with key and return its value or d if key is not found. If d is not provided and key is not found, raises KeyError.
popitem( )	Remove and return an arbitary item (key, value). Raises KeyError if the dictionary is empty.
setdefault(key[,d])	If key is in the dictionary, return its value. If not, insert key with a value of d and return d (defaults to none).
update([other])	Update the dictionary with the key/value pairs from other, overwriting existing keys.
values( )	Return a new view of the dictionary's values.

1. The clear method

This method removed all the items from the dictionary. This method does not take any parameter and does not return anything.

For example

```
>>> my_dictionary={1:'AAA',2:'BBB',3:'CCC'} # creation of dictionary
>>> print(my_dictionary) #display
{1: 'AAA', 2: 'BBB', 3: 'CCC'}
>>> my_dictionary.clear()  #using clear method
>>> print(my_dictionary) #display
{}
>>>
```

2. The copy method

The copy method returns the copy of the dictionary. It does not take any parameter and returns a shallow copy of dictionary.

For example

```
>>> my_dictionary={1:'AAA',2:'BBB',3:'CCC'}
>>> print(my_dictionary)
{1: 'AAA', 2: 'BBB', 3: 'CCC'}
>>> new_dictionary=my_dictionary.copy()
>>> print(new_dictionary)
{1: 'AAA', 2: 'BBB', 3: 'CCC'}
>>>
```

3. The fromkey method

The **fromkeys( )** method creates a new dictionary from the given sequence of elements with a value provided by the user.

Syntax

```
dictionary.fromkeys(sequence[, value])
```

The **fromkeys( )** method returns a new dictionary with the given sequence of elements as the keys of the dictionary. If the value argument is set, each element of the newly created dictionary is set to the provided value.

For example

```
>>> keys={10,20,30}
>>> values = 'Number'
>>> new_dict=dict.fromkeys(keys,values)
>>> print(new_dict)
{10: 'Number', 20: 'Number', 30: 'Number'}
>>>
```

4. The get method

The **get( )** method returns the value for the specified key if key is in dictionary. This method takes two parameters key and value. The get method can return either key or value or nothing.

Syntax

```
dictionary.get(key[, value])
```

For example

```
>>> student={'name':'AAA','roll':10,'marks':98} #creation of dictionary
>>> print("Name: ",student.get('name'))
Name:  AAA
>>> print("roll: ",student.get('roll'))
roll:  10
>>> print("marks: ",student.get('marks'))
marks:  98
>>> print("Address: ",student.get('address')) #this key is 'address' is not specified in the list
Address:  None #Hence it returns none
>>>
```

5. The value method

This method returns the **value** object that returns view object that displays the list of values present in the dictionary.

For example

```
>>> my_dictionary ={'marks1':99,'marks2':96,'marks3':97}#creating dictionary
>>> print(my_dictionary.values()) #displaying values
dict_values([99, 96, 97])
>>>
```

6. The pop method

This method the pop() method removes and returns an element from a dictionary having the given key.

Syntax

```
pop(key[, default])
```

where **key** is the key which is searched for removing the value. And **default** is the value which is to be returned when the key is not in the dictionary.

For example

```
my_dictionary={1:'Red',2:'Blue',3:'Green'}#creation of dictionary
>>> val=my_dictionary.pop(1) #removing the element with key 1
>>> print("The popped element is: ",val)
The popped element is:  Red
>>> val=my_dictionary.pop(5,3)
#specifying default value. When the specified key is not present in the list, then the
#default value is returned.
>>> print("The popped element using default value is: ",val)
The popped element using default value is:  3 #default value is returned
>>>
```

Example 3.4.1 : Write a python program to sort the elements of dictionary.

Solution :

Sort.py

```
d={'Red':10, 'Blue':42,'Yellow':32,'Orange':21}
myList=list(d.keys())
print(myList)
myList.sort()
print("Sorted list based on Keys")
for i in myList:
   print(i)
```

Output

```
Sorted list based in Keys
Blue
Orange
Red
Yellow
```

Example 3.4.2 : Write a python program to create a tuple from given dictionary elements.

Solution : Using the method named item of dictionary, the list of tuples can be returned.

For example -

```
>>> d={'AAA':10,'BBB':20,'CCC':30}
>>> t1=list(d.items())
>>> print(t1)
[('AAA', 10), ('BBB', 20), ('CCC', 30)]
>>>
```

Review Questions

1. *What is dictionary ? How to create a dictionary ? Also explain how to update the dictionary elements.*
2. *Explain any two methods used in dictionary with illustrative python code.*

3.5 Strings

- String is basically the sequence of characters.
- Any desired character can be accessed using the index. For example

```
>>> country="India"
>>> country[1]   ← Here using index the particular character is accessed
'n'
>>> country[2]
'd'
>>>
```

The index is an integer value if it is a decimal value, then it will raise an error. For example

```
>>> country[1.5]
TypeError: string indices must be integers
>>>
```

The string can be created using double quote or single quotes. For example

```
>>> msg="Hello"
>>> print(msg)
Hello
>>> msg='Goodbye'
>>> print(msg)
Goodbye
```

The string is a collection of characters stored sequentially as follows -

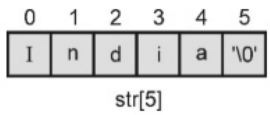


Fig. 3.5.1 String indexes

Review Question

1. *Explain the concept of string used in python.*

3.6 String Operations

3.6.1 Finding Length of String

There is an in-built function to find length of the string. This is **len** function. For example

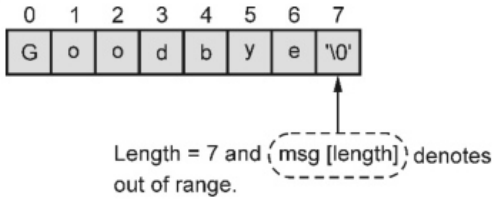
```
>>> msg='Goodbye'
>>> print(msg)
```

```
Goodbye
>>> len(msg)
7
```

Sometimes, we may get tempted to use the length value to refer to the last character. For example

```
>>> msg='Goodbye'
>>> length=len(msg)
>>> last_char=msg[length]
Traceback (most recent call last):
File "<pyshell#2>", line 1, in <module>
last_char=msg[length]
IndexError: string index out of range
>>>
```

The **msg** array will be as follows -



Hence for referring the last character of string, it should be msg[length-1] and not msg [length]

3.6.2 Concatenation

Joining of two or more strings is called concatenation.

In python we use + operator for concatenation of two strings.

For example -

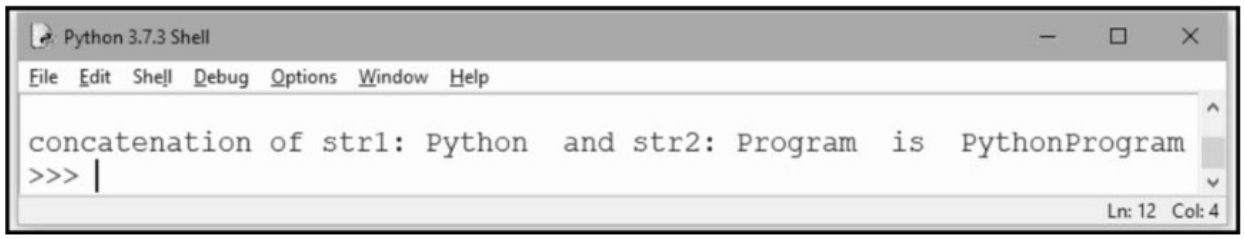
Example 3.6.1 : Write a python program to concatenate two strings.

Solution :

Concatenate.py

```
str1="Python"
str2="Program"
str3=str1+str2
print("concatenation of str1:",str1," and str2:",str2," is ",str3)
```

Output



Program explanation : In above program, we have used + operator to concatenate two strings.

3.6.3 Appending

We can use += operator to append some string to already existing string. Following example illustrates this -

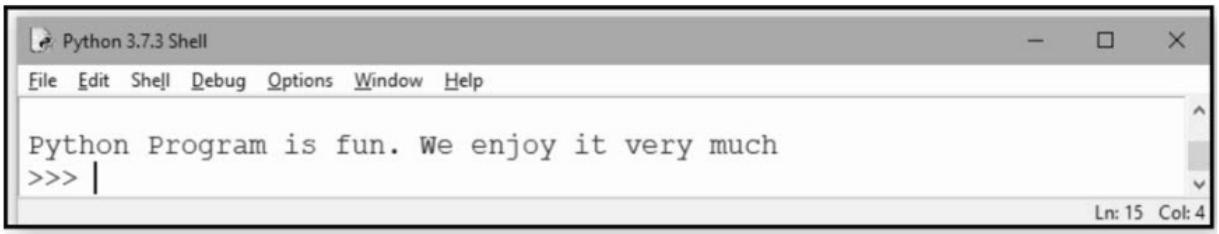
Example 3.6.2 : Write a python program to demonstrate the use of += operator form concatenation of the string.

Solution :

ConcatenateDemo.py

```
str="Python Program is fun"
str+=". We enjoy it very much"
print(str)
```

Output



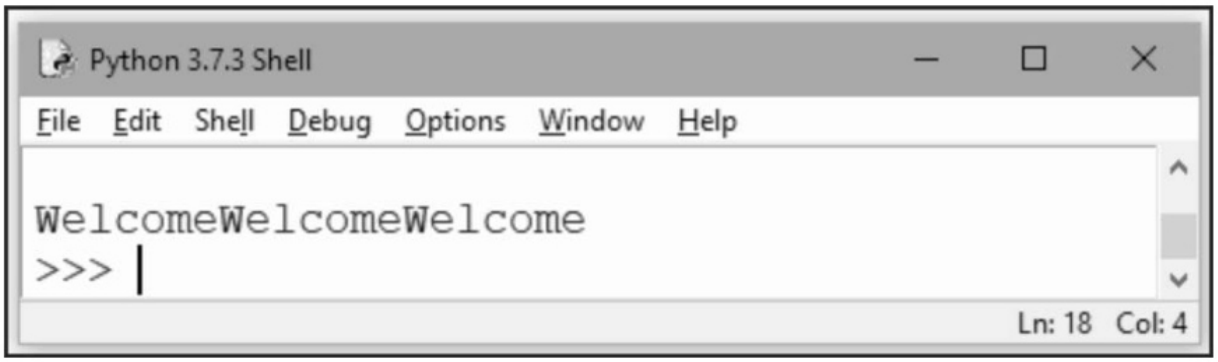
3.6.4 Multiplying Strings

The * operator is used to repeat the string for n number of times. Following program illustrates it.

StarDemo.py

```
str="Welcome"
print(str*3)
```

Output



The print statement always prints the output on the new line. If we do not want the output on separate lines then just add **end** statement with the separator like whitespace, comma etc. Following code illustrates this

Test1.py

```
print("Python", end=' ')
print("Program")
```

Output

```
Python Program
>>>
```

Review Questions

1. *Write a python program to concatenate the strings.*
2. *Write a python program to multiply the strings.*

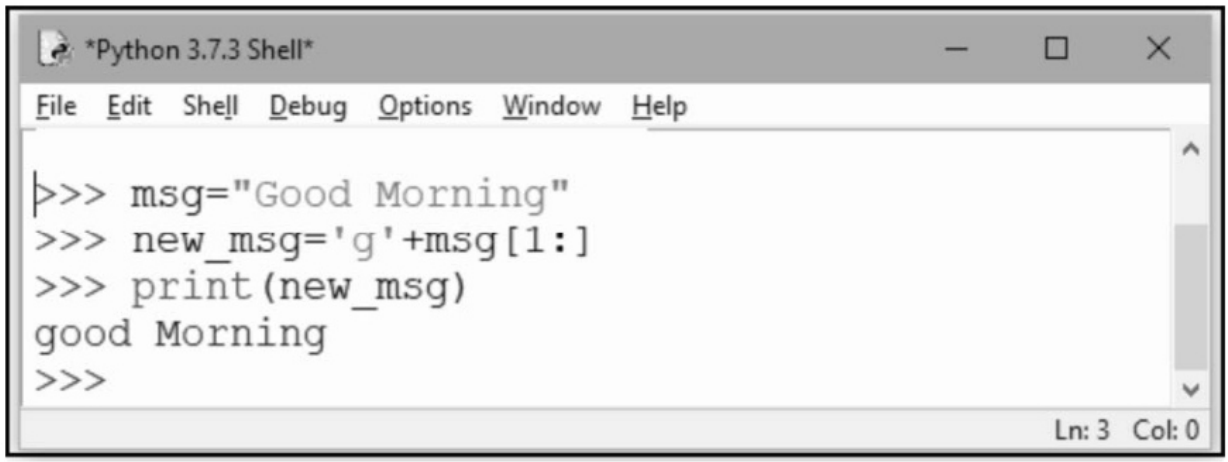
3.7 Strings are Immutable

- Strings are immutable i.e we cannot change the existing strings. For example

```
>>> msg="Good Morning"
>>> msg[0]='g'
```

TypeError : 'str' object does not support item assignment.

- To make the desired changes we need to take new string and manipulate it as per our requirement. Here is an illustration.

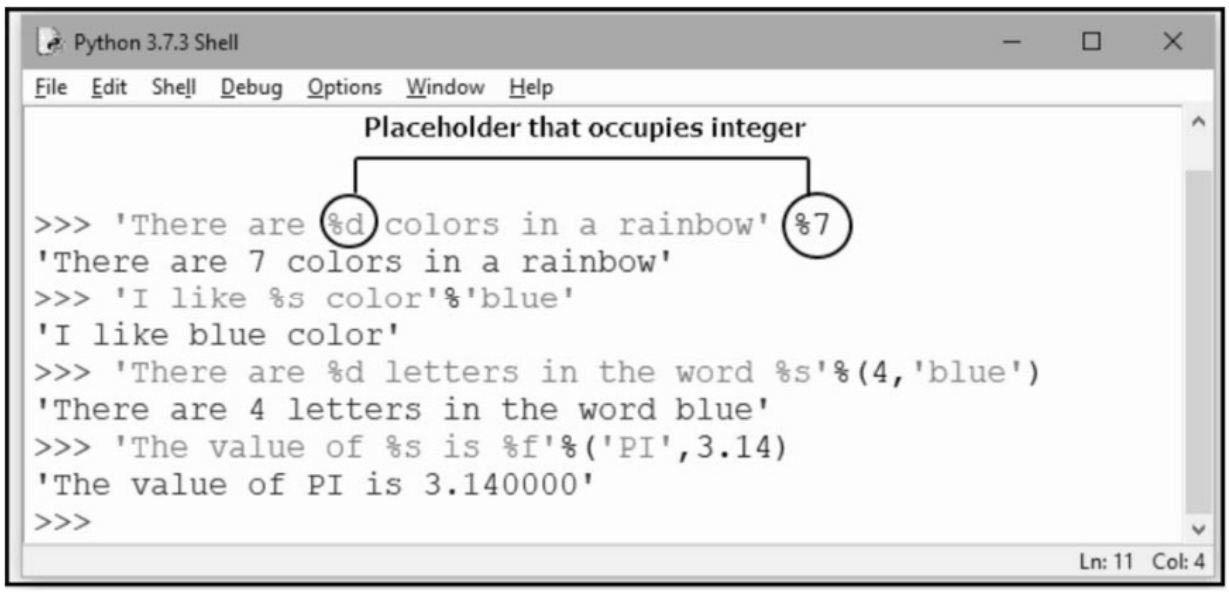


Review Question

1. *Explain the term - Strings are immutable.*

3.8 String Formatting Operator

- The format operator is specified using % operator.
- For example - If we want to display integer value then the format sequence must be %d, similarly if we want to display string value then the format sequence must be %s and so on.
- Here is the illustration.



- Various format specifiers are enlisted in the following table

Conversion	Meaning
d	Signed integer decimal.
i	Signed integer decimal.
u	Obsolete and equivalent to 'd', i.e. signed integer decimal.
x	Unsigned hexadecimal (lowercase).
X	Unsigned hexadecimal (uppercase).
e	Floating point exponential format (lowercase).
E	Floating point exponential format (uppercase).
f	Floating point decimal format.
F	Floating point decimal format.
g	Same as "e" if exponent is greater than - 4 or less than precision, "f" otherwise.
G	Same as "E" if exponent is greater than - 4 or less than precision, "F" otherwise.
c	Single character (accepts integer or single character string).
r	String (converts any python object using repr()).
s	String (converts any python object using str()).
%	No argument is converted, results in a "%" character in the result.

- If there is more than one format sequence in the string, the second argument has to be a tuple. Each format sequence is matched with an element of the tuple, in order.
- For example : Following errors are due to mismatch in tuple with format sequence.

```
>>> 'There are %d%d%d numbers'%(1,2)      #mismatch in number of elements
TypeError : Not enough arguments for format string
>>> 'There are %d rows'%'three'
TypeError : %d format: a number is required, not str     #mismatch in type of element
```

Example 3.8.1 : Write a python program to display the name of the student, his course and age.

Solution :

```
name='Parth'
course='Computer Engineering'
age=18
print("Name = %s and course= %s and age = %d"%(name,course,age))
print("Name = %s and course= %s and age = %d"%('Anand','Mechanical Engineering',21))
```

Output

```
Name = Parth and course= Computer Engineering and age = 18
Name = Anand and course= Mechanical Engineering and age = 21
>>>
```

Review Question

1. *What are different format operators ?*

3.9 Built in Functions and Methods in Strings

Python has a set of built in methods and functions. Some of the commonly used methods and functions are listed in the following table

Method	Purpose	Example
capitalize()	This method converts the first letter of the string to capital.	str = "i like python programming very much." msg = str.capitalize() print (msg) **Output** I like python programming very much. >>>
count()	It returns the number of times particular string appears in the statement.	str = 'twinkle, twinkle little start, How I wonder what you are' msg = str.count("twinkle") print (msg) **Output** 2
center(width, fillchar)	This method returns centered in a string of length width. Padding is done using the specified fillchar.	str = 'India' print(str.center(20,'#')) **Output** #######India######## >>>
endswith(value, start,end)	This method returns true if the string ends with the specified value, otherwise false.	str = 'India is the best country in the world!' print(str.endswith("!")) **Output** True
find()	It returns the index of first occurrence of substring.	str = 'I scream, you scream, we all scream for ice cream' print(str.find("scream")) **Output** 2
index()	It searches the string for specified value and returns the position of where it was found.	str = 'Sometimes later becomes never. Do it now.' print(str.index("later")) **Output** 10
isalpha()	It returns true if all the characters in the string are alphabets.	str = 'India' print(str.isalpha()) **Output** True
isdecima()	It returns true if all characters in the string are decimal(0-9).	s="1234" print(s.isdecimal()) t="ab12345" print(t.isdecimal()) **Output** True False

isdigit()	It returns true if all the characters in the string are digits.	s="1234" print(s.isdecimal()) t="Hello 1234" print(t.isdecimal()) **Output** True False
islower()	It returns true if all the characters of the string are in lowercase.	s="india is a great" print(s.islower()) t="I Love my Country" print(t.islower()) **Output** True False
isupper()	It returns true if all the characters of the string are in uppercase.	s="INDIA IS GREAT" print(s.isupper()) t="I Love my Country" print(t.isupper()) **Output** True False
isspace()	It returns true if all the characters of the string are in uppercase.	s="\t " print(s.isspace()) t="I Love my Country" print(t.isupper()) **Output** True False
len()	It returns length of the string or number of characters in the string.	str="I Love my Country" print(len(str)) **Output** 17
replace()	It replaces one specific phrase by some another specified phrase.	str="I Love my Country" print(str.replace("Country","India")) **Output** I Love my India
lower()	Converts all characters into the string to lowercase.	str="I LOVE MY COUNTRY" print(str.lower()) **Output** i love my country
upper()	Converts all characters into the string to uppercase.	str="i love my country" print(str.upper()) **Output** I LOVE MY COUNTRY

lstrip()	Removes all leading whitespaces of the string.	str=" India " print(str.lstrip()+"is a country") **Output** India is a country
rstrip()	Removes all trailing whitespaces of the string.	str=" India " print(str.rstrip()+"is a country") **Output** India is a country
strip()	Removes all leading and trailing whitespaces of the string.	str=" India " print(str.strip()+"is a country") **Output** India is a country
max()	Returns highest alphabetical character.	str='zebracrossing' print(max(str)) **Output** z
min()	Returns lowest alphabetical character.	str='zebra crossing' print(min(str)) **Output** a
title()	Returns first letter of the string to uppercase.	str='python is easy to learn' print(str.title()) **Output** Python Is Easy To Learn
split()	This function splits the string at specified separator and returns a list.	str='python#is#easy#to#learn' print(str.split('#')) **Output** ['python', 'is', 'easy', 'to', 'learn'] >>>
zfill()	This method adds zeros at the beginning of the string until it reaches the specified length.	str='python' print(str.zfill(15)) **Output** 000000000python

Example 3.9.1 : Write a function that takes a list of words and returns the length of the longest one.

Solution :

```
def my_function():
    word_list=[]
    n = int(input("Enter the number of words in list:")) #Reading number of words
    for i in range(0,n):
        word=input("Enter the word:" ) #inputting each word in the list
        word_list.append(word)
    longest=len(word_list[0])  #Assuming length of first word as longest
    temp=word_list[0]  #taking the first word in variable temp
```

4 Python Functions, Modules and Packages

4.1 Use of Python Built in Functions

What is function ?

- Function is a block of instructions that performs a specific and well defined task. This block can be repeatedly called by other programming instruction.
- Every function is specified by a name.
- **Following are some reasons why do we need to use functions in python.**

1. All the logically related statements can be grouped together in one entity. This makes the program easy to read, understand and debug.
2. The repetition of frequently required code can be avoided by using function.
3. Dividing a long program into functions allows us to debug the parts one at a time and then assemble them into a working whole.
4. Once the code of the function is written and tested, we can reuse this code. Reusability is an important use of function.

Review Question

1. *What is function ? Explain the need for writing a function in python.*

4.2 User Defined Functions

4.2.1 Function Definition

The function can be defined using the **def**.

The **syntax** of function definition is as follows -

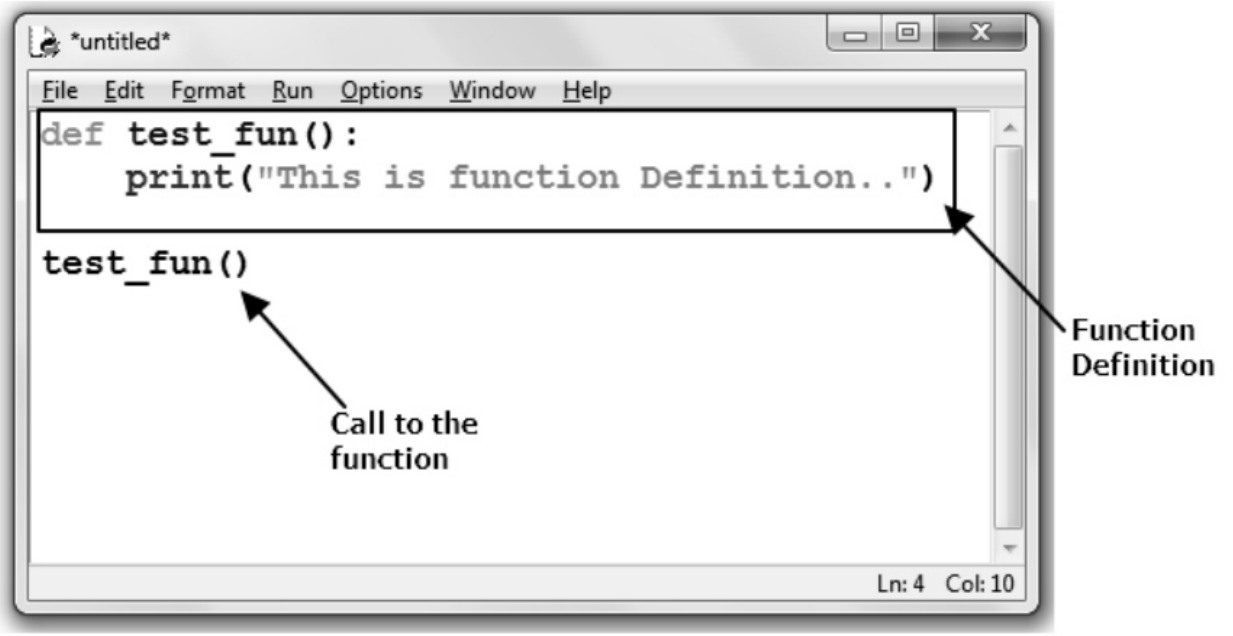
```
def function_name(parameters):
    statements
```

This colon is must

Example of creation of function

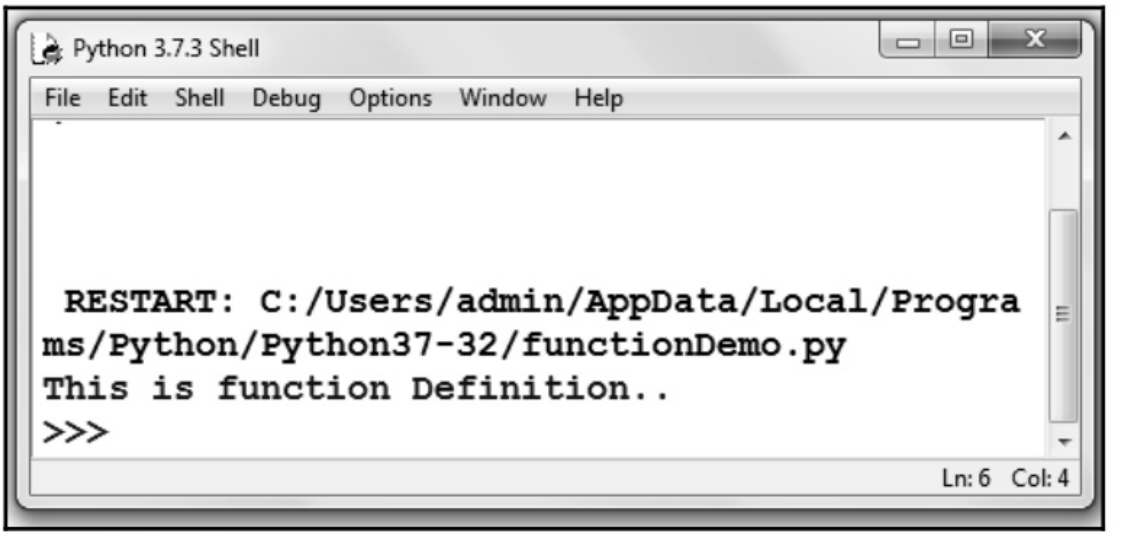
Step 1 : Create a function definition in a file. Open **File->New File** and type following code.

```
def test_fun():
  print("This is function Definition..")

test_fun()
```

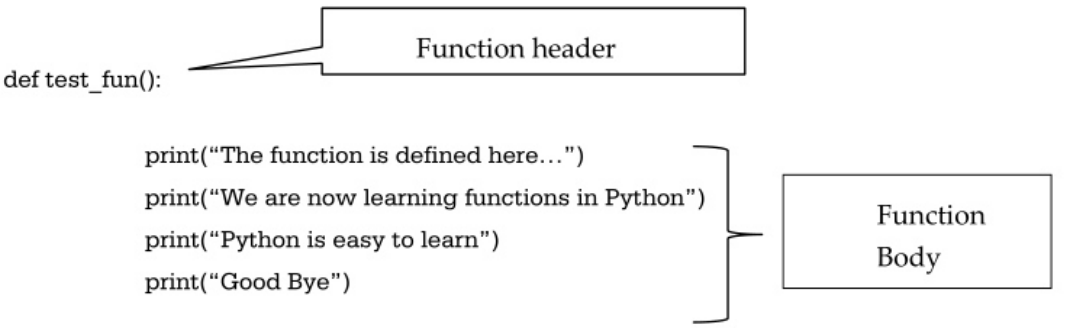


Step 2 : Now save the above file using some suitable file name. For that click on **File->Save.** I have given the name **functionDemo.py.**

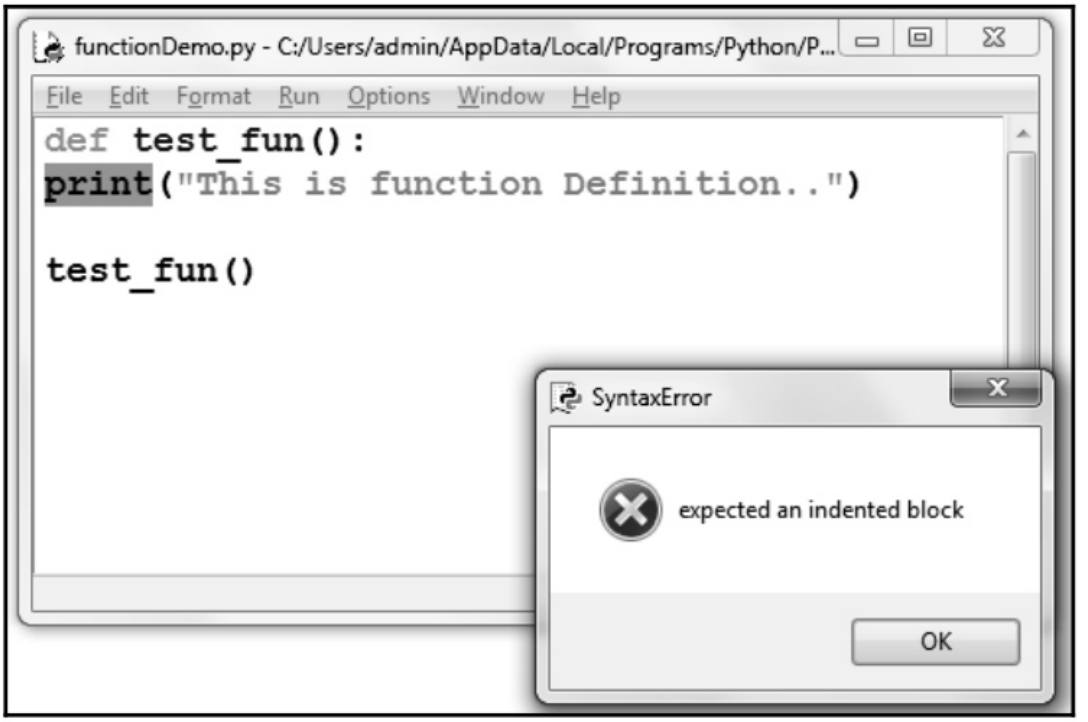
Step 3 : Press F5 button and run the above program. You will get the following output –



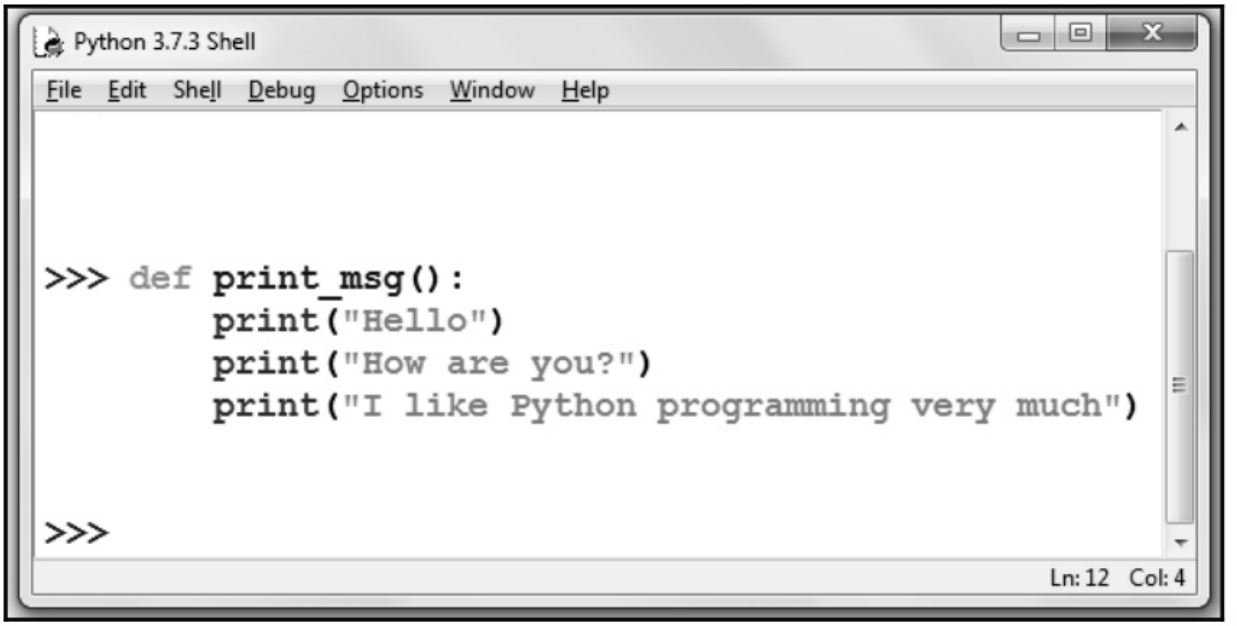
Here is a sample function.



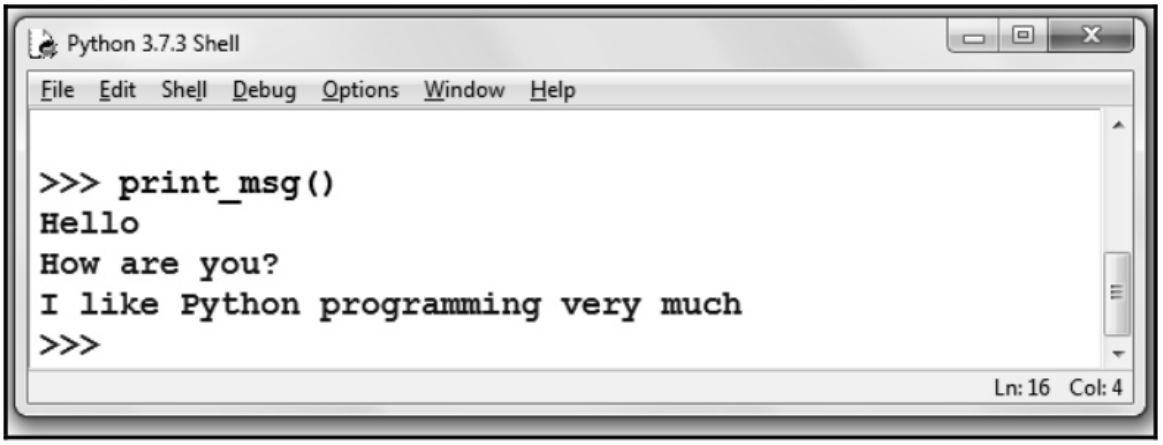
- The first line of function definition is called function **header** and rest is called **body.**
- Note that - The code within every function **starts with a colon (:)** and **should be indented (space)**.
- Python functions don't have any explicit begin or end like curly braces to indicate the start and stop for the function, they have to rely on this indentation.
- **For example :** If we write the above function without indentation, then error will occur. Following screenshot illustrates this -



- If you type a function definition in interactive mode, the interpreter prints dots (...) to let you know that the definition is not complete. For example :



- Then the call to the function in interactive mode is given as follows –



4.2.2 Function Calling

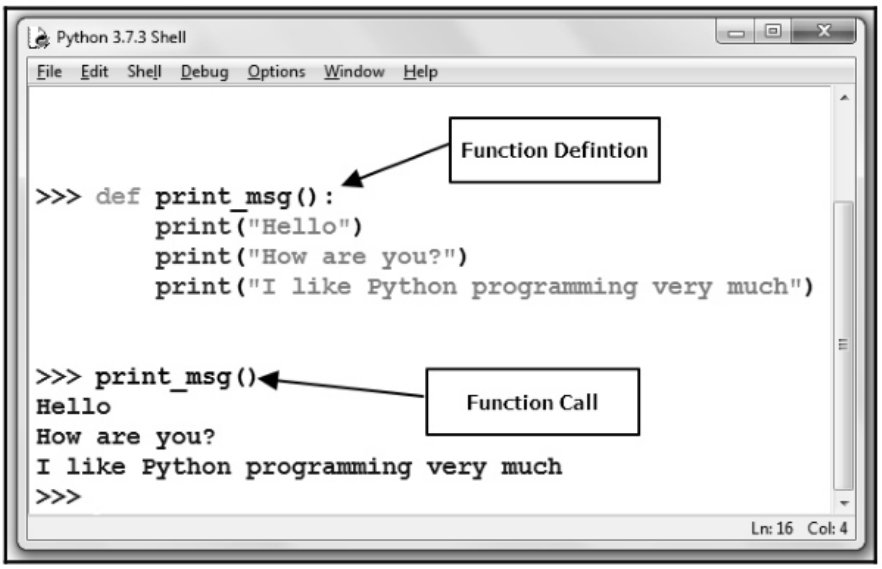
- The functional call is a statement that invokes the function.
- When a function is called the program control jumps to the definition of the function and executes the statements present in the function body.
- Once all the statements in function body get executed, the control goes back to the calling function.
- **Syntax**

  Function_name()

- **Example**

```
>>> def print_msg():
        print("Hello")
        print("How are you?")
        print("I like Python programming very much")
#call to the function
>>> print_msg()
Hello
How are you?
I like Python programming very much
>>>
```

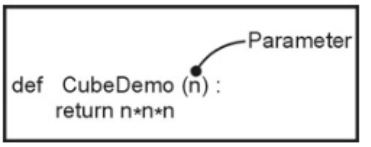
The following is a screenshot of the above code.



4.2.3 Function Argument and Parameter Passing

- The argument is a value that is passed to the function when it is called.
- **Syntax**

  Function_name(variable1, variable2,…)
- On the calling side, it is an **argument** and on the function side it is a **parameter.**
- For example - Consider following function in which there is single parameter passed to the function **CubeDemo.**

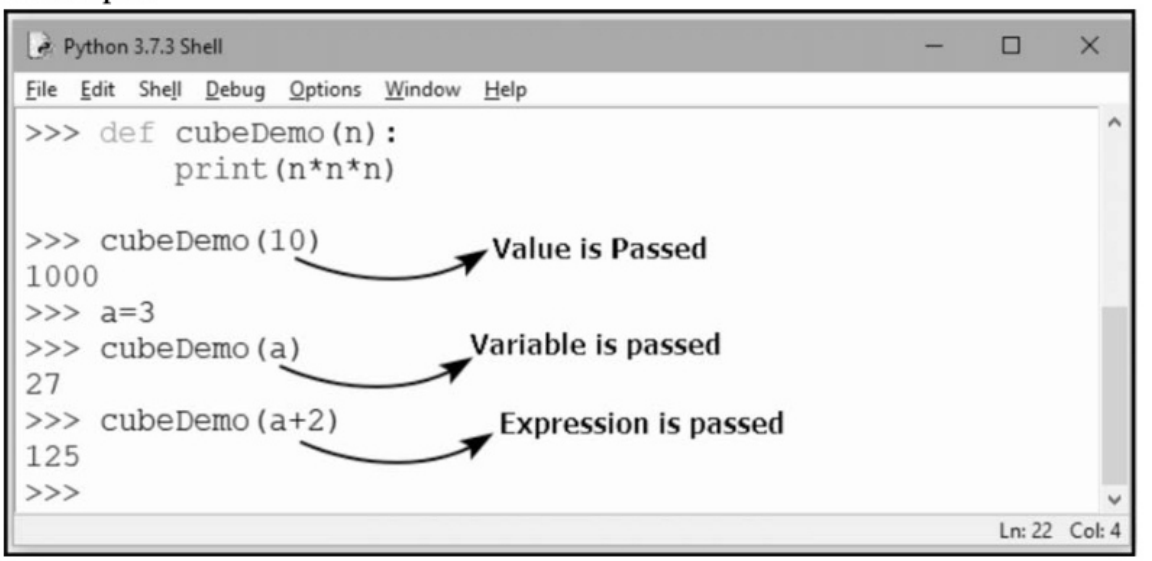


- We can pass

  1) Directly a value as a parameter or

  2) A variable or

  3) An expression.
- **For example :**



4.2.4 Return Statement

- The value can be returned from a function using the keyword **return**. For example
- **Syntax**

  return [expression_list]

Uses of Return Statement

1) The return statement is used to return a value from a function to the caller.
2) The return statement is sometimes used to return a control back to the caller.

Example 4.2.1 : Write a Python program to compute area of circle using function and a return statement.
Solution :

```
def areaCircle(r):
PI = 3.14
result=PI*r*r
print("The result of circle: ")
return result
```

Output

```
>>> areaCircle(10)
The result of circle:
314.0
>>>
```

4.2.5 Scope of Variables

- **Scope** of the variable means - the part of the program in which a variable is accessible.
- **Lifetime** of variable means - duration for which the variable exists.

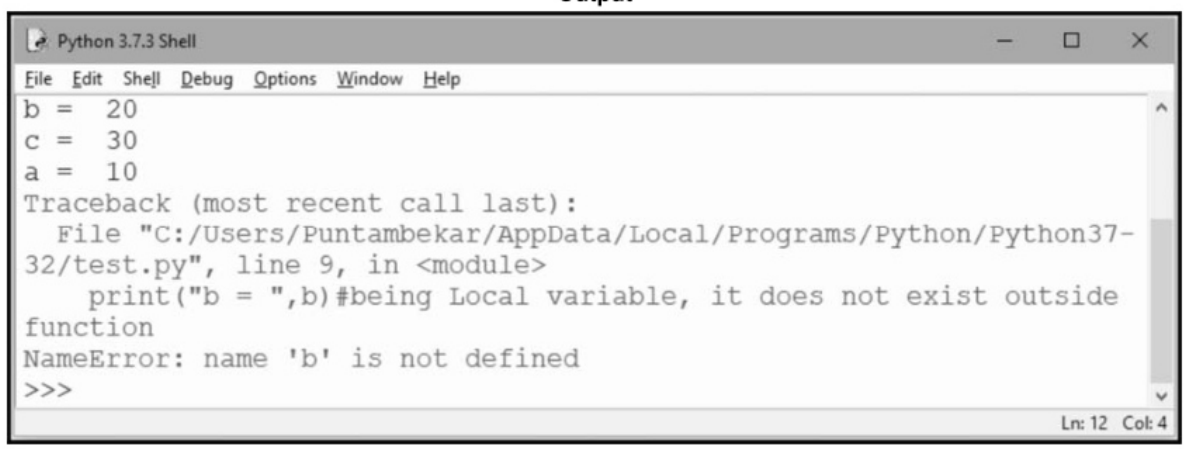
Local and Global Variable

- **Definition of Global Variable :** A variable which is defined in the main body of a file is called a **global variable.** It will be visible throughout the file, and also inside any file which imports that file.
- Only objects which are intended to be used globally, like functions and classes, should be put in the **global namespace**.
- **Definition of Local Variable :** A variable which is defined inside a function is local to that function. It is accessible from the point at which it is defined until the end of the function, and exists for as long as the function is executing.
- The parameter names in the function definition behave like local variables, but they contain the values that we pass into the function when we call it.
- When we use the assignment operator (=) inside a function, its default behaviour is to create a new local variable.

For example -

```
a=10 # a is Global Variable
def My_Function(b):# b is Local variable
c=30 # c is Local variable
print("b = ",b)
print("c = ",c)

My_Function(20)
print("a = ",a)#a still exists
print("b = ",b)#being Local variable, it does not exist outside function
print("c = ",c)#being Local variable, it does not exist outside function
#Error: for b and c variable
```

Output



Difference between Local and Global Variable

Sr. No.	Local Variable	Global Variable
1.	A variable which is declared inside a function is called as local variable.	A variable that is declared outside the function is called global variable.
2.	Accessibility of variable is only within the function in which it is declared.	Accessibility of variable is throughout the program. All functions in the program can access the global variable.
3.	The local variable is created when the function(in which it is defined) starts executing and it is destroyed when the execution is complete.	The global remains throughout the entire program execution.
4.	It is preferred as local variables are more reliable. Because the values cannot be changed outside the function.	It is generally not preferred as any function can change the values of global variable.

4.2.6 Python Programs based on Functions

Example 4.2.2 : Write a Python program to find the maximum of three numbers using function.

Solution :

```
def largest_among_three(a,b,c):
   if(a>b and a>c):
      print("a is largest");
   elif(b>a and b>c):
      print("b is largest")
   else:
      print("c is largest")

print("\t\tProgram for finding largest of three numbers")
print("Enter value of a: ")
a=int(input())
print("Enter value of b: ")
b=int(input())
print("Enter value of c: ")
c=int(input())
largest_among_three(a,b,c)
```

Output(Run1)

```
      Program for finding largest of three numbers
Enter value of a:
10
Enter value of b:
20
Enter value of c:
30
c is largest
>>>
```

Output(Run2)

```
    Program for finding largest of three numbers
Enter value of a:
20
Enter value of b:
10
Enter value of c:
5
a is largest
>>>
```

Output(Run3)

```
    Program for finding largest of three numbers
Enter value of a:
10
Enter value of b:
30
Enter value of c:
20
b is largest
>>>
```

Example 4.2.3 : Write a Python program to swap two numbers using function.

Solution :

```
def swap(a,b):
   temp=a
   a=b
   b=temp
   print("After swapping")
   print("a: ",a)
   print("b: ",b)


print("\t\t Program for swapping two numbers")
print("Enter value of a: ")
a=int(input())
print("Enter value of b: ")
b=int(input())
print("Before swapping")
print("a: ",a)
print("b: ",b)

swap(a,b)
```

Output

```
Program for swapping two numbers
Enter value of a:
10
Enter value of b:
20
Before swapping
a:  10
```

```
b:  20
After swapping
a:  20
b:  10
>>>
```

Example 4.2.4 : Write a Python program to calculate simple interest using function.

Solution :

```
def interest_calculation(p,n,r):
    interest=0.0
    interest=float(p*n*r)/float(100)
    return interest

print("\t\t Program for calculating simple interest")
print("Enter value of p: ")
p=float(input())
print("Enter value of n: ")
n=int(input())
print("Enter value of r: ")
r=float(input())

print("Simple Interest = ",interest_calculation(p,n,r))
```

Output

```
      Program for calculating simple interest
Enter value of p:
10000
Enter value of n:
5
Enter value of r:
6.5
Simple Interest =  3250.0
>>>
```

Example 4.2.5 : Write a Python program to reverse the given number using function.

Solution :

```
def Reverse(n):
    rev=0
    while(n>0):
        remainder=n%10
        rev=(rev*10)+remainder
        n=n//10
    return rev

print("\t\t Program for Reversing a number")
print("Enter value of n: ")
n=int(input())
print("The reverse of ",n," is ",Reverse(n))
```

Output

```
    Program for Reversing a number
Enter value of n:
1234
The reverse of  1234  is  4321
>>>
```

Example 4.2.6 : Write a python program to find out the factorial of a given number

Solution :

```
def factorial(n):
   if(n==1):
      return n
   else:
      return n*factorial(n-1)

print("\t\t Program for Factorial a number")
print("Enter value of n: ")
n=int(input())
if(n<0):
   print("Can not find factorial of a negative number")
elif (n==0):
   print("Factorial of 0 is 1")
else:
   print("The reverse of ",n," is ",factorial(n))
```

Output

```
      Program for Factorial a number
Enter value of n:
5
The reverse of  5  is  120
>>>
```

Example 4.2.7 : Write a Python program using function to check if the entered year is leap year or not

Solution :

```
def leap(year):
   if((year % 4 == 0) and (year % 100 != 0)):
      print(year, "is a Leap Year")
   elif (year % 400 == 0):
      print(year, "is a Leap Year")
   elif (year % 100 == 0):
      print(year, "is not a Leap Year")
   else:
      print(year, "is not a Leap Year")

print("\t\t Program for Check whether the year is leap year or not")
print("Enter the value of year: ")
year=int(input())
leap(year)
```

Output(Run 1)

```
      Program for Check whether the year is leap year or not
Enter the value of year:
2001
2001 is not a Leap Year
>>>
```

Output(Run 2)

```
 Program for Check whether the year is leap year or not
Enter the value of year:
2000
2000 is a Leap Year
>>>
```

Example 4.2.8 : Write a Python program check the number is prime or not.

Solution :

```
def isprime(num):
   if num > 1:
      for i in range(2, num):
         if(num%i)==0:
            print(num," is not a prime number")
            print(i," times",num//i," is ",num)
            break
      else:
         print(num," is a prime number")
   else:
      print(num," is not a prime number")


print("\t\t Program for Check whether the number is prime or not")
print("Enter the number: ")
num=int(input())
isprime(num)
```

Output(Run 1)

```
Program for Check whether the number is prime or not
Enter the number:
13
13  is a prime number
>>>
```

Output(Run2)

```
Program for Check whether the number is prime or not
Enter the number:
12
12  is not a prime number
2  times 6  is  12
>>>
```

Example 4.2.9 : Write a Python program to find the list of prime numbers from given range.

Solution :

```
def prime(lower,upper):
   for num in range(lower,upper + 1):
      if num > 1:
         for i in range(2,num):
            if (num % i) == 0:
```

```
            break
        else:
            print(num)
print("\t\t Program to Find is prime numbers from a range")
print("Enter the range[lower-upper]: ")
lower=int(input())
upper=int(input())
print(" The prime numbers are...")
prime(lower,upper)
```

Output

```
    Program to Find is prime numbers from a range
Enter the range[lower-upper]:
1
100
 The prime numbers are...
2
3
5
7
11
13
17
19
23
29
31
37
41
43
47
53
59
61
67
71
73
79
83
89
97
>>>
```

Example 4.2.10 : Write a Python program using function to check if the number is perfect or not.

Solution :

A perfect number is a positive integer that is equal to the sum of its proper positive divisors, that is, the sum of its positive divisors excluding the number itself.

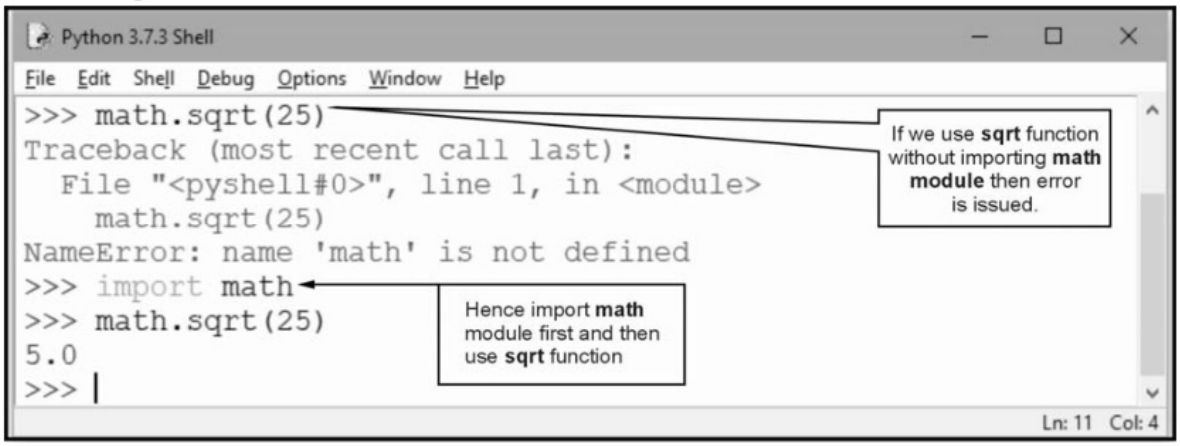
For example – Consider number 28 =1 + 2 + 4 + 7 + 14.

Equivalently, a perfect number is a number that is half the sum of all of its positive divisors (including itself).

4.3 Modules

4.3.1 Introduction to Modules

- There are some standard functionalities in python that are written in particular module. For using those functionalities, it is essential to import the corresponding module first.
- For example : There are various functionalities available under the module math. For instance, if you want to find out the square root of some number then you need to import module math first and then use the sqrt function.
- **Following screenshot illustrates this idea :**



- Basically modules in python are .py files in which set of functions are written.
- Modules are imported using **import** command.

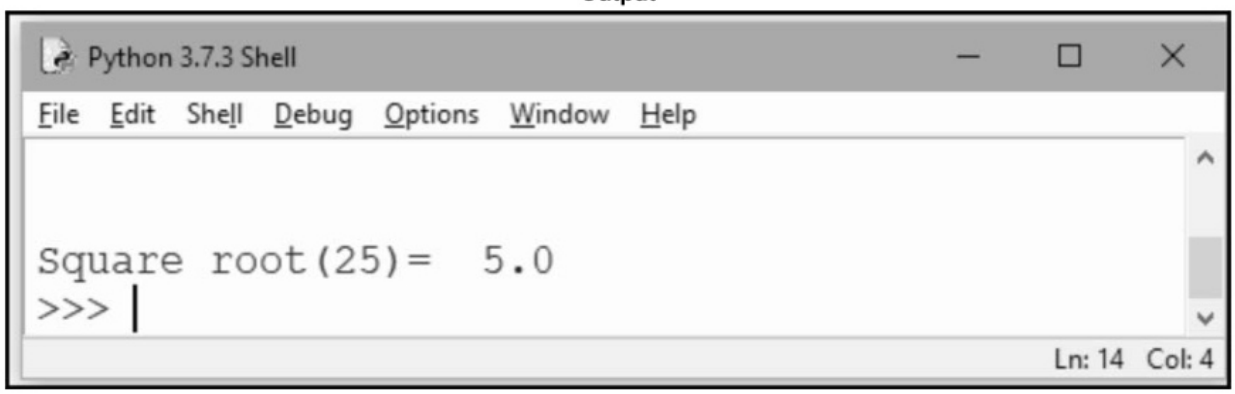
4.3.2 Importing Objects from Modules

- There are many variables and functions present in the module. We can use them by using the import statement.
- When we simply use **import** statement, then we can use any variable or function present within that module.
- When we use **from…import** statement, then we can use only selected variables or functions present within that module.

For example

```
from math import sqrt
print("Square root(25)= ",sqrt(25))
```

Output



- If we want to use some different name for the standard function present in the module, then we can use **as** keyword. Following code illustrates this –

moduleProg1.py

```
from math import sqrt as my_sq_root
print("Square root(25)= ",my_sq_root(25))
```

Output

```
Square root(25)=  5.0
>>>
```

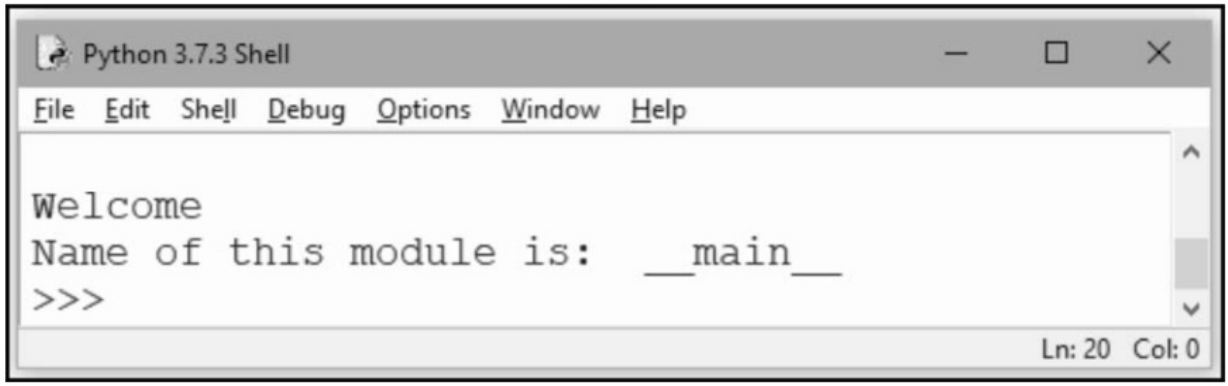
4.3.3 Name of the Module

- Every module has a name. One can use the name of the module using __name__ attribute of the module. For example

moduleProg3.py

```
print("Welcome")
print("Name of this module is: ",__name__)
```

Output



The __name__ is a built-in variable that is set when the program starts. If the program is running as a script, __name__ has the value '__main__'

For every standalone program written by the user the module is always __main__. Hence is the output.

4.3.4 Writing Modules

- We can create our own module. Every Python program is a module. That means every file that we save using .py extension is a module.
- Following are the steps that illustrates how to create a module –

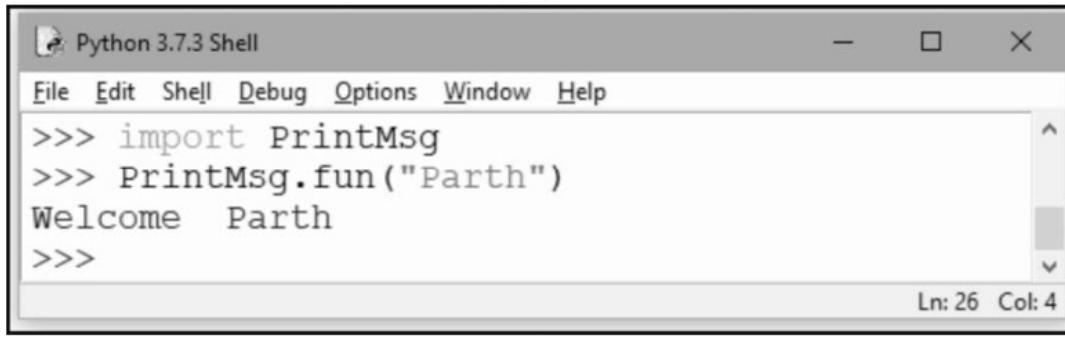
Step 1 : Create a file having extension .py. Here we have created a file PrintMsg.py. In this file, the function fun is defined.

PrintMsg.py

```
def fun(usr):
  print("Welcome ",usr)
```

Step 2 : Now open the python shell and import the above module and then call the functionality present in that module.

Output

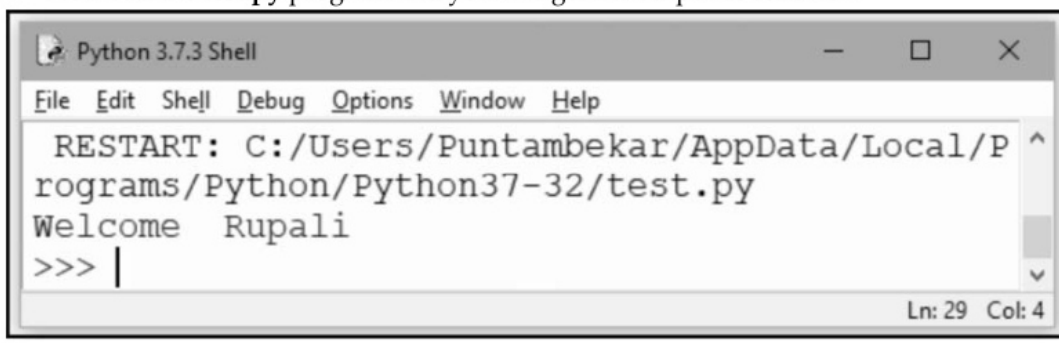


Step 3 : We can also call the above created **PrintMsg** module in some another file. For that purpose, open some another file and write the code in it as follows –

Test.py

```
import PrintMsg  #importing the user defined module
PrintMsg.fun("Rupali")  #calling the function present in that module
```

Step 4 : Now run the **test.py** program and you will get the output as follows -



Example 4.3.1 : *Write a module for displaying the Fibonacci series.*

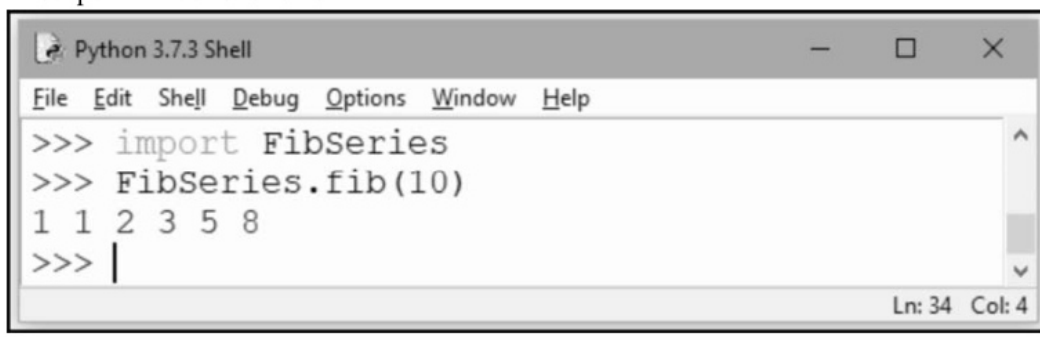
Solution :

Step 1 :

FibSeries.py

```
def fib(n):    # write Fibonacci series up to n
    a, b = 0, 1
    while b < n:
        print(b, end=' ')
        a, b = b, a+b
    print()
```

Step 2 : Open the python shell and import the above module and access the functionality within it. The output will be as follows :



Example 4.3.2 : *Write a Python program to define a module for swapping the two values. Then write a driver program that will call the function defined in your swap module.*

Solution :

Step 1 : Create a module in a file and write a function for swapping the two values. The code is as follows –

swapProg.py

```
def Swap(a,b):
    temp=a
    a=b
    b=temp
    print("After Swapping")
    print(a)
    print(b)
```

Save and Run the above code.

Step 2 : Now create a driver program that will invoke the swap function created in above module.

Test.py

```
import swapProg
print("Swap(10,20)")
swapProg.Swap(10,20)
print("Swap('a','b')")
swapProg.Swap('a','b')
```

Step 3 : Save and run the above code. The output will be as follows –

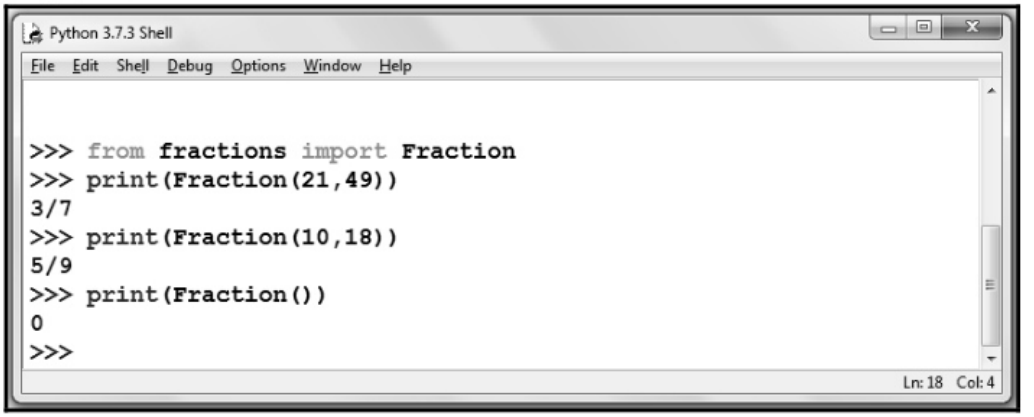
4.3.5 Python Built in Modules

1. Numeric and Mathematical Modules

- The **numbers** module defines a hierarchy of numeric abstract base classes which progressively define more operations. None of the types defined in this module can be instantiated.
- The **fractions** module is used to return the instance with the value=numerator/denominator. For example

  21/49 = 3/7

- Following Python code makes use of fractions module.



- The **math** module used in Python is for various mathematical functionalities such as finding log, computing sine or cosine values, calculating square root and so on.

Some of the functions supported by math module are enlisted as follows –

Function	Purpose
ceil(x)	Returns the smallest integer greater than or equal to x.
factorial(x)	Returns the factorial of x
floor(x)	Returns the largest integer less than or equal to x
isinf(x)	Returns True if x is a positive or negative infinity
exp(x)	Returns e**x
log(x[, base])	Returns the logarithm of x to the base (defaults to e)
pow(x, y)	Returns x raised to the power y
sqrt(x)	Returns the square root of x
cos(x)	Returns the cosine of x
sin(x)	Returns the sine of x
tan(x)	Returns the tangent of x
pi	Mathematical constant, the ratio of circumference of a circle to it's diameter (3.14159...)
e	mathematical constant e (2.71828...)

For example

i) Finding square root of a number.

```
>>> import math
>>> print(math.sqrt(25))
5.0
>>>
```

(ii) Finding gcd(greatest common divisor) of two numbers.

```
>>> import math
>>> print(math.gcd(12,15))
3
>>>
```

(iii) Finding Factorial of a given number.

```
>>> import math
>>> print(math.factorial(5))
120
>>>
```

(iv) Finding power i.e. x^y.

```
>>> import math
>>> print(math.pow(2,3))
8.0
>>>
```

(v) Finding remainder.

```
>>> import math
>>> print(math.remainder(10,3))
1.0
>>>
```

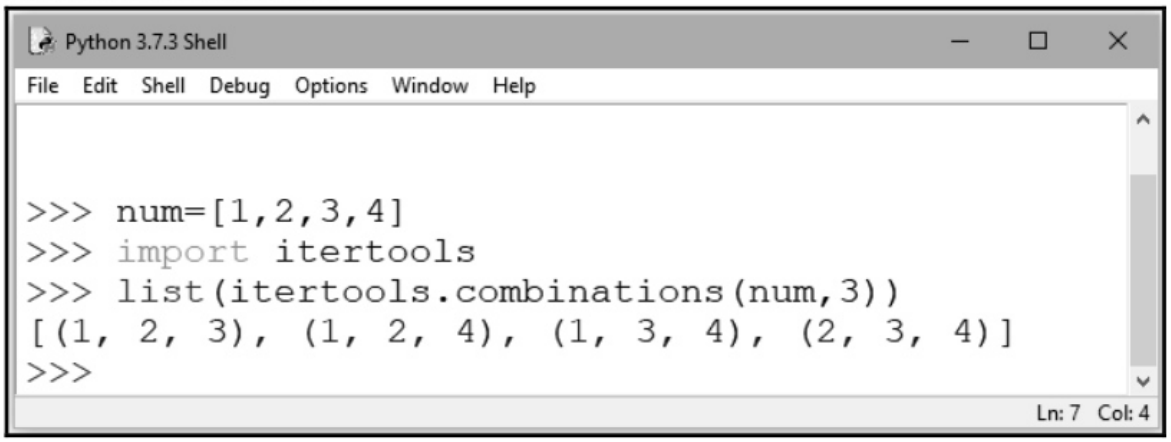
(vi) Displaying the pi constant.

```
>>> import math
>>> print(math.pi)
3.141592653589793
>>>
```

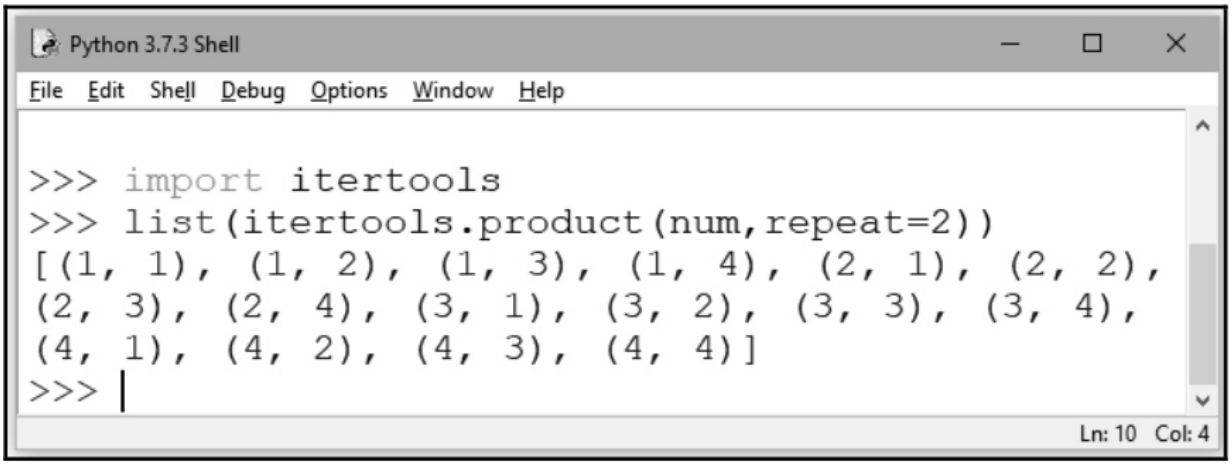
2. Functional Programming Module

The functional programming module provides functions and classes that support functional programming style.

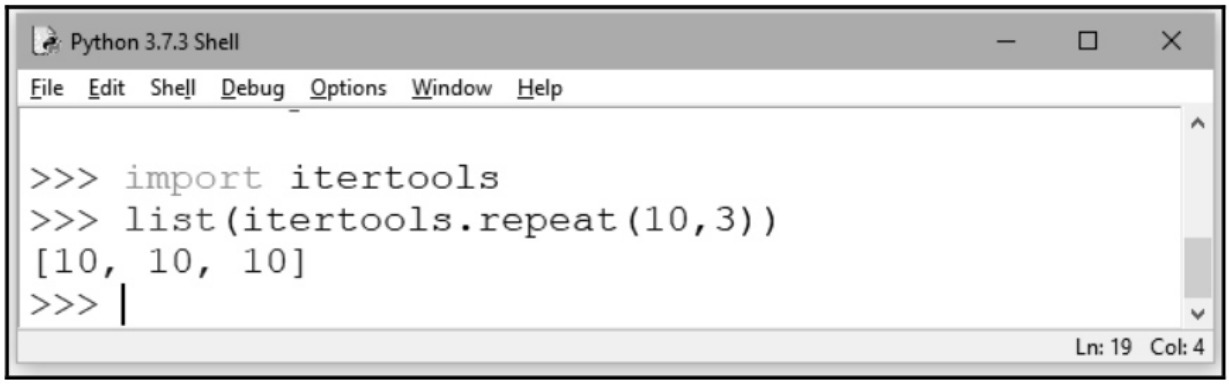
For creating iteration or repetition the **itertools** module is used. There are various functionalities that can be used from itertools module.



Similarly we can have cartesian product, equivalent to a nested for-loop.



We can repeat the numbers using the itertools modules. Following screenshot illustrates this -



Use of map function

The map( ) function applies the values to every member of the list and returns the result.

Syntax : The map function takes two arguments -

```
result=map(function, sequence)
```

For example

```
>>> def increment_by_three(x):
return x+3
>>> new_list=list(map(increment_by_three,range(10)))
>>> new_list
[3, 4, 5, 6, 7, 8, 9, 10, 11, 12]
>>>
```

Another Example :

There is a built in function for calculating length of the string and i.e. len( ). We can pass this function to map and find the length of every element present in the list as a list.

```
>>> names=['Sandip','Lucky','Mahesh','Sachin']
>>> lengths=list(map(len,names))
>>> lengths
[6, 5, 6, 6]
>>>
```

4.3.6 Namespace and Scope

- Namespace is basically a collection of different names. Different namespaces can co-exist at a given time but are completely isolated.
- If two names(variables) are same and are present in the same scope then it will cause name clash. To avoid such situation we use the keyword namespace.
- In Python each module has its own namespace. This namespace includes all the names of its function and variables.
- If we use two functions having same name belonging to two different modules at a time in some other module then that might create name clash. For instance –

Step 1 : Create first module for addition functionality.

FirstModule.py
```
def display(a,b):
    return a+b
```

Step 2 : Create second module for multiplication functionality.

SecondModule.py
```
def display(a,b):
    return a*b
```

Step 3 : If we call above modules in our driver program for performing both addition and multiplication, then we get error because of name clash for the function.

Test.py
```
import FirstModule
import SecondModule
print(display(10,20))
print(display(5,4))
```

Step 4 : To resolve this ambiguity we should call these functionalities using their module names. It is illustrated as follows –
```
import FirstModule
import SecondModule
print(FirstModule.display(10,20))
print(SecondModule.display(5,4))
```

Output
```
30
20
```

4.3.6.1 Global, Local and Built-in Namespace

- There are three commonly used categories of namespaces – **Global namespace**, **local namespace**, and **built-in namespace**.
- The global namespace contains the module which is currently executing. The local namespace is a namespace for defining the names in a local function. The built-in namespace is a namespace in which the built in functionality can be invoked.
- Following Python code represents all these namespaces.

```
import math
def even_number(number):  #global namespace
   num=number #local namespace
   if(num%2==0):
      return num
   else:
      return 0

print("Enter some number")
num=int(input())
if(even_number(num)!=0):
  print("The square root of ",num,"is ",math.sqrt(num)) #built-in namespace
```

Review Questions

1. *What is module in Python ?*
2. *How will you import objects from modules?*
3. *Explain any two built in modules in Python.*
4. *Write a short note on – Namespace and scoping.*
5. *Write a Python program to demonstrate global, local and built in namespace.*

4.4 Python Packages

4.4.1 Introduction to Packages

- Packages are namespaces which contain multiple packages and modules. They are simply directories.
- Along with packages and modules the package contains a file named _ _init_ _.py. In fact **to be a package, there must be a file called _ _init_ _.py in the folder.**
- Packages can be nested to any depth, provided that the corresponding directories contain their own __init__.py file.
- The arrangement of packages is as shown in Fig. 4.4.1.

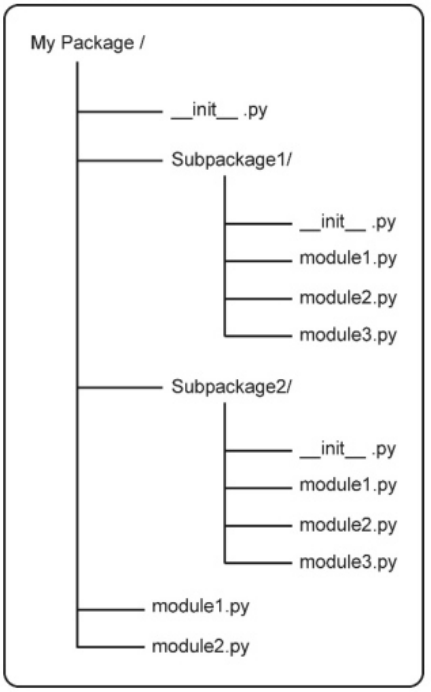


Fig. 4.4.1

For example :

We can access the package and various subpackages and modules in it as follows :

```
import My_Package #loads My_Package/__init__.py
import My_Package.module1 #loads My_Package/module1.py
from My_Package import module2
import My_Package.SubPackage1
```

- When we import Main package then there is no need to import subpackage. For example

```
import My_Package
My_Package.SubPackage1.my_function1()
```

- Thus we need not have to import SubPackage1.
- The primary use of __init__.py is to initialize Python packages. Simply putting the subdirectories and modules inside a directory does not make a directory a package, what it needs is a __init__.py inside it. Then only Python treats that directory as package.

4.4.2 Writing Python Packages

- To understand how to create a package, let us take an example. We will perform arithmetic operations addition, multiplication, division and subtraction operations by creating a package. Just follow the given steps to create a package.

Step 1 : Create a folder named **MyMaths** in your working drive. I have created it on D: drive.

Step 2 : Inside MyMaths directory create **__init__.py** file. Just create an empty file.

Step 3 : Create a folder named **Add** inside **MyMaths.** Inside **Add** folder create file named **addition.py.** The **addition.py** will be as follows

addition.py

```
def add_fun(a,b):
  return a+b
```

Step 4 : Similarly the folder **Sub** insider MyMaths is created. Inside which create a file **subtraction.py.** The code for this file is

subtraction.py

```
def sub_fun(a,b):
  return(a-b)
```

Step 5 : Similarly the folder **Mul** insider MyMaths is created. Inside which create a file **multiplication.py.** The code for this file is

multiplication.py

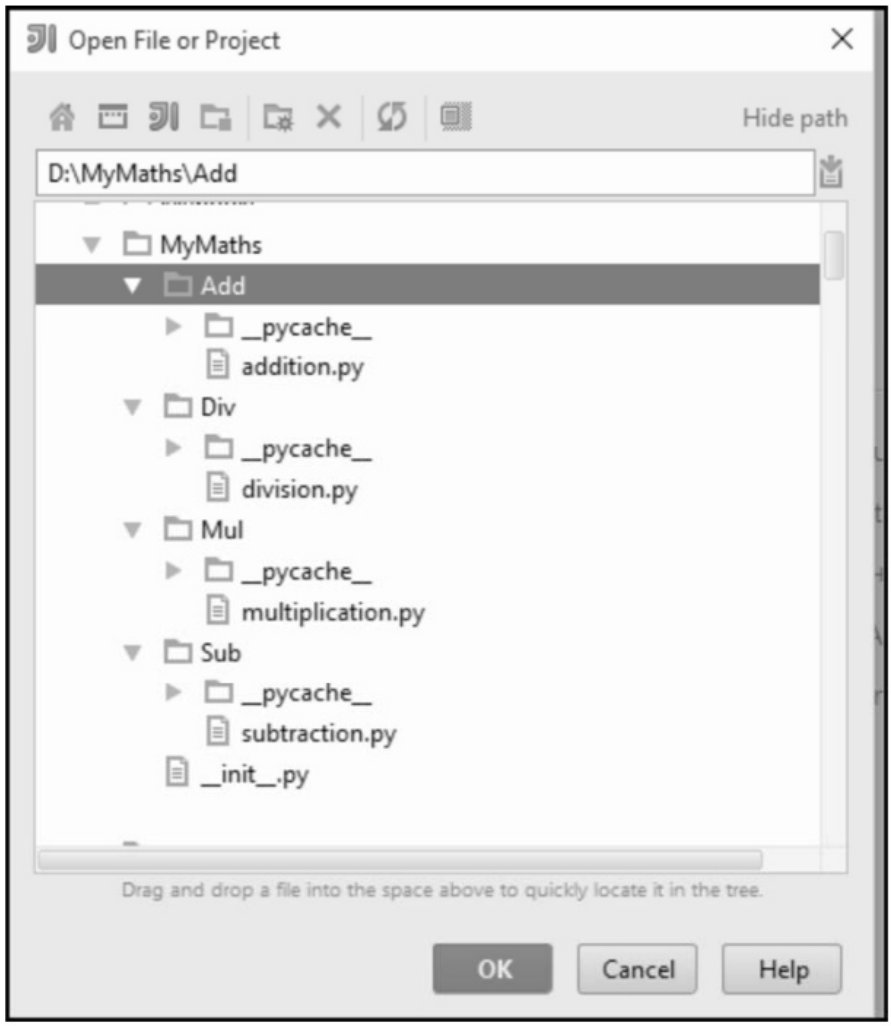
```
def mul_fun(a,b):
  return(a*b)
```

Step 6 : Similarly the folder **Div** insider MyMaths is created. Inside which create a file **division.py.** The code for this file is

division.py

```
def div_fun(a,b):
  return (a/b)
```

Step 7 : The overall directory Structure will be as given below. Note that _pycache_ is automatically generated directory.



Step 8 : Now on the D: drive create a driver program which will invoke all the above functionalities present in the MyMaths package. I have named this driver program as **testing_arithmetic.py**. It is as follows

```
import MyMaths.Add.addition
import MyMaths.Mul.multiplication
import MyMaths.Div.division
import MyMaths.Sub.subtraction
print("The addition of 10 and 20 is: ",MyMaths.Add.addition.add_fun(10,20))
print("The multiplication of 10 and 20 is: ",MyMaths.Mul.multiplication.mul_fun(10,20))
print("The division of 10 and 5 is: ",MyMaths.Div.division.div_fun(10,5))
print("The subtraction of 20 and 10 is: ",MyMaths.Sub.subtraction.sub_fun(20,10))
```

We need to import the modules in this way, to use the respective functionalities

Step 9 : Now just run this testing program and you will get following output

Output

```
The addition of 10 and 20 is:30
The multiplication of 10 and 20 is:200
The division of 10 and 5 is:2.0
The subtraction of 20 and 10 is:10
>>>
```

4.4.3 Using Standard Packages

In this section we will discuss how to install the standard packages of Python and use them

(1) Installation of numpy

Step 1 : Open command prompt and type the command

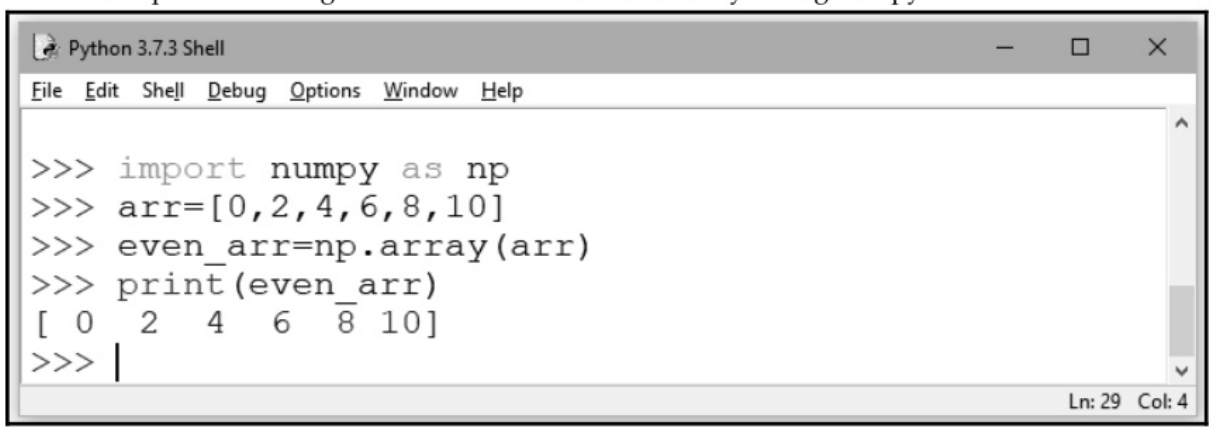
```
C:\> python -m pip install numpy
```

Step 2 : The package will be installed on your PC. On successful installation you will get the following message on your command prompt window.

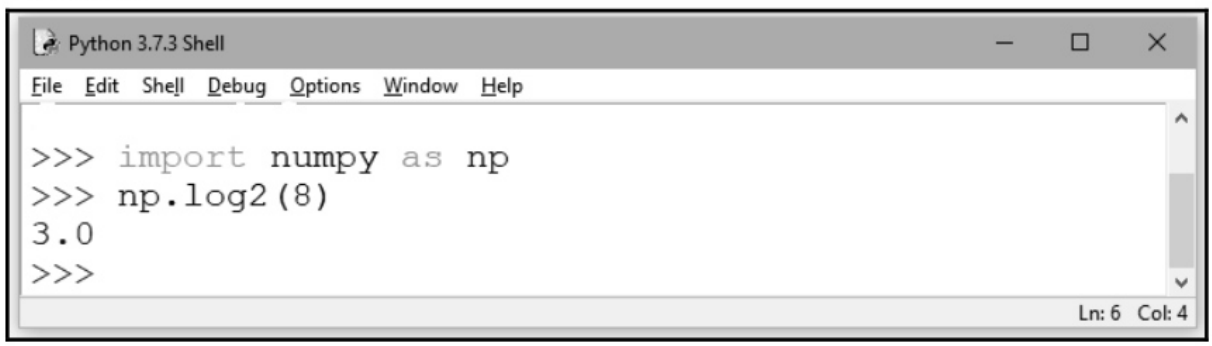
```
Successfully installed numpy-1.17.4
You are using pip version 19.0.3, however version 19.3.1 is available.
```

What is numpy Package ?

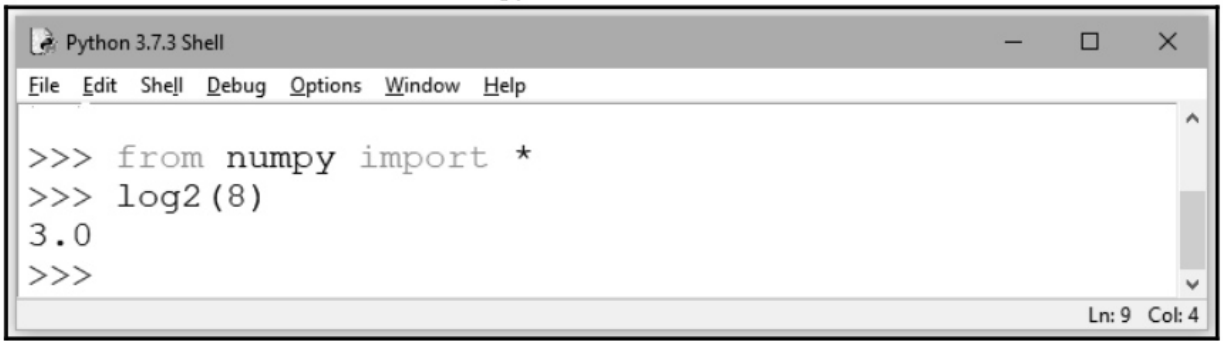
- Numpy is the core library for **scientific computing** in Python. It provides a high-performance multidimensional array object, and tools for working with these arrays.
- Advantages of using Numpy with Python :
  - Array oriented computing.
  - Efficiently implemented multi-dimensional arrays.
  - Designed for scientific computation.
- For example – Following screenshot show how to use arrays using numpy.



Some other scientific function such as finding log can be used using the numpy package. Here is an illustration -



Or we can also use the function from numpy as follows

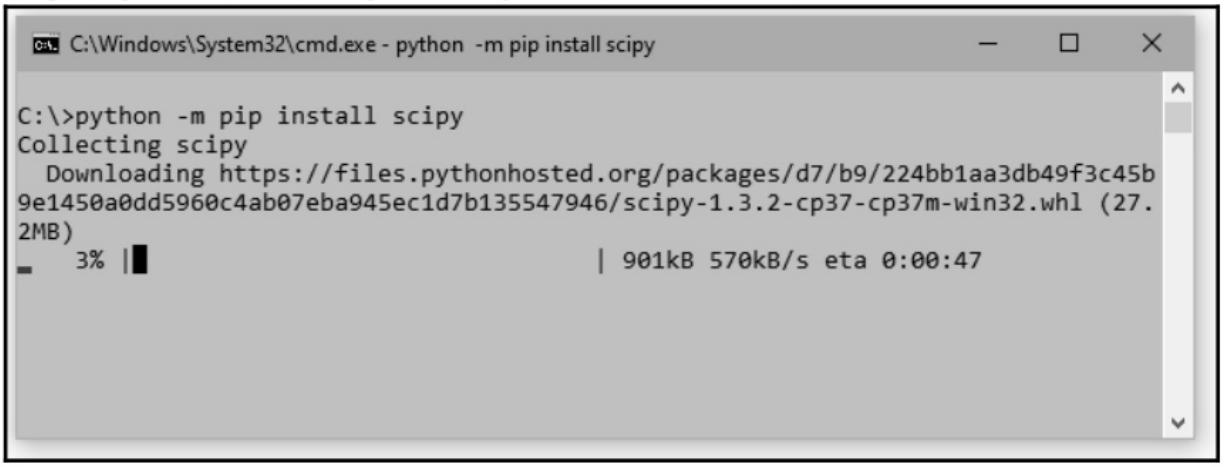


(2) Installation of scipy

Similar to numpy installation procedure, issue the following command at command prompt

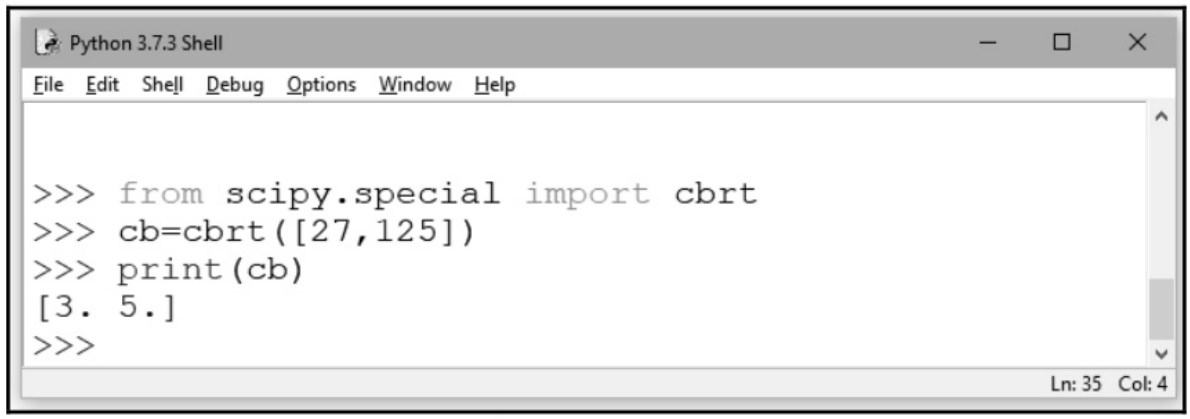
Python –m pip install scipy

The package will start installing. Following screenshot illustrates it.

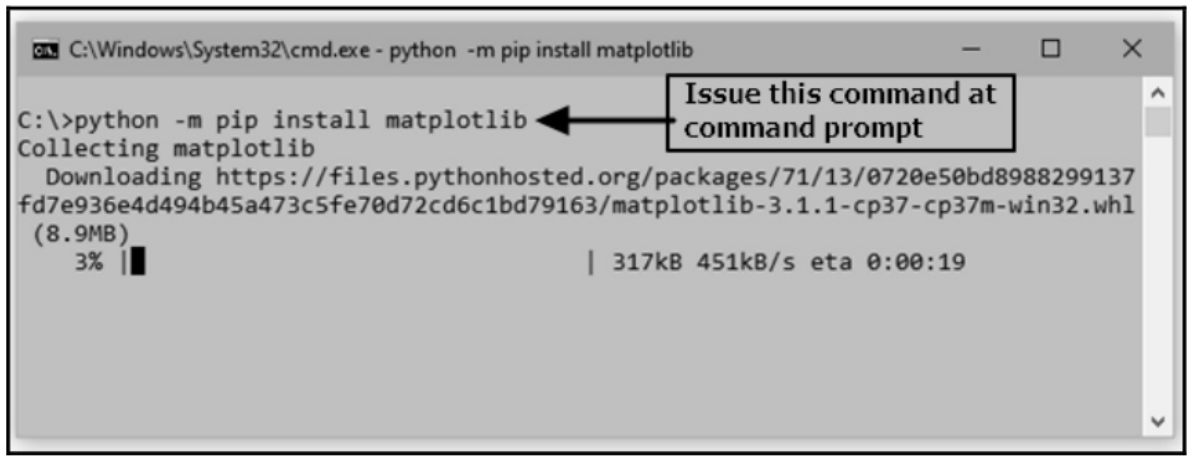


What is scipy Package ?

- SciPy is a collection of mathematical algorithms and convenience functions built on the NumPy extension of Python.
- SciPy is an Open Source Python-based library, which is used in mathematics, scientific computing, Engineering, and technical computing.
- SciPy also pronounced as "Sigh Pi."
- SciPy is a fully-featured version of Linear Algebra while Numpy contains only a few features.
- Most new Data Science features are available in Scipy rather than Numpy.
- **Example :** Following example shows how to compute cuberoot of values –



(3) Installation of matplotlib



What is matplotlib Package ?

- Matplotlib is a plotting library for the Python programming language and its numerical mathematics extension NumPy.
- Matplotlib consists of several plots like line, bar, scatter, histogram etc.
- We have to import the **pyplot** module of **matplotlib** library under the alias plt.
- With the help of plot function we can start preparing data for plotting and finally with the help of show function we can display the drawing on the screen.
- Let us understand how to plot the objects with the help of matplotlib.

Example 4.4.1 : Write a Python program to plot a line.

Solution : Step 1 : Open **File->New File** and type the following code in the file. Save this file with suitable name.

```
import numpy as np
import matplotlib.pylab as plt

arr=[5,10,15,20,25]
a=np.array(arr)

plt.plot(a)
plt.show()
```

(4) Installation of pandas

Similar to above installation procedures issue the following command at command prompt.

Python –m pip install pandas

The package will start installing. Following screenshot illustrates it.

Finally you will get the successful installation message on your command prompt.

What is Pandas Package ?

- Pandas is a Python package providing fast, flexible, and expressive data structures designed to make working with structured (tabular, multidimensional, potentially heterogeneous) and time series data both easy and intuitive.
- Pandas is well suited for many different kinds of data :
- **Tabular data** with heterogeneously-typed columns, as in an SQL table or Excel spreadsheet
- **Ordered and unordered** (not necessarily fixed-frequency) time series data.
- **Arbitrary matrix data** (homogeneously typed or heterogeneous) with row and column labels
- Any other form of **observational / statistical data sets**. The data actually need not be labeled at all to be placed into a pandas data structure.
- There are three data structures used in Panda
  - Series
  - DataFrame
  - Panel
- **Series :** It is a one dimensional homogeneous array. The size of this array is immutable, i.e. this size can not be changed.

A	B	C	D	E	F

Following is a Python script that represents the series

SeriesDemo.py

```
import pandas as pd
import numpy as np
a=np.array(['A','B','C','D','E'])
s=pd.Series(a)
print(s)
```

Output

```
0  A
1  B
2  C
3  D
4  E
dtype: object
>>>
```

DataFrame : It is a two dimensional heterogeneous array. The size is mutable i.e. it can be changed.

EmpNo	Name	Salary	Designation
101	AAA	10000	Accountant
102	BBB	15000	Executive
103	CCC	25000	Manager

The DataFrame can be implemented using following Python script.

Dframe.py

```
import pandas as pd
import numpy as np
a=[[10,20],[30,40]]
data_fr=pd.DataFrame(a)
print(data_fr)
```

Output

```
0  1
0  10  20
1  30  40
>>>
```

Another Example : We can create labeled column using dictionary

```
import pandas as pd
import numpy as np
dic={'EmpNo':[101,102,103],'Name':['AAA','BBB','CCC'],'Salary':[10000,15000,25000]}
data_fr=pd.DataFrame(data=dic)
print(data_fr)
```

Output

```
   EmpNo  Name  Salary
0   101    AAA   10000
1   102    BBB   15000
2   103    CCC   25000
>>>
```

- **Panel :** It is a three dimensional labeled array. The size is mutable.

  Syntax : The panel can be created using following syntax

  ```
  pandas.Panel(data, items, major_axis, minor_axis, dtype, copy)
  ```

where

- **data :** The data can be arrays,series,dict,list or another DataFrame.
- **items :** axis=0
- **major_axis:** axis=1

- **minor_axis:** axis=2
- **dtype:** The data type of each column
- **copy:** Copy data.

Review Questions

1. *What is package ? How will you create package in Python ?*
2. *What is packages ?*
3. *Explain the numpy package with suitable example.*
4. *What is the use of matplotlib package? Give an example.*

❑❑❑

5 Object Oriented Programming in Python

5.1 Introduction to Object Oriented Programming

- In this language everything is modeled as **object**. Hence is the name.
- This language has a modular programming approach in which data and functions are bound in one entity called class.
- This programming paradigm has gained a great popularity in recent years because of its characteristic features such as data abstraction, encapsulation, inheritance, polymorphism and so on.
- Examples of object oriented programming languages are - Smalltalk, C++, Java, Python.

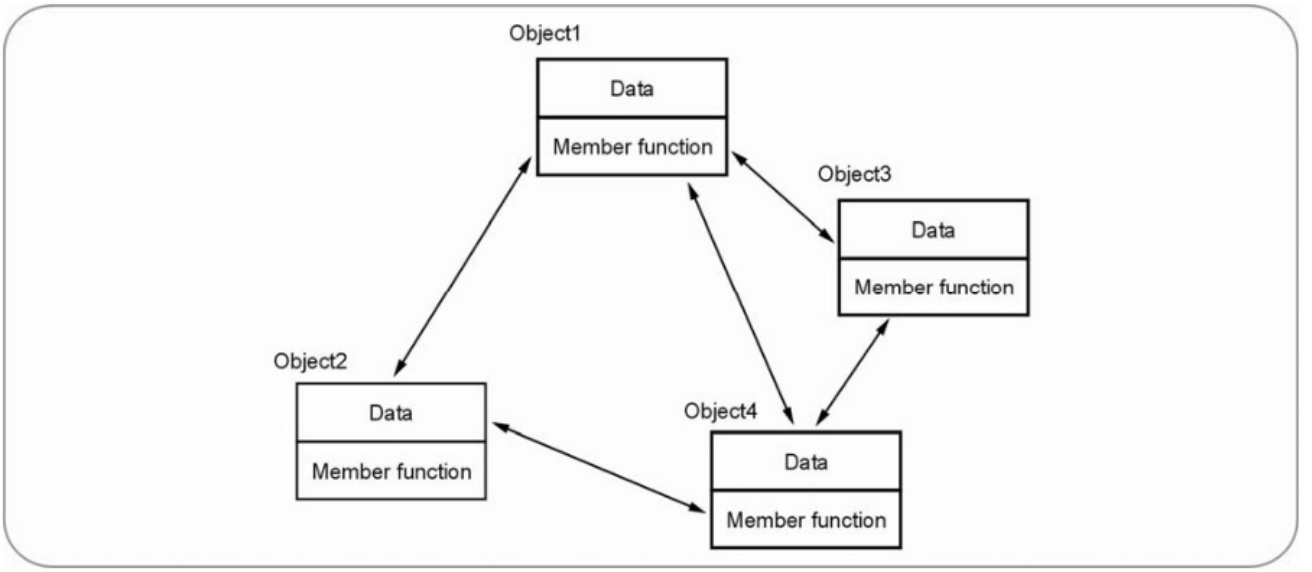


Fig. 5.1.1 Structure of object oriented programming

Advantages

1. It provides a modular programming approach.
2. It provides abstract data type in which some of the implementation details can be hidden and protected.
3. Modifying the code for maintenance become easy, due to modules in the program. Modification in one module can not disturb the rest of the code.
4. Finding bugs become easy.

5. Object oriented programming provides good framework for code library from which the software components can be easily used and modified by the programmer.

Disadvantages

1. Sometimes real world objects can not be realized easily. Hence it is **complex to implement**.
2. Some of the members (data or methods) of the class are **hidden**. Hence they can not be accessed by the objects of other class.
3. In object oriented programming everything is arranged in the form of classes and modules. For the **lower level** applications it is not desirable feature.

Difference between procedural and object oriented programming approach

Sr. No.	Procedural Programming Language	Object Oriented Programming Language (OOP)
1.	The procedural programming executes **series of procedures** sequentially.	In object oriented programming approach there is a **collection of objects.**
2.	This is a **top down** programming approach.	This is a **bottom up** programming approach.
3.	The major focus is on **procedures or functions**.	The main focus is on **objects.**
4.	**Data reusability** is not possible.	**Data reusability** is one of the important feature of OOP.
5.	Data hiding is not possible.	**Data hiding** can be done by making it private.
6.	It is **simple** to implement.	It is **complex** to implement.
7.	For example : C, Fortran, COBOL	For example : C++, JAVA

Review Questions

1 *What is object oriented programming? Give the advantages and disadvantages of it.*

2. *What is the difference between procedural and object oriented programming languages ?*

5.2 Features of Object Oriented Programming

5.2.1 Classes

- Class is a basic entity in object oriented programming.
- In any object oriented program the class is declared using the keyword **class**.
- Every class consists of data members and methods.
- For example -

```
Class Student:
    Roll=101
    Name="ABC"
obj=Student()
print("Roll Number: ",obj.Roll)
print("Name: ",obj.Name)"
```

5.2.2 Objects

- An object is an instance of a class.

- **For example**

```
Obj=Student() #creation of object for class 'Student'
```

- Objects are runtime entities of the object oriented programs.
- Objects communicate with each other using messages. While communicating with each other, it is not necessary to know the details of communicating object.
- Thus the internal details of each object can be hidden from another object.

Difference between class and objects

Sr.No.	Classes	Objects (instances)
1.	The class is collection of data and the functions. These functions are used to manipulate data.	An object is an instance of a class.
2.	The class represent the basis for the object.	Object represents the real world entity.
3.	Single class can create multiple objects.	Every object is created from one at only one class.
4.	Example - Class vehicle	Example - Alto, ZEN, SANTRO are the objects of class vehicle.

5.2.3 Methods and Message Passing

- An object provides some service. A message for an object is a request made for execution of **method** for obtaining the service.

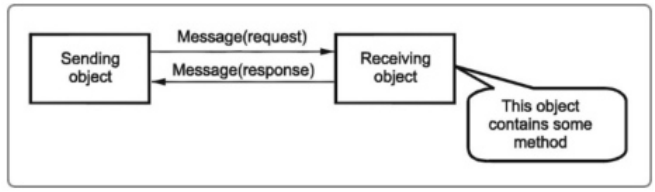


Fig. 5.2.1 Communication between two objects

- Another object containing the method, executes that method and then responds back by passing the message. Thus action is initiated in object oriented programming by the transmission of a message to the object responsible for the action.

A **message** is a request to an object to invoke one of its methods. A message therefore contains,

- The **name** of the method and
- The **arguments** of the method.

5.2.4 Inheritance

Inheritance *is a property by which the new classes are created using the old classes. In other words the new classes can be developed using some of the properties of old classes.*

- The old classes are referred as base classes and the new classes are referred as **derived classes**. That means the derived classes inherit the properties (data and functions) of base class. This allows to reuse the existing properties of base class in derived class.

For example

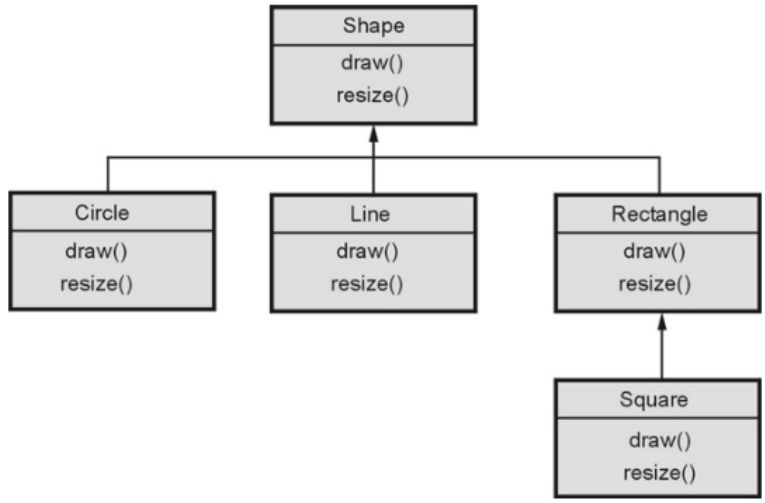


Fig. 5.2.2 Example of inheritance

- Here the shape is a base class from which the **circle, line** and **rectangle** are the derived classes. These classes inherit the functionality **draw()** and **resize()**. Similarly the rectangle is a base class for the derived class square. Along with the derived properties the derived class can have its own properties. For example, the class circle may have the function like **backgrcolor( )** for defining the background color.

5.2.5 Polymorphism

- Polymorphism means many structures.
- ***Polymorphism*** *is the ability to take more than one form and refers to an operation exhibiting different behavior in different instances (situations).*
- The behavior depends on the type of data used in the operation. It plays an important role in allowing objects with different internal structures to share the same external interface.
- Object oriented programs use polymorphism to carry out the same operation in a manner customized to the object. Without polymorphism, you would have to create separate module names for each method.
- **For example** the method **clean** is used to clean a **dish** object, one that cleans a **car** object, and one that cleans **vegetable** object.
- With polymorphism, you create a single "clean" method and apply it for different objects.

5.3 Creating Classes and Objects

- Python is an object oriented programming language. That means almost all the code is implemented using a special construct called classes.
- Programmers make use of classes to keep the data values and methods together.
- Object is simply a collection of data (variables) and methods (functions) that act on those data. And, class is a blueprint for the object.

Terminologies used in object oriented programming

1. **Class :** Class is a collection of data attributes and the methods that can access or manipulate the data attributes. For defining the class in Python the keyword **class** is used.
2. **Class variable :** This is a kind of variable which is shared by all instances of class. The class variable are defined within a class but outside the class method.
3. **Data member :** The class variable that holds data associated with a class and its objects.
4. **Instantiation :** The creation of instance of a class.
5. **Class method :** It is a special type of function which is defined inside the class definition.
6. **Object :** It is an instance of a class. An object comprises both data members and methods.
7. **Instantiation :** Creating a new object is called instantiation and the object is called **instance** of a class.

5.3.1 How to Create a Class and Object ?

The keyword **class** is used to define the class followed by the class name.

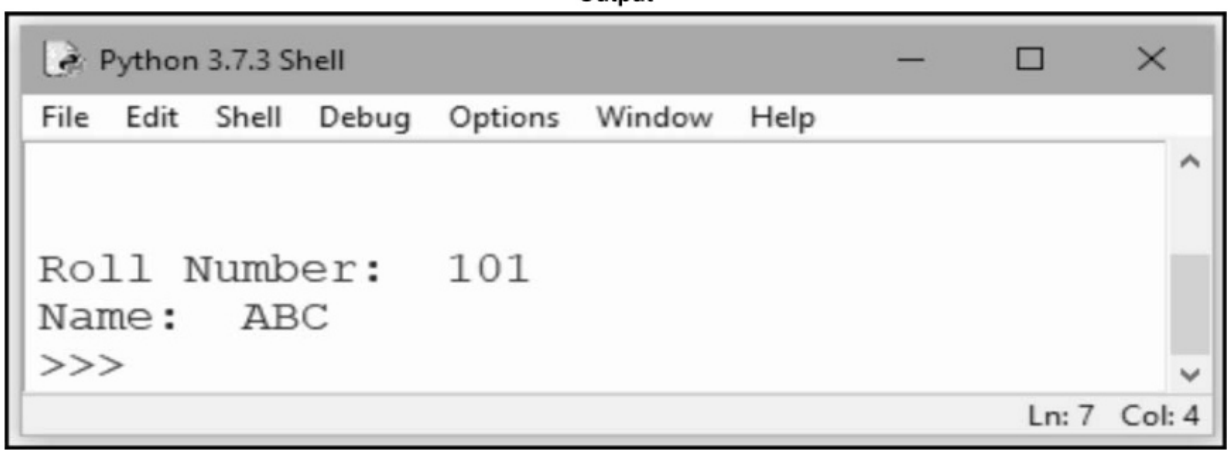
Syntax of class

```
Class class_name:
 Statement1
 Statement2
 …
Statementn
Syntax of Object
Object_name=class_name()
```

For example

```
Class Student:
 Roll=101
 Name="ABC"
obj=Student()
print("Roll Number: ",obj.Roll)
print("Name: ",obj.Name)
```

Output



5.3.2 Class Method and Self Object

- The class methods are the functions defined inside the class. These are actually the normal functions but belong to that class.
- The class method defines the first argument as **self**. This is always defined as the first argument. If there is no argument to the class method then only self will be the argument for that method. And if there are some arguments present in the class method, then always the first argument will be **self** and then remaining will be the list of arguments of that class method.
- The **self**-argument refers to the object itself.
- The class methods are accessible from outside the class using object. Hence these methods are also called as **instance methods**.

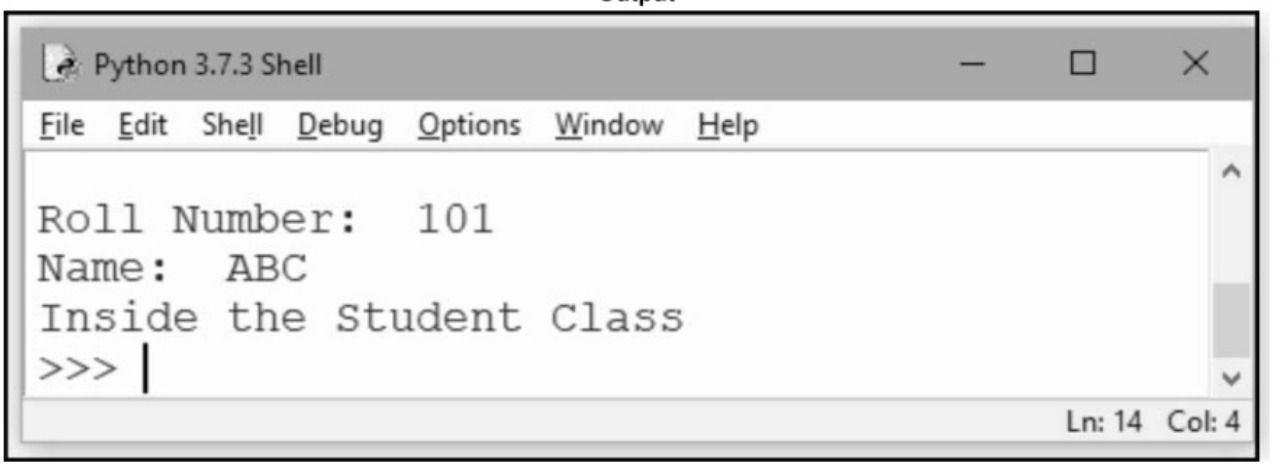
Programming example

In the following example we define the class method and demonstrate the use of **self** object

StudentDemo.py

```
class Student:
    Roll=101 #Data member
    Name='ABC' #Data member
    def show(self):
        print("Inside the Student Class")#definition of class method
obj=Student() #creating object of class
print("Roll Number: ",obj.Roll)#Accessing data member using object
print("Name: ",obj.Name)#Accessing data member using object
obj.show()#Accessing class method using object
```

Output



5.3.3 Public and Private Members

- Public variables are those variables that are defined in the class and can be accessed from within the class and from outside the class.
- All members in a Python class are public by default.

For example

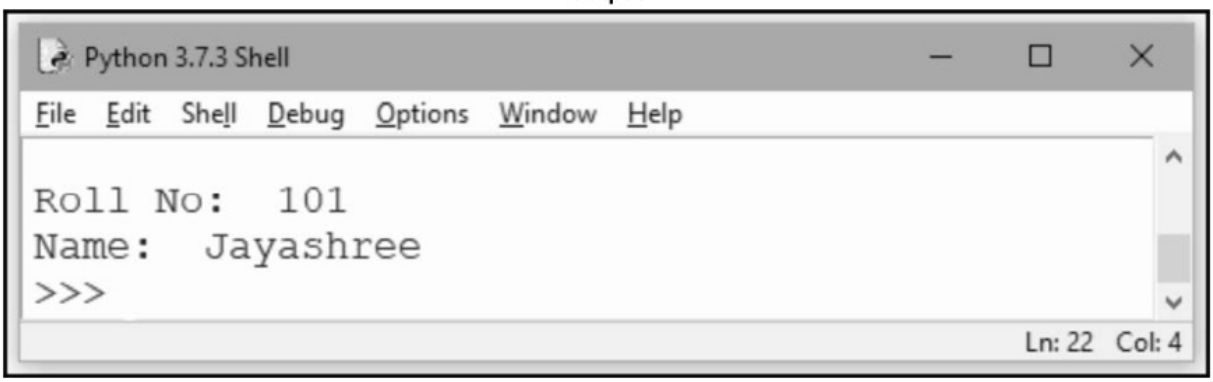
StudentDemo.py

```
class Student:
   def __init__(self,R,N):
      self.Roll=R
      self.Name=N
```

These are the public members of class **Student**

```
obj=Student(101,"Jayashree")
print("Roll No: ",obj.Roll) #Accessing public variable outside the class
print("Name: ",obj.Name)
```

Output



Program explanation : In above program,

Two variables - **Roll** and **Name** are the public variables and we are accessing them outside the class.

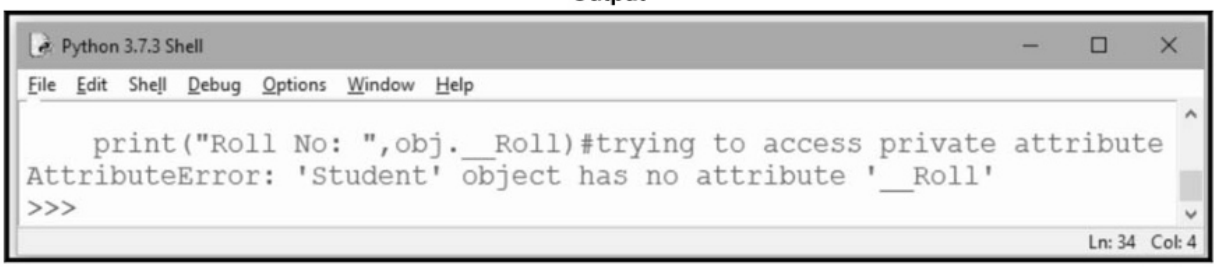
- Private variables are those variables that are defined in the class but can be accessible by the methods of that class only. The private variables are used with double underscore(__) prefix.

For example

StudentDemo.py

```
class Student:
   def __init__(self,R,N):
      self.__Roll=R #private variable 'Roll'
      self.__Name=N #private variable 'Name'

obj=Student(101,"Jayashree")
print("Roll No: ",obj.__Roll) #trying to access private attribute
                              #outside the class and that will cause error
```

Output



Program explanation : In above program,

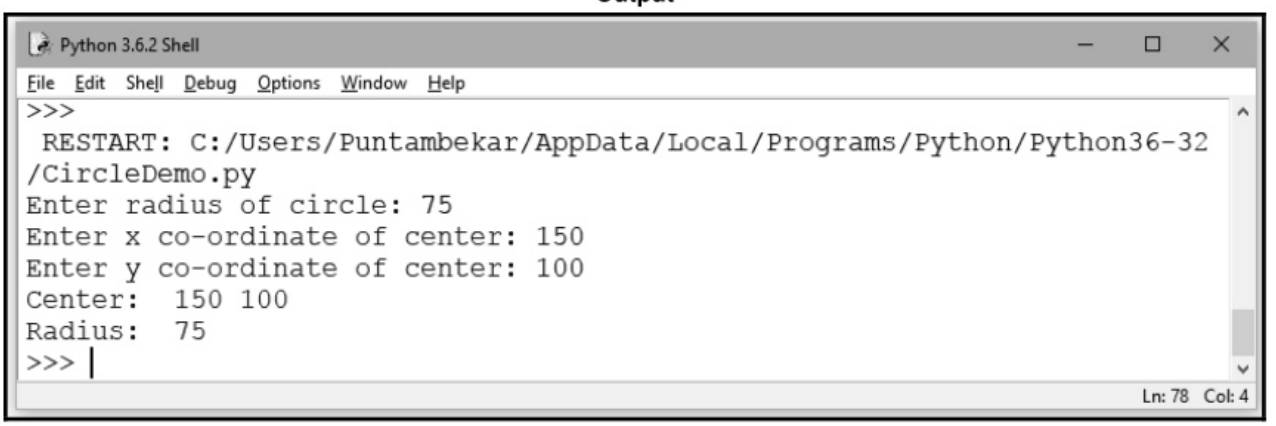
1) There are two private variables - Roll and Name. Note that to make these variables private they are written using the prefix as double underscore.
2) When we try to access these variables outside the class using the object of that class, it gives an Attribute error. Refer the screenshot for the output.

Example 5.3.1 : Write a definition for a class named Circle with attributes center and radius, where center is a Point object and radius is a number.
Instantiate a Circle object that represents a circle with its center at (150, 100) and radius 75.
Solution :

CircleDemo.py

```
class Point():
   x=0
   y=0
class Circle():
   radius=0

def display(pt,cir):
      print("Center: ",pt.x,pt.y)
      print("Radius: ",cir.radius)

radius=int(input("Enter radius of circle: "))
a=int(input("Enter x co-ordinate of center: "))
b=int(input("Enter y co-ordinate of center: "))
pt=Point()
pt.x=a
pt.y=b
cir=Circle()
cir.radius=radius
display(pt,cir)
```

Output



Review Question

1. *Explain how to create class and object in object oriented programming.*

5.4 Method Overloading and Overriding

5.4.1 Method Overloading

- Method overloading is an ability of a function to behave in different ways based on the number of parameters passed to it.
- In method overloading the same function name is used for different behavior. Following is an example program which shows method overloading concept.

```
OverloadDemo.py
class Color:
   def selectColor(self, name=None):
      if name is not None:
         print('My Favourite color is ' + name)
      else:
         print('I do not like any color')

# Create instance
obj = Color()
# Call the method
obj.selectColor()
# Call the method with a parameter
obj.selectColor('White')
```

Output

```
I do not like any color
My Favourite color is White
>>>
```

5.4.2 Method Overriding

- Method overriding is a mechanism in object oriented programming which allows to change the implementation function in child class which is defined in parent class.
- There are two essential criteria for method overriding :
  1. For performing method overriding, there must be inheritance.
  2. The function that is redefined in child class must have same number of arguments in child class.

Following is a Python program that illustrates the concept of method overriding.

```
class Parent: #Parent Class
   def __init__(self):
      self.age=45

   def display(self):
      print("Age: ",self.age)

class Child(Parent):  #Child Class
   def display(self):   #Note that the same method name but different implementation
      self.age=self.age-25
      print("Age: ",self.age)
```

```
p=Parent()
c=Child()
print("Parent's Age:")
p.display()
print("Child's Age:")
c.display()
```

Output

```
Parent's Age :
Age :  45
Child's Age:
Age :  20
>>>
```

Difference between Method Overloading and Method Overriding

Method Overloading	Method Overriding
Different number of parameters can be passed to the function.	The number of parameters that are passed to the function are the same.
The overloaded functions may have different return types.	In the method overriding all the methods will have same return types.
Method overloading is performed within a class.	Method overriding is always performed between two classes that have **inheritance relationship.**

Review Questions

1. *What is method overloading ? Write a Python program that implements the concept of method overloading.*
2. *What is method overriding ? Write a Python program that implements the concept of method overriding.*
3. *What is the difference between method overloading and method overriding.?*

5.5 Data Hiding

- **Definition :** Data hiding is a mechanism which does not allow to access the attributes outside the class.
- Data hiding is an important property of object oriented programming.
- In Python the data hiding capability can be achieved by making the members of the class **private** or **protected.**
- In Python, the **double underscore** is used before the name to make the attribute **private.**
- Similarly the **single underscore** is used before the name to make the attribute **protected.**

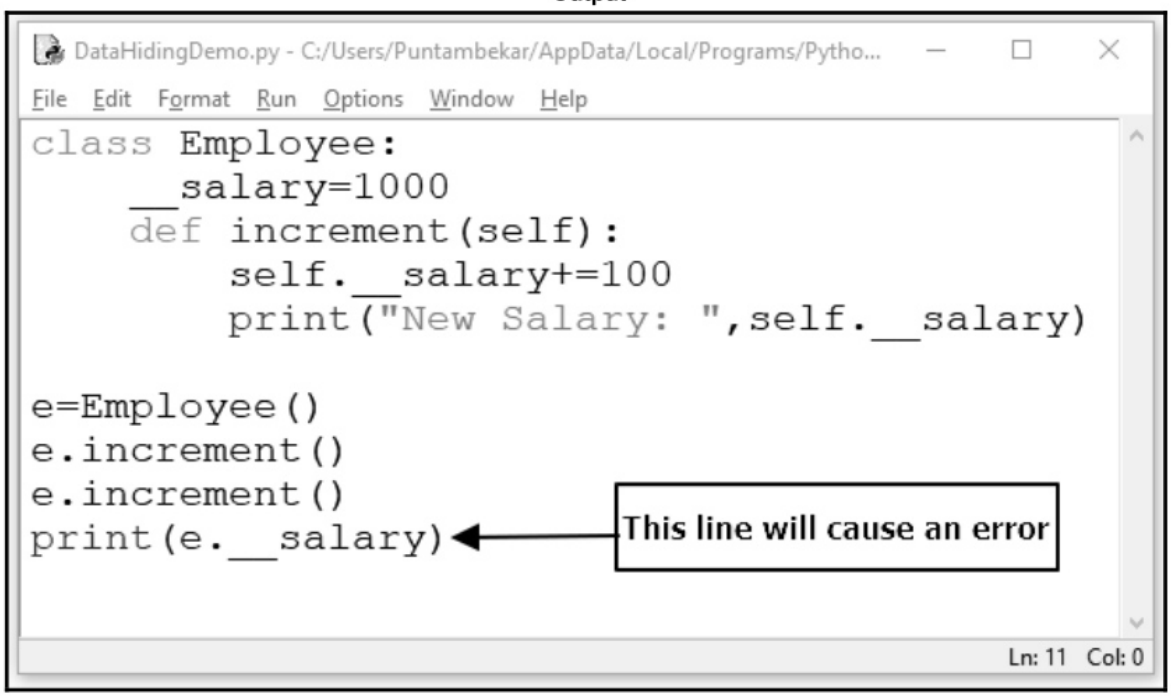
Python Program on Data Hiding

```
class Employee:
   __salary=1000
   def increment(self):
      self.__salary+=100
      print("New Salary: ",self.__salary)
```

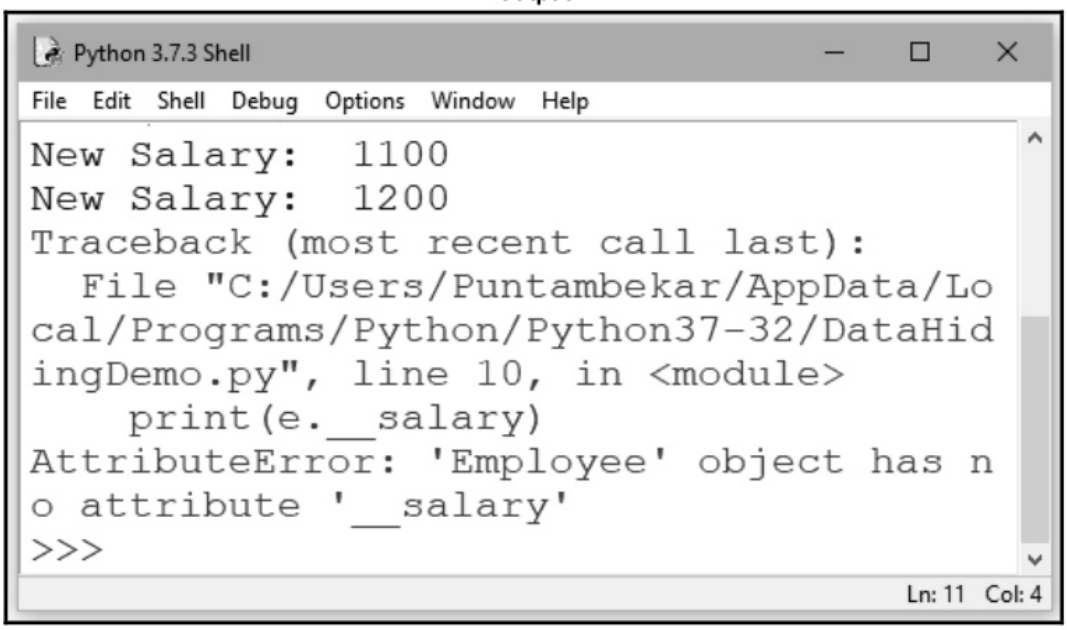
```
e=Employee()
e.increment()
e.increment()
print(e.__salary)
```

This can be represented by following screenshot.

Output



Output



Code Explanation : In above Python Program,

(1) We have created on class named **Employee.** This class contains a variable named **salary.** This variable is a private variable.

(2) The **salary** is incremented inside the method named **increment.** This method belongs to the same class **Employee.**

(3) Outside the class, we have created an instance or an object of the class **Employee** by the statement

```
e=Employee()
```

(4) Using this instance or object e we can invoke or call the method **increment** with the help of dot operator. In our program we have invoked **increment** function twice. Hence salary will become 1100 and 1200. Hence is the output.

(5) But as soon as we try to access the private variable **salary** directly with the help of object **e** outside the class **Employee** then the error occurs. That means we can not access the private variable directly outside the class. This achieves data hiding property.

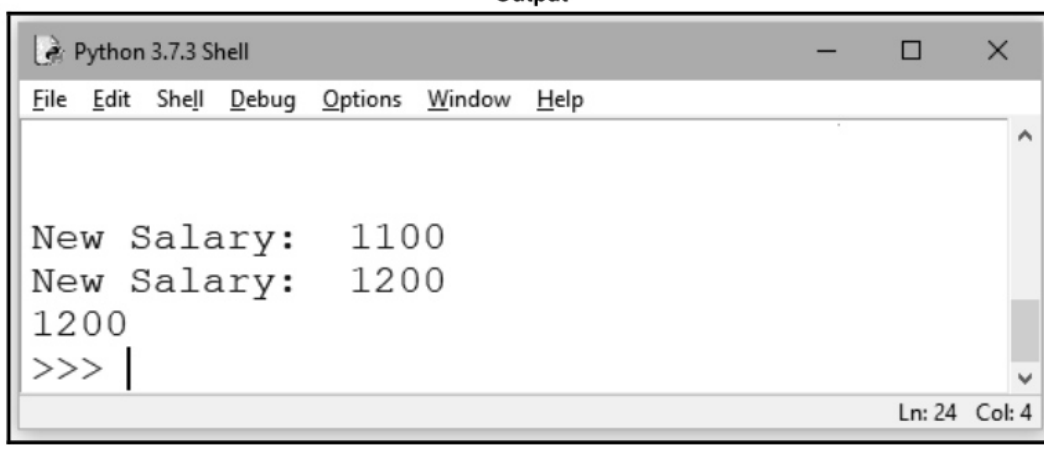
However, Python allows to access the private member of the class outside the class using a special construct

```
Object._classname__attributeName
```

Following program illustrates this. Note that the following program is just similar to above program. The only change made in above program is shown by boldfaced line -

```
class Employee:
   __salary=1000
   def increment(self):
      self.__salary+=100
      print("New Salary: ",self.__salary)

e=Employee()
e.increment()
e.increment()
print(e._Employee__salary)
```

Output



This shows that accessing the private attributes in Python is not easy but nothing in Python is truly private.

Review Question

1. *How to achieve data hiding concept in a Python program. Explain it with suitable example.*

5.6 Data Abstraction

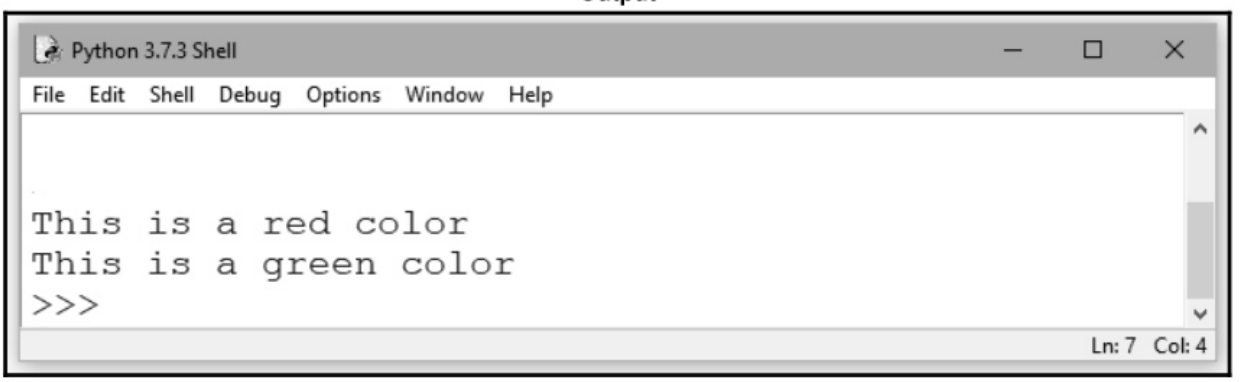
Definition : Data abstraction is a mechanism in which the abstract class is created. The abstract class is a class in which abstract method is defined. An abstract method is a method which does not have any implementation or definition. However, this abstract method is defined inside the child classes.

Following is an example program in which abstract class **Color** is created.

```
from abc import ABC,abstractmethod
class Color:
    @abstractmethod
    def showColor(self):
        none

class Red(Color):
    def showColor(self):
        print("This is a red color")

class Green(Color):
    def showColor(self):
        print("This is a green color")

def main():
    r=Red()
    r.showColor()

    b=Green()
    b.showColor()

if __name__=="__main__":
    main()
```

This is a abstract class with abstract method **showColor**

Output



Code Explanation : In above Python program,

(1) In Python, it is not possible to use abstract classes by default. For supporting abstraction, python comes up with a module which provides the base for defining **Abstract Base classes(ABC)** and that module name is **ABC.** The **ABC** works by marking methods of the base class as abstract and then registering concrete classes as implementations of the abstract base. A method becomes an abstract by decorated it with a keyword @abstractmethod.

Hence at the beginning of the program we have written,

```
from abc import ABC,abstractmethod
```

(2) Now we have defined an abstract class named **Color.** This class contains an abstract method named **showColor.** By writing **none** we are having no implementation code for this method.

(3) Then we have written two child classes namely **Red** and **Green** in which the method **showColor** is redefined.

(4) Inside the main() function the objects for **Red** and **Blue** classes are created. Using the corresponding objects the **showColor** method of respective classes is invoked.

(5) Every module in Python has a special attribute called **__name__.** The value of **__name__** attribute is set to '**__main__**' when module run as main program.

Review Questions

1 *What is data abstraction ?*

2. *How to use the abstract method in Python program ? Give suitable example.*

5.7 Inheritance and Composition Classes

Inheritance

Definition : Inheritance is a relationship which is similar to a family tree. In this relationship **type-of** relations is defined. For example – A Person can be a Student or a Teacher. It is represented as,

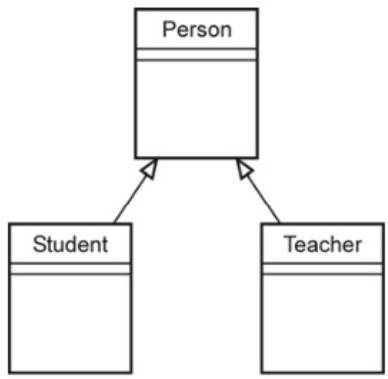


Fig. 5.7.1 Representing Inheritance

Types of Inheritance

There are two form of inheritance – single inheritance and multiple inheritance.

(1) Single Inheritance : In single inheritance, there is one base class and one derived class. Following figure represents single inheritance.

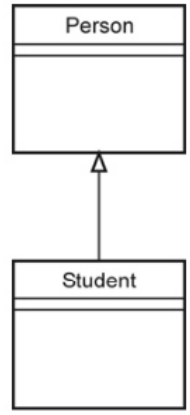
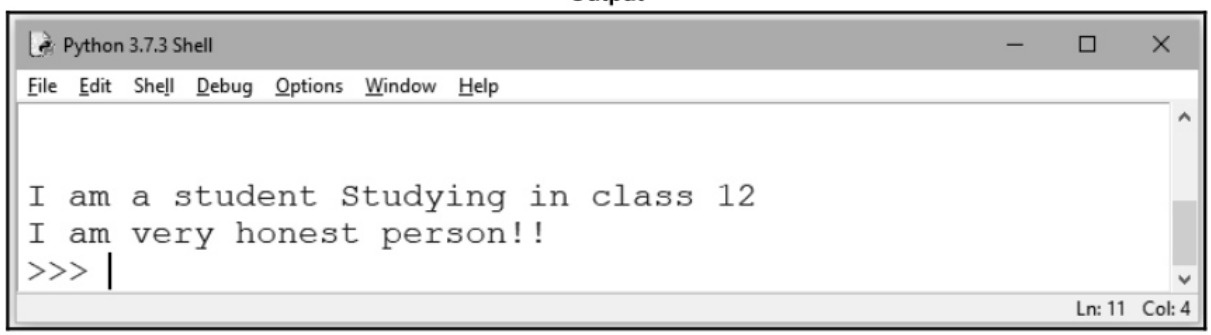


Fig. 5.7.2 Single Inheritance

Example program to implement Single level inheritance

```
class Person:
   def display(self):
      print("I am very honest person!!")
#child class Dog inherits the base class Animal
class Student(Person):
   def show(self):
      print("I am a student Studying in class 12")
s = Student()
s.show()
s.display()
```

Output



(2) Multiple Inheritance : In multiple inheritance, more than one child classes can be derived from single base class. Following figure represents the multiple inheritance –

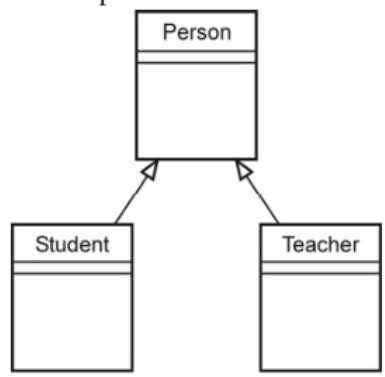


Fig. 5.7.3 Multiple inheritance

Example Program to Implement Inheritance

```
class Person : #Base class
    def __init__(self, first, last) :
        self.firstname = first
        self.lastname = last

    def Name(self):
        return self.firstname + " " + self.lastname


class Student(Person): #Derived Class
    def __init__(self, first, last, marks) :
        Person.__init__(self,first, last)
        self.studMarks = marks

    def GetStudent(self) :
        return self.Name() + ", " + self.studMarks

class Teacher(Person): #Derived Class
    def __init__(self,first,last,subject):
        Person.__init__(self,first,last)
        self.subject = subject

    def GetTeacher(self):
        return self.Name() + ", " + self.subject

print("The student details are ...\n")
s = Student("AAA", "BBB", "100")
print("Student Name and his marks: ",s.GetStudent())
print("-----------------------------------------------------")
print("The Teacher details are ...\n")
t = Teacher("PPP", "TTT", "Mathematics")
print("Teacher Name and subject taught: ",t.GetTeacher())
```

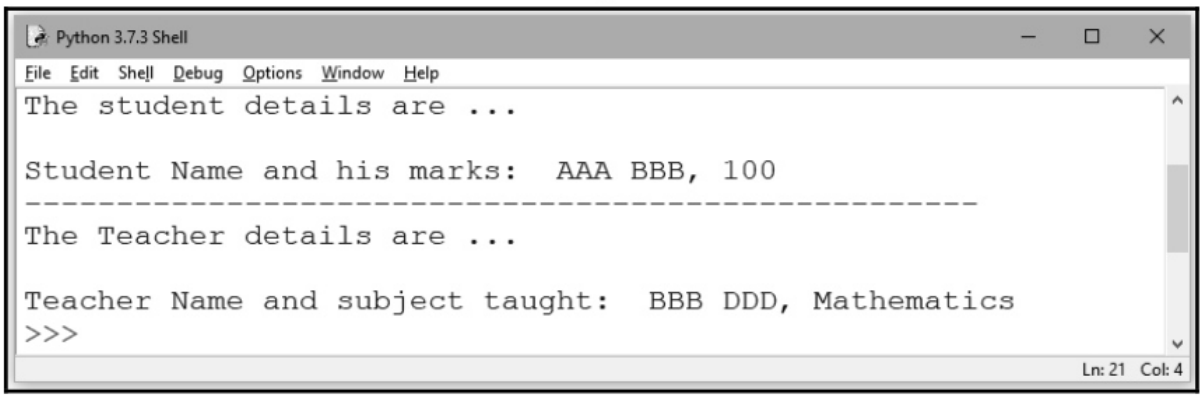
Base class or parent class

First derived class or child class

Second derived class or child class

Calling the child classes and displaying the contents of these classes.

Output



Composition

Composition is a relationship in which one object is defined as a part of another object.

For example - Car has engine or Student has marks

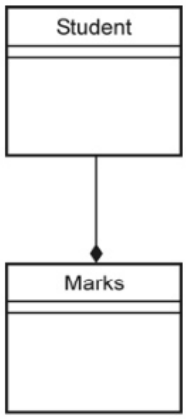
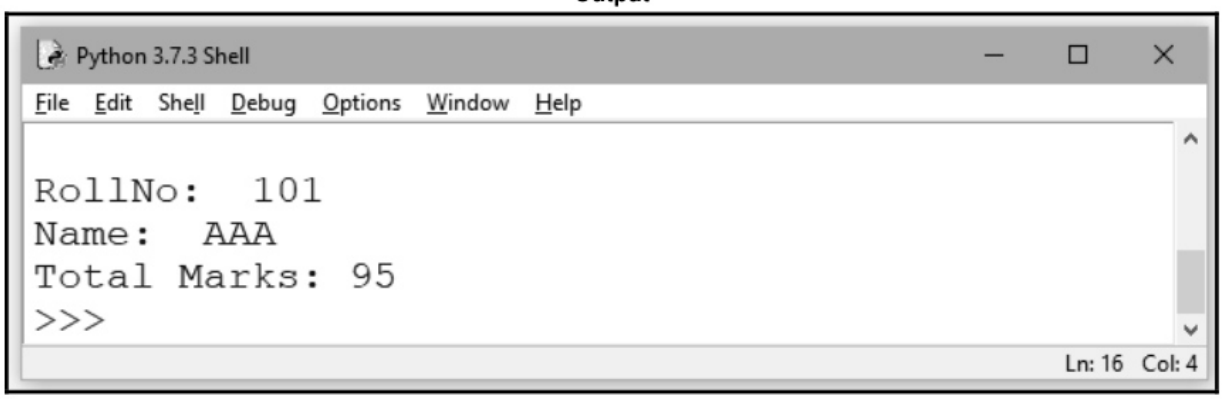


Fig. 5.7.4 Representing Composition

Example Program to implement Compostion

```
class Marks:
    def __init__(self,theory,practical):
        self.theory=theory
        self.practical=practical

    def calculate_marks(self):
        return (self.theory+self.practical)

class Student:
    def __init__(self,rollNo,name,theory,practical):
        self.rollNo=rollNo
        self.name=name
        self.theory=theory
        self.practical=practical
        self.obj_Marks=Marks(self.theory,self.practical) #composition

    def total_marks(self):
        return self.obj_Marks.calculate_marks()

s=Student(101,'AAA',70,25)
print("RollNo: ",s.rollNo)
print("Name: ",s.name)
print("Total Marks:",s.total_marks())
```

Output



In composition one of the classes is composed of one or more instance of other classes.

In above code there are two classes Student and Marks.

The Student class is composed of an instance of Marks class. This is done by using following code

```
self.obj_Marks=Marks(self.theory,self.practical)
```

Here, the **Student** class has delegated the responsibility of calculating total marks to the **Marks** class. When one class delegates the responsibilities to other class,it is called **composition.**

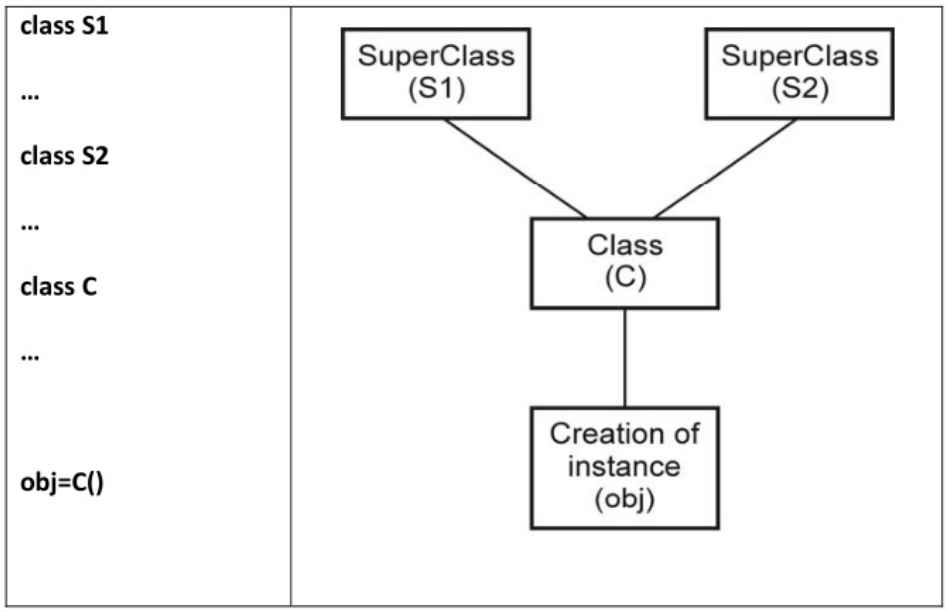
Review Questions

1. *What is inheritance ? Explain it with suitable example.*
2. *What is composition ? Write a Python program to implement the composition relation.*

5.8 Customization via Inheritance

- Inheritance uses attribute definition tree.
- The object.attr searches in upward direction of the tree for first occurrence of the word "attr".
- Lower definitions of the methods in tree redefine or override the upper definitions of the methods in the tree structure of inheritance.

Construction of Attribute Tree



5.8.1 Specializing Inherited Methods

During execution of inheritance tree, the subclass name is located prior to the superclass name.

The direct superclass method calls

```
Class.method(self,...)
```

Example

```
class A:
  def display(self):
    print("Executing in Super Class")
```

```
class B(A):
   def display(self):
      print("Beginning of sub Class")
      A.display(self)
      print("Ending of sub Class")

obj=A()
A.display()
obj=B()
obj.display()
```

Output

Executing in Super Class

Beginning of sub Class

Ending of sub Class

Code Explanation :

Note that when we create an instance of a super class, and call the method using this instance, then the method defined in super class is executing directly, but when we create an instance of sub class, and calling a method using this instance then first the method of super class gets executed.

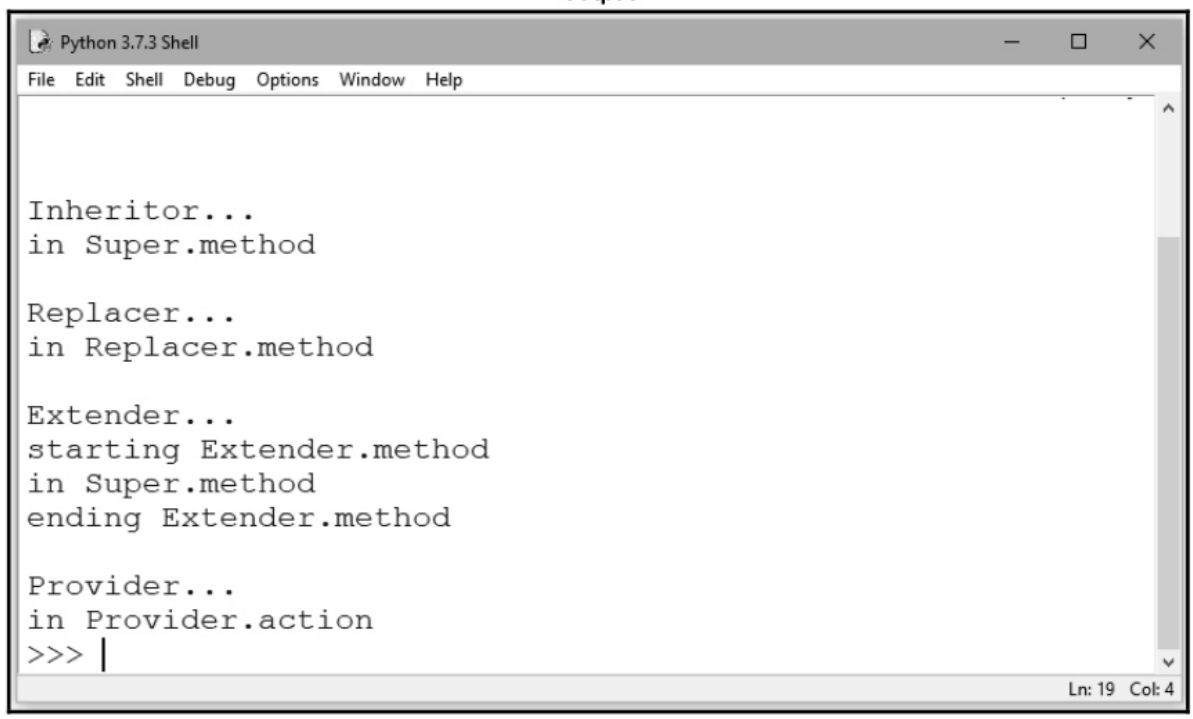
Use of Specialized Method

The execution of specializing inherited methods is represented by following example -

```
class Super:
   def method(self):
      print ('in Super.method')    # default
   def delegate(self):
      self.action()                # expected

class Inheritor(Super):
   pass

class Replacer(Super):
   def method(self):
      print ('in Replacer.method')

class Extender(Super):
   def method(self):
      print ('starting Extender.method')
      Super.method(self)
      print ('ending Extender.method')

class Provider(Super):
   def action(self):
```

```
        print ('in Provider.action')

if __name__ == '__main__':
    for obj in (Inheritor, Replacer, Extender):
        print ('\n' + obj.__name__ + '...')
        obj().method()
    print ('\nProvider...')
    Provider().delegate()
```

Output



Review Question

1. *Write a short note on – Customization via inheritance.*

❑❑❑

6 File I/O Handling and Exception Handling

6.1 I/O Operations

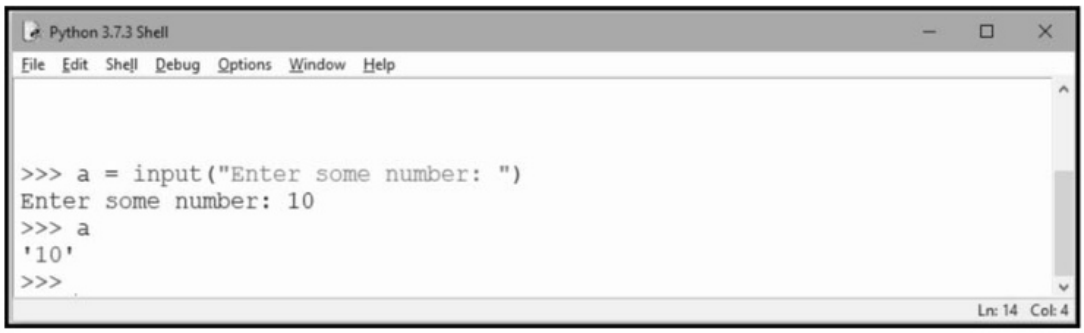
6.1.1 Reading Keyboard Input

- In python it is possible to input the data using keyboard.
- For that purpose, the function **input()** is used.
- **Syntax**

input([prompt])

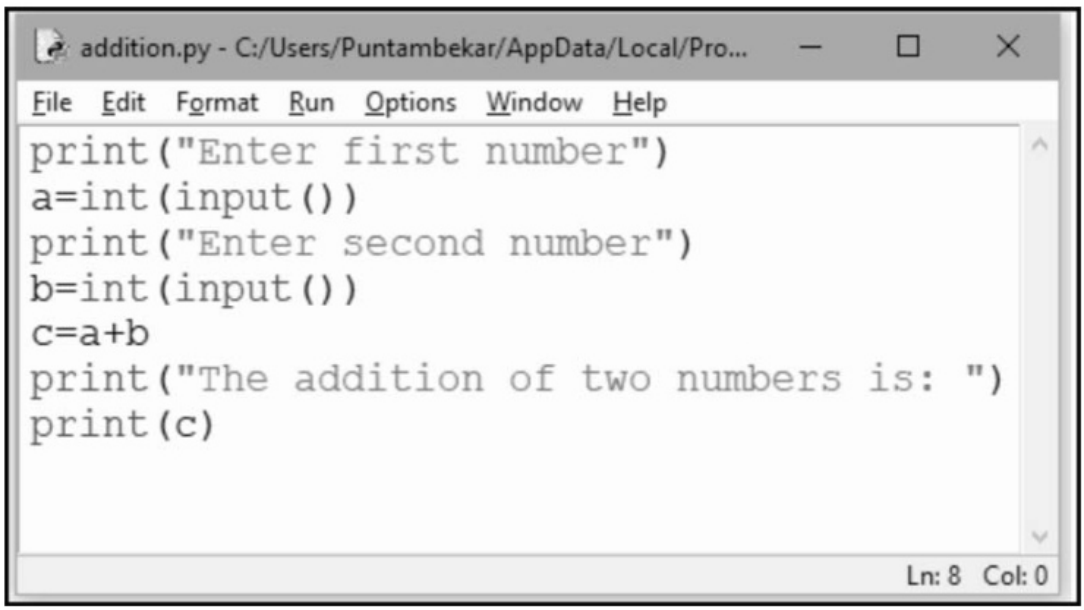
where prompt is the string we wish to display on the screen. It is optional.

Example 6.1.1 : *In the following screenshot, the input method is used to get the data*



Example 6.1.2 : *Write a python program to perform addition of two numbers. Accept the two numbers using keyboard*

Solution : addition.py



Output

For getting the output click on **Run-> Run Module** or press F5 key, following shell window will appear -

Program Explanation :

- In above program, we have used **input()** function to get the input through keyboard. But this input will be accepted in the form of string.
- For performing addition of two numbers we need numerical values and not the strings. Hence we use **int()** function to which the **input()** function is passed as parameter. Due to which whatever we accept through keyboard will be converted to integer.
- Finally the addition of two numbers as a result will be displayed.
- The above program is run using F5 and on the shell window the messages for entering first and second numbers will be displayed so that user can enter the numbers.

Example 6.1.3 : *Write a Python program to find the square root of a given number*
Solution:

SqrtDemo.py

```
print("Enter the number:")
num=float(input())
result=num**0.5
print("The sqaure root of ",num," is ",result)
```

Output

```
Enter the number:
25
The sqaure root of  25.0  is  5.0
>>>
```

Example 6.1.4 : *Write a program in Python to obtain principle amount, rate of interest and time from user and compute simple interest.*

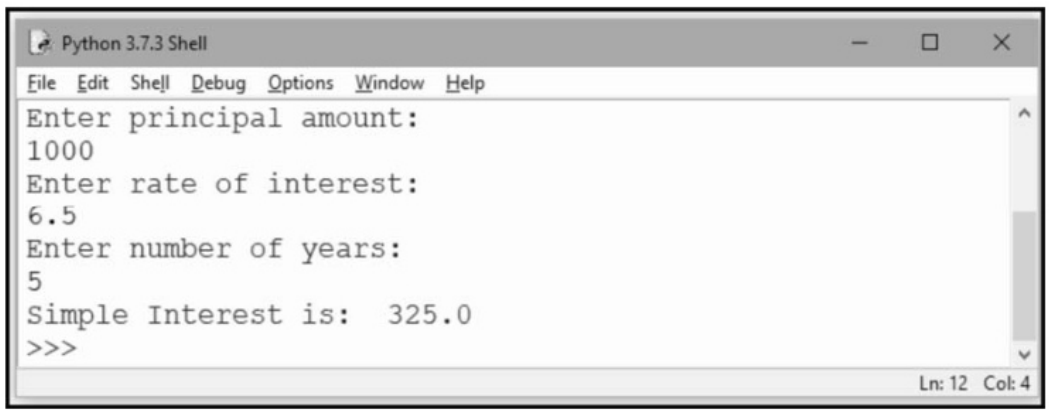
Solution :

Interest.py

```
print("Enter principal amount: ")
p=float(input())
print("Enter rate of interest: ")
r=float(input())
print("Enter number of years: ")
n=float(input())
I=(p*n*r)/100
print("Simple Interest is: ",I)    # Output will be displayed on console
```

Here we are reading the values through keyboard. Note we are reading the values as float

Output



Example 6.1.5 : *To calculate salary of employee given his basic pay(take as input from user). Calculate gross salary of employee. Let HRA be 10 % of basic pay and TA be 5 % of the basic pay. Let employee pay professional tax as 2 % of total salary. Calculate net salary payable after deductions.*

Solution :

```
print("Enter basic salary: ")
basic=float(input())
HRA=(basic*10)/100
TA=(basic *5)/100
gross_salary=basic+HRA+TA
print("Gross Salary is: ",gross_salary)
PTax=(gross_salary*2)/100
net_salary=gross_salary-PTax
print("Net Salary is:",net_salary)
```

Output

```
Enter basic salary:
10000
Gross Salary is:  11500.0
Net Salary is: 11270.0
>>>
```

Example 6.1.6 : *To accept an object mass in kilogram and velocity in meters per second and display its momentum. Momentum is calculated as* $e = mc^2$ *where m is the mass of the object and c is its velocity.*

Solution :

```
print("Enter mass of the object(kg)")
m=float(input())
print("Enter velocity of the object(m/s)")
c=float(input())
```

```
e=m*(c**2)
print("Momentum is: ",e)
```

Output

```
Enter mass of the object(kg)
10
Enter velocity of the object(m/s)
2.5
Momentum is:  62.5
>>>
```

Example 6.1.7 : *To accept numbers from user and print digits of number in reverse order.*
Solution :

```
print("Enter some number")
n=int(input())
reverse=0
while (n >0):
   reminder = n %10
   reverse = (reverse *10) + reminder
   n = n //10
print("The reversed number is: ",reverse)
```

Output

```
Enter some number
1234
The reversed number is:  4321
>>>
```

Example 6.1.8 : *To input binary number from user and convert it to decimal number*
Solution :

```
print("Enter a binary number: ")
binary = list(input())
value = 0

for i in range(len(binary)):
        digit = binary.pop()#reading each digit of binary number from LSB
        if digit == '1':#if it is 1
                value = value + pow(2, i) #then get 2^i and add it to value
print("The decimal value of the number is", value)
```

Output

```
Enter a binary number:
1100
The decimal value of the number is 12
>>>
```

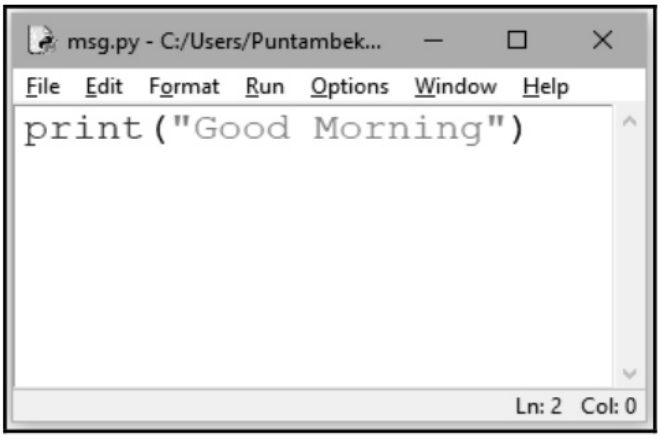
6.1.2 Printing to Screen

Using the print function we can print the information to the screen. Following are examples that illustrate how to print to the screen

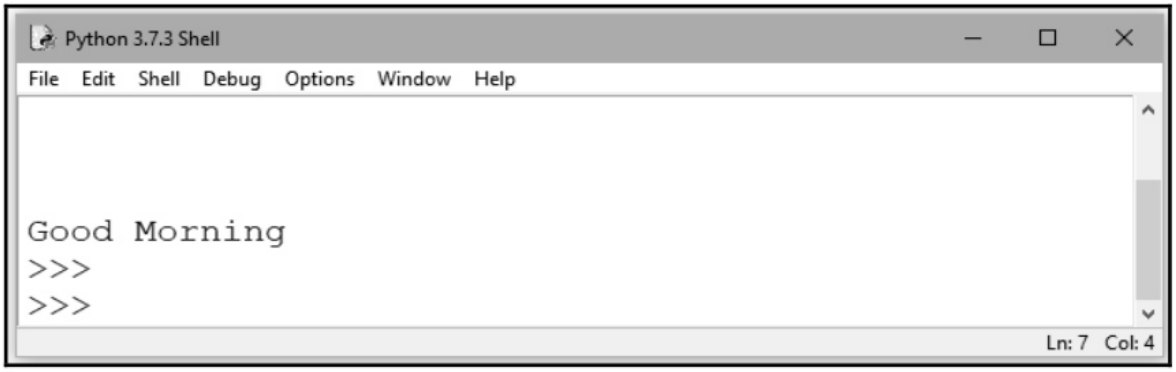
Printing simple text message

Example 6.1.9 : *Display the 'Good Morning' message to screen*
Solution:

Method 1 : You can save the following code in a separate file. I have named the file as msg.py
msg.py

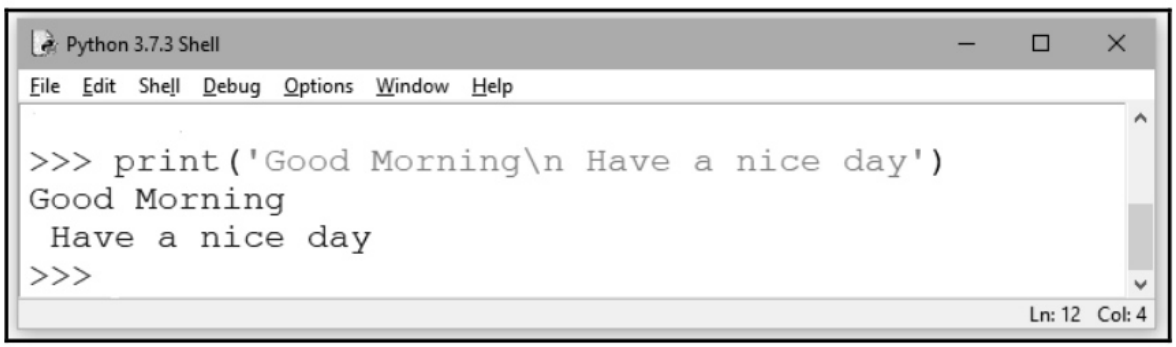


And then press F5 to run the above code – The output will be

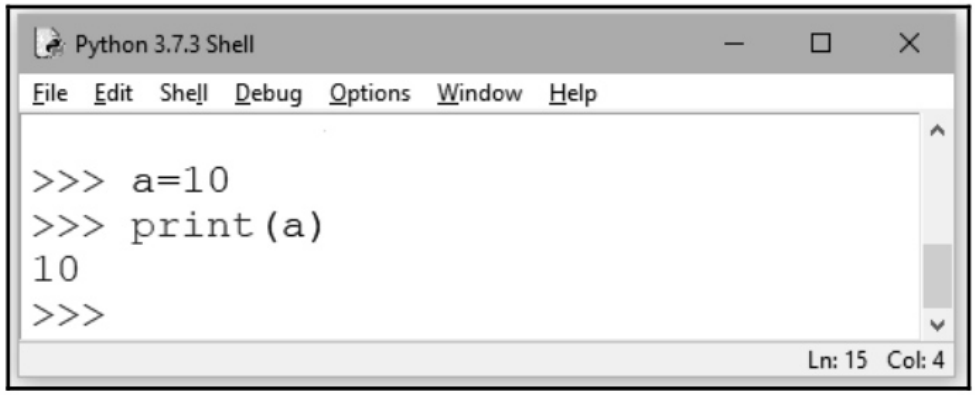


Method 2 : Open the Python Shell window and type the print command. It is as shown below -

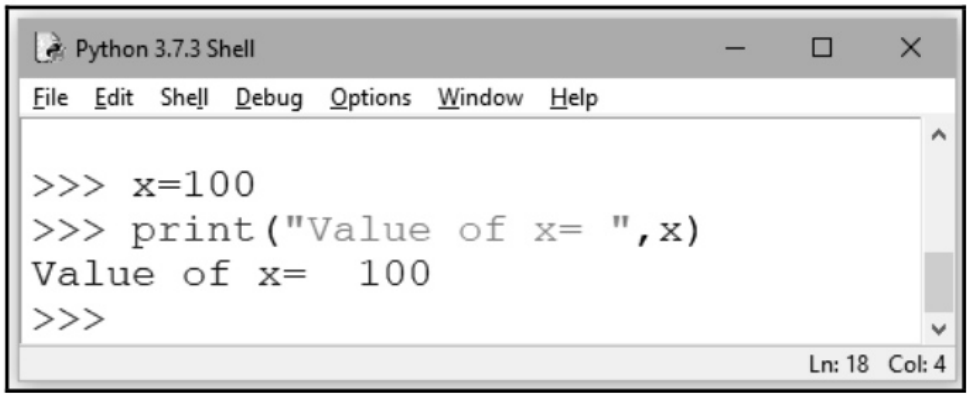
Printing Simple text message with new line



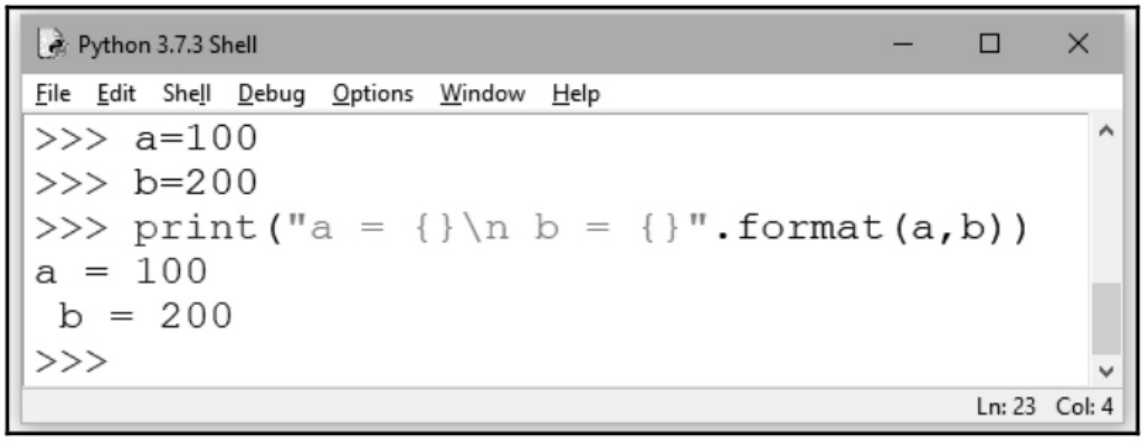
Printing Variables



We can display the value of variable along with the message



Printing variables using format



Review Questions

1. *Write a Python program to read some input using keyboard*
2. *What are different ways of printing data to screen ? Explain it with illustrative examples.*

6.2 File Handling

Definition : File is a named location on the disk to store information.

- File is basically used to store the information permanently on secondary memory.
- Due to this, even if the computer is switched off, the data remains permanently on secondary memory (hard disk). Hence it is called **persistent data structure.**

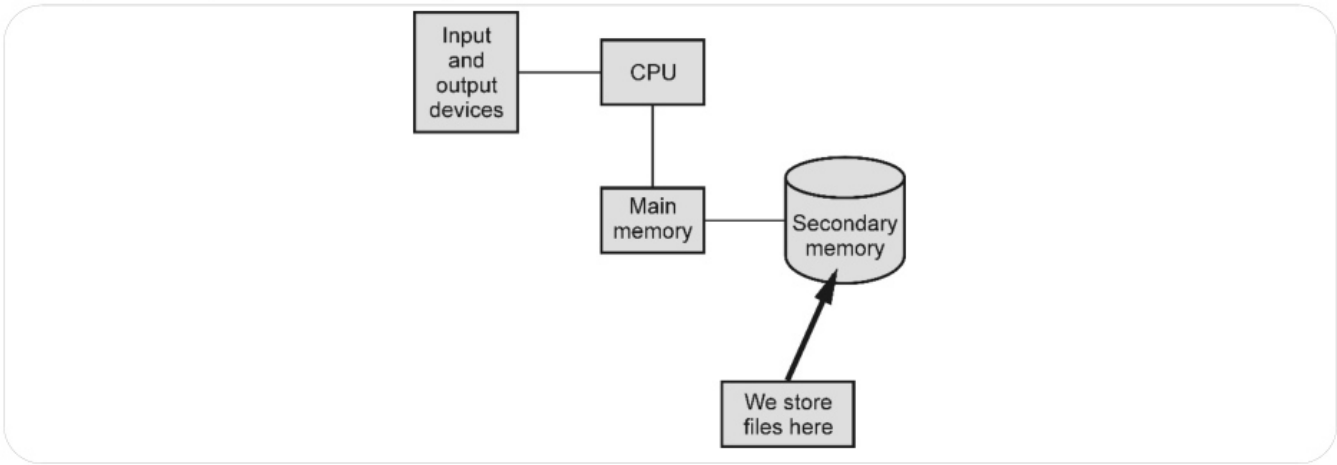


Fig. 6.2.1 File as persistent storage

- Every file is located by its path. This path begins at the root folder. In windows it can C:\, D:\ or E;\ etc.
- The file path is also called as **pathname.**

  For example c:\MyPythonPrograms\displayStrings.py is a pathname. In this path name C:\ is a root folder in which the sub-folder **MyPythonProgram** has a file called **displayStrings.py**
- The character \ used in the pathname is called **delimiter**.
- **Concept of Relative and Absolute Path :**
  - The absolute path is a complete path from the root directory. For example – the path **C: \MyPythonPrograms\Strings\displayStrings.py** is an absolute path using which the file can be located.
  - The relative path is a path towards the file from the current working directory. For example - **\Strings\displayStrings.py** can be a relative path if you are currently in the working directory C:\MyPythonPrograms

6.2.1 Opening a File in Different Mode

Opening a file

- In python there is a built in function open() to open a file.

Syntax

```
File_object=open(file_name,mode)
```

Where File_object is used as a handle to the file.

Example

```
F=open("test.txt")
```

Here file named **test.txt** is opened and the file object is in variable **F.**

We can open the file in text mode or in binary mode. There are various modes of a file in which it is opened. These are enlisted in the following table.

Mode	Purpose
'r'	Open file for reading
'w'	Open file for writing. If the file is not created, then create new file and then write. If file is already existing then truncate it.
'x'	Open a file for creation only. If the file already exists, the operation fails.
'a'	Open the file for appending mode. Append mode is a mode in which the data is inserted at the end of existing text. A new file is created if it does not exist.
't'	Opens the file in text mode
'b'	Opens the file in binary mode
'+'	Opens a file for updation i.e. reading and writing.

For example :

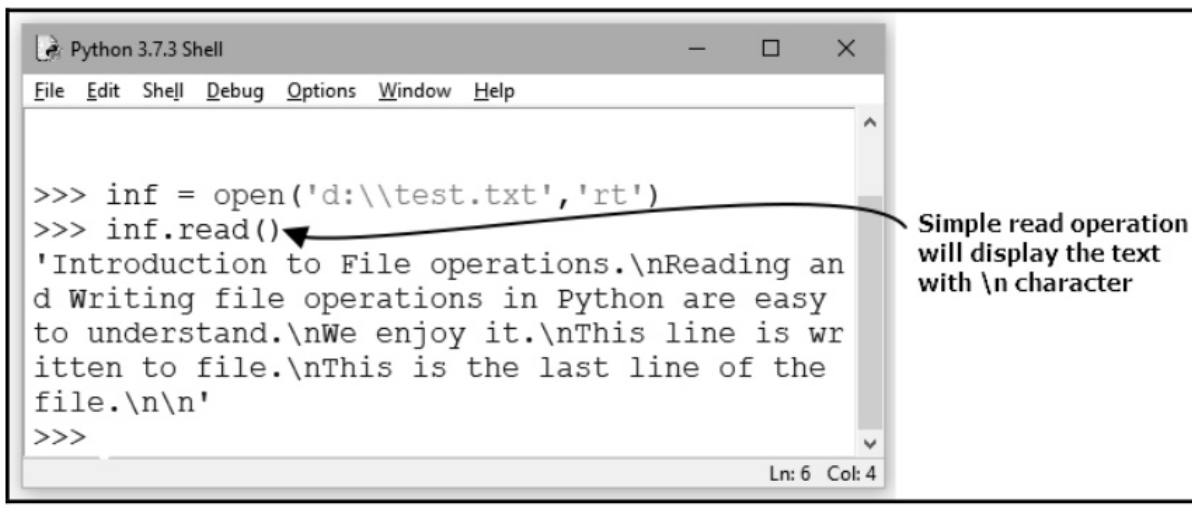
```
fo=open("test.txt",w) #opens a file in write mode
fo=open("text.txt",rt) #Open a file for reading in text mode
```

6.2.2 Accessing File Contents using Standard Library Functions

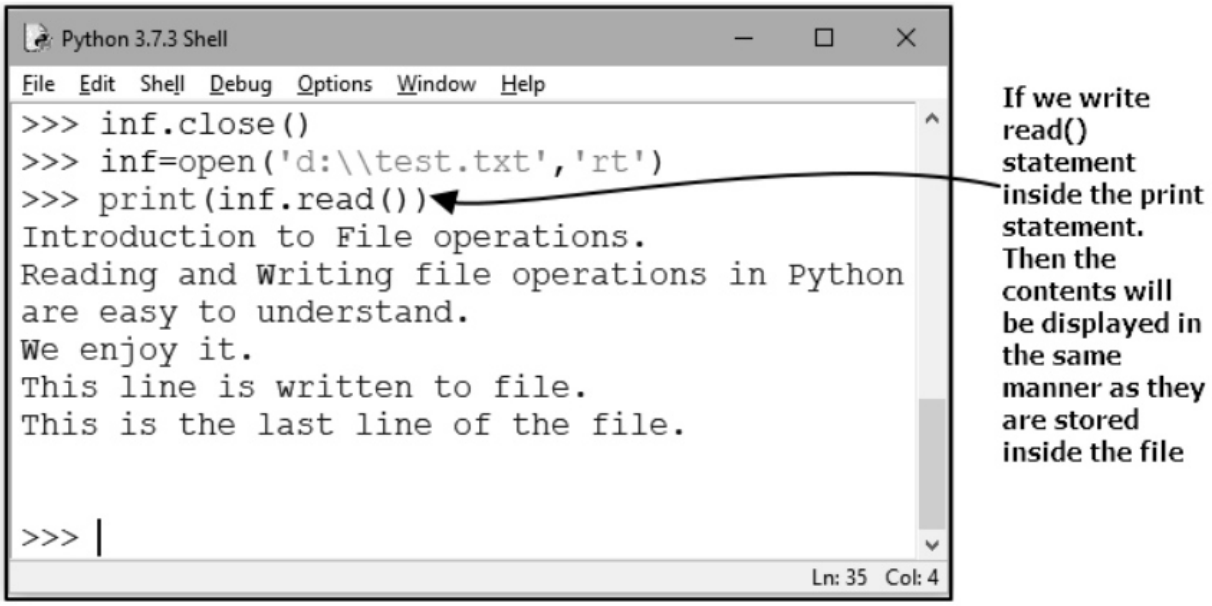
6.2.2.1 Reading Files

For reading the file , we must open the file in r mode and we must specify the correct file name which is to be read

The read() method is used for reading the file. For example –



Let us close this file and reopen it. Then call read statement inside the print statement, illustrated as follows –



Example 6.2.1 : *Write a Python program to read the contents of the file named 'test.txt*
Solution :

ReadFile.py

```
inf = open('D:\\test.txt','rt')
print(inf.read())
inf.close()
```

Output

```
Introduction to File operations.
Reading and Writing file operations in Python are easy to understand.
We enjoy it.
This line is written to file.
This is the last line of the file.
>>>
```

Note that a blank line is returned when the file reaches the end of the file.

There some other useful methods for reading the contents of the file. Let us discuss them with the help of necessary illustrations

The readline() Method

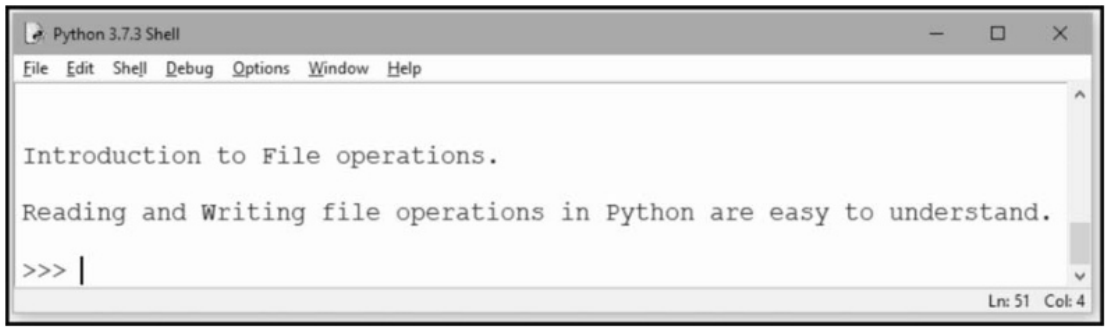
The readline() method allows us to read a single line from the file. When file reaches to the end, it returns an empty string.

Example 6.2.2 : *Write a Python program to read and display first two lines of the text file.*
Solution :

ReadLineDemo.py

```
inf = open('D:\\test.txt','rt')
print(inf.readline())
```

```
print(inf.readline())
inf.close()
```



Program Explanation : In above program,

1) The file is opened using **open** statement.
2) The we call **readline()** statements inside two subsequent print statement. After reading from the file using the **readline()** method, the control automatically passes to the next line. Hence we call **readline()** inside the **print** statement again.
3) Finally we must not forget to close the file using **close()** method.

The readLines() Method

The readLines() method is used to print all the lines in the program. Following program illustrates it –

ReadLinesDemo.py

```
inf = open('D:\\test.txt','rt')
print(inf.readlines())
inf.close()
```

Output

```
Introduction to File operations.

Reading and Writing file operations in Python are easy to understand.

We enjoy it.

This line is written to file.

This is the last line of the file.

>>>
```

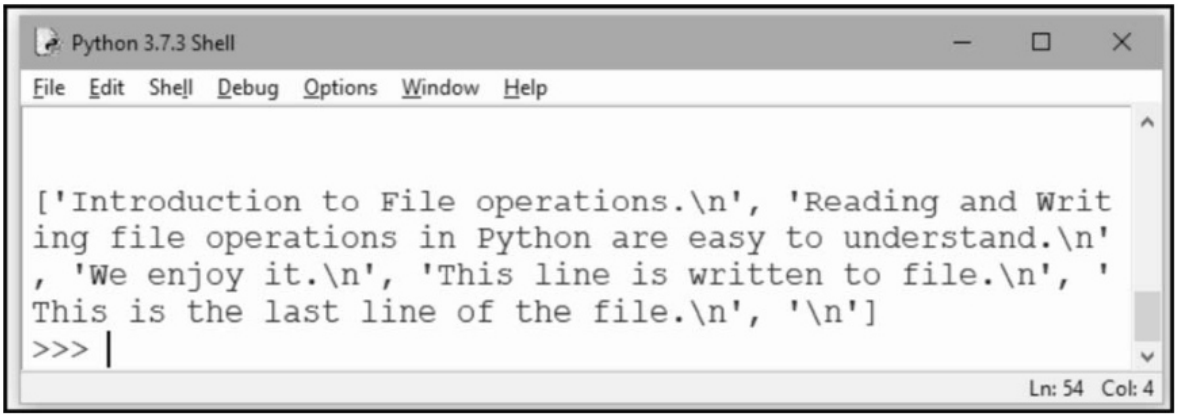
The list() Method

The list method is also used to display the contents of the file as a list. The program is as follows –

ListDemo.py

```
inf = open('D:\\test.txt','rt')
print(list(inf))
inf.close()
```

Output



Note that we passed the file object as an argument to the list method.

Displaying File using Loop

This is the most commonly used method of reading the file. In this method the contents of the file are read line by line using for loop.

Example 6.2.3 : *Write a program to display the contents of the file using for loop.*
Solution :

DisplayFile.py

```
inf = open('D:\\test.txt','rt')
for line in inf:
   print(line)
inf.close()
```

Output

```
Introduction to File operations.

Reading and Writing file operations in Python are easy to understand.

We enjoy it.

This line is written to file.

This is the last line of the file.

>>>
```

Opening a file using with

We can open the file using keyword **with**. The advantage of this is that the file gets closed properly after the read or write operations

OpenWithDemo.py

```
with open('D:\\test.txt','rt') as inf:
   for line in inf:
      print(line)
inf.close()
```

Output

```
Introduction to File operations.
Reading and Writing file operations in Python are easy to understand.
We enjoy it.
This line is written to file.
This is the last line of the file.
>>>
```

Example 6.2.4 : *Write a Python program to find the line that starts with the word "This" from the following text which is stored in a file*

Test.txt

This is a python program

python is superb.

This is third line of program

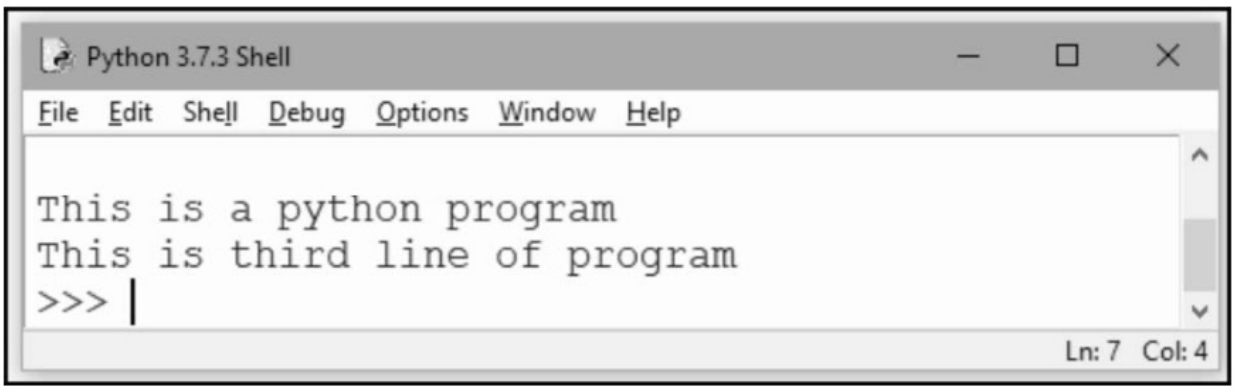
this python program is nice

Solution :

FileDemo1.py

```
fh=open('d:\\test.txt')
i=0
for line in fh:
   line=line.rstrip()
   if line.find('This')==-1:continue
   print(line)
```

Output



Example 6.2.5 : *Write a program in Python to split the text line written in the file into words.*

Solution :

```
with open('d:\\test.txt','rt') as inf:
   line=inf.readline()
   word_list=line.split()
   print(word_list)
```

Output

```
['This', 'is', 'a', 'python', 'program']
>>>
```

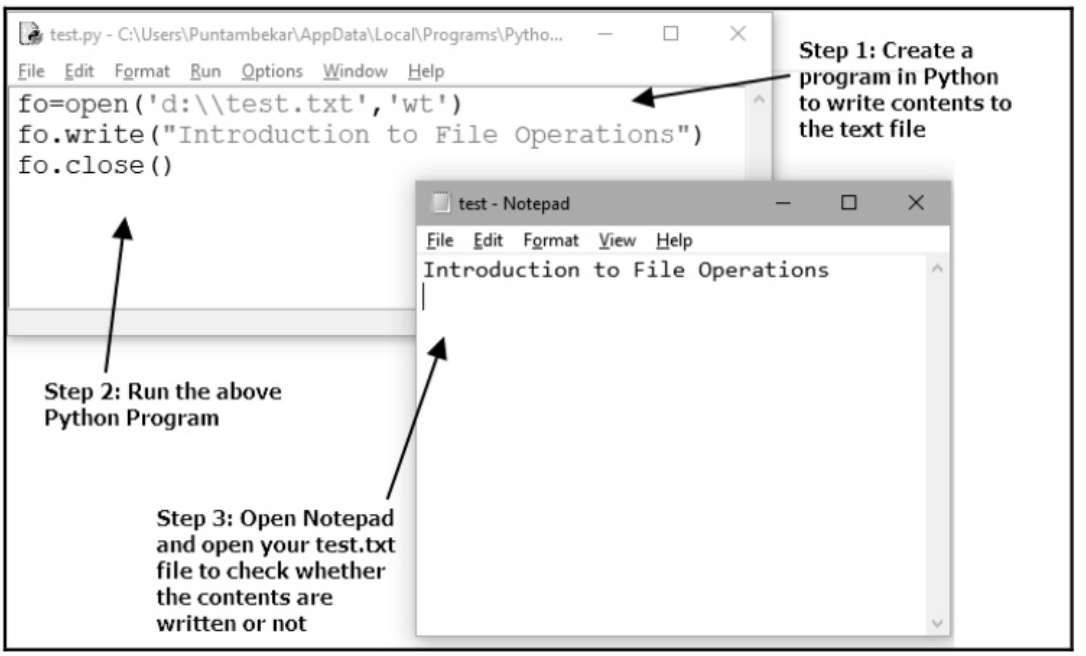
6.2.2.2 Writing Files

- For writing the contents to the file, we must open it in writing mode. Hence we use 'w' or 'a' or 'x' as a file mode.
- The write method is used to write the contents to the file.

Syntax

File_object.write(contents)

- The write method returns number of bytes written.
- While using the 'w' mode in open, just be careful otherwise it will overwrite the already written contents. Hence in the next subsequent write commands we use "\n" so that the contents are written on the next lines in the file.
- For example :



The writelines() method

The writelines() method is used to write list of strings to the file. Following program illustrates this

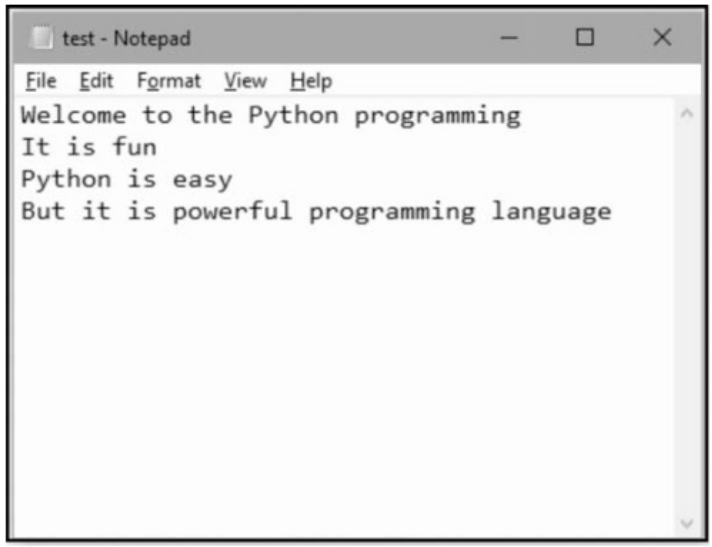
Example 6.2.6 : *Write a python program to write multiple lines to a text file using writelines() method*

Solution :

```
fo=open('d:\\test.txt','wt')
lines=["Welcome to the Python programming\n","It is fun\n",
    "Python is easy\n", "But it is powerful programming language"]
fo.writelines(lines)
fo.close()
```

Output

Now open the Notepad and open the test.txt file in it. It will be something like this -



Appending the file

Appending the file means inserting records at the end of the file.

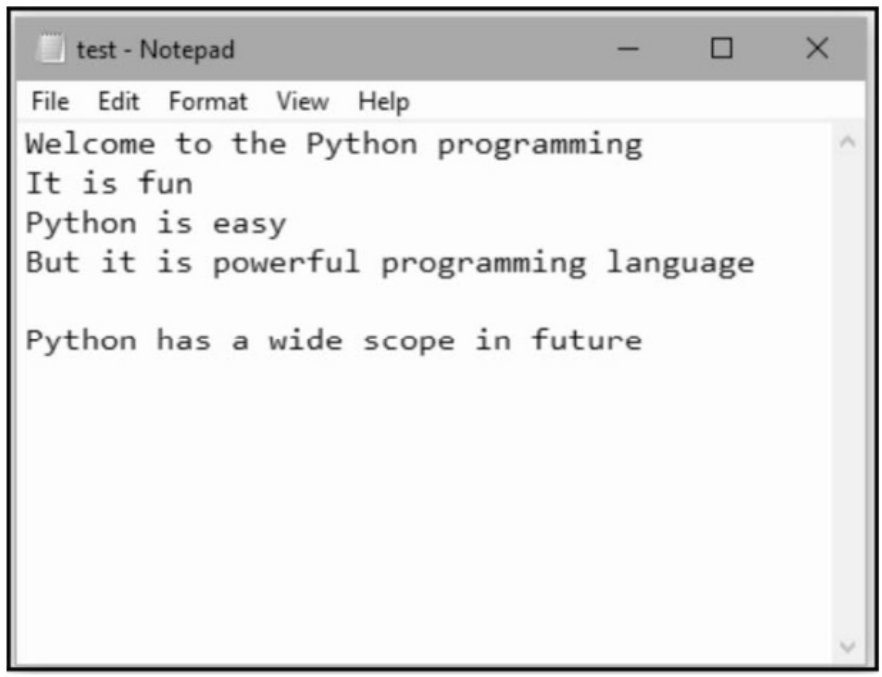
For appending the file we must open the file in 'a' or 'ab' mode.

For example

```
fo=open('d:\\test.txt','a')
fo.write("Python has a wide scope in future")
fo.close()
```

Output

Just open the existing d:\test.txt file to check the contents. It will be as follows –

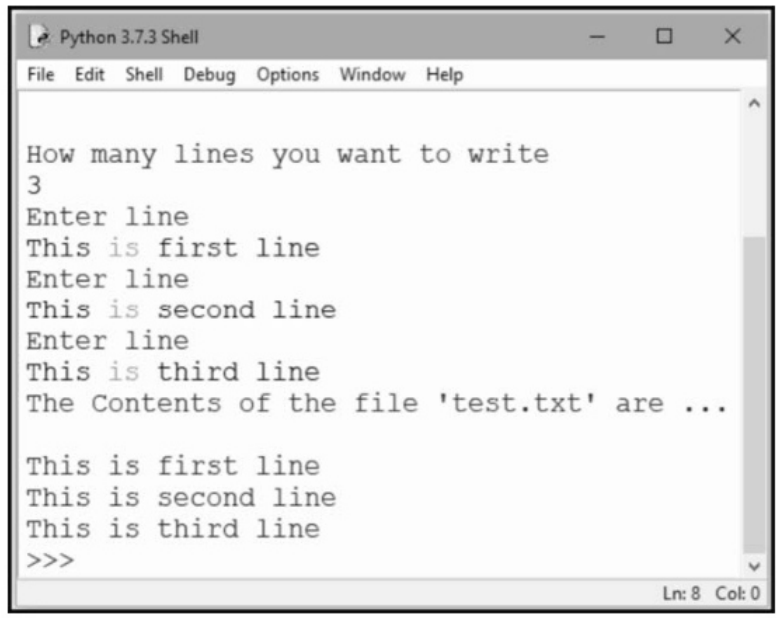


Example 6.2.7 : *Write a python program to write n number of lines to the file and display all these lines as output.*

Solution :

```
print("How many lines you want to write")
n=int(input())
outFile=open("d:\\test.txt",'wt')
for i in range(n):
   print("Enter line")
   line=input()
   outFile.write("\n"+line)
outFile.close()
print("The Contents of the file 'test.txt' are ...")
inFile=open("d:\\test.txt",'rt')
print(inFile.read())
inFile.close()
```

Output



Example 6.2.8 : *Write a python program to write the contents in 'one.txt' file. Read these contents and write them to another file named 'two.txt'*

Solution :

```
print("How many lines you want to write")
n=int(input())
outFile=open("d:\\one.txt",'wt')
for i in range(n):
   print("Enter line")
   line=input()
   outFile.write("\n"+line)
outFile.close()
inFile=open("d:\\one.txt",'rt')
outFile=open("d:\\two.txt",'wt')
i=0
for i in range(n+1):
```

```
    line=inFile.readline()
    outFile.write("\n"+line)
inFile.close()
outFile.close()
print("The Contents of the file 'two.txt' are ...")
inFile=open("d:\\two.txt",'rt')
print(inFile.read())
inFile.close()
```

6.2.3 Closing Files

After performing all the file operations, it is necessary to close the file.

Closing of file is necessary because it frees all the resources that are associated with file.

Example -

```
fo=open("test.txt",rt)
fo.close()  #closing a file.
```

6.2.4 Renaming and Deleting Files

(i) Renaming File

For renaming a file in Python, the **rename()** method is used. For that purpose, it is required to import the module **os**.

Syntax :

```
os.rename(src, dst)
```

Where

src: The file name with its complete path which is to be renamed.

dst: The new name for the file.

renameDemo.py

```
import os
os.rename(r'd:\test1.txt',r'D:\test2.txt')
print("File is renamed")
```

Output

```
File is renamed
```

To check if the file name is changed or not, we must find the file at corresponding location, and check the contents by opening it.

(ii) Deleting File

For removing the file, we use **remove** method belonging to the **os** module.

Syntax :

```
os.rename(filename)
```

The code is as follows

```
import os
os.rename('d:\test1.txt')
print("File is removed")
```

Output

```
File is removed
```

6.2.5 Directories in Python

Finding File Position

The seek and tell method : The **seek** method is used to change the file position. Similarly the **tell** method returns the current position.

The method seek() sets the file's current position at the offset.

Syntax

fileObject.seek(offset[, whence])

Where

- offset − This is the position of the read/write pointer within the file.
- whence − This is optional and defaults to 0 which means absolute file positioning, other values are 1 which means seek relative to the current position and 2 means seek relative to the file's end.

The method tell() returns the current position of the file read/write pointer within the file.

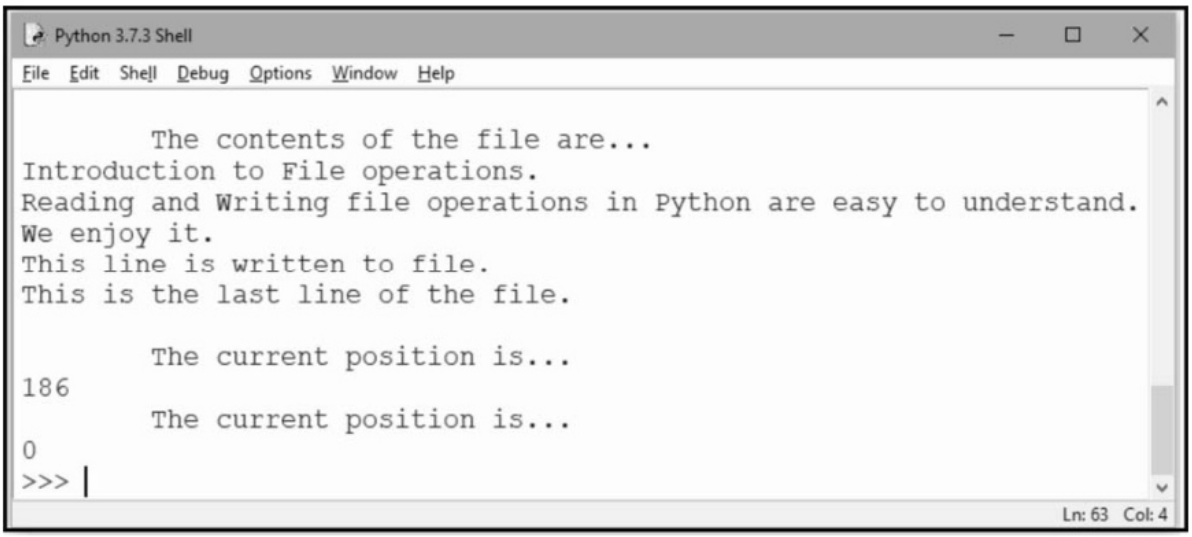
Syntax

fileObject.tell()

Programming Example

```
inf=open('d:\\test.txt','rt')
print("\tThe contents of the file are...")
print(inf.read())
print("\tThe current position is...")
print(inf.tell())#get current postion of file
inf.seek(0) #moves the file cursor to initial position
print("\tThe current position is...")
print(inf.tell())#get current position of file
```

Output



We can pass number of bytes to the seek method. These many bytes are skipped and then remaining contents are displayed. For example

```
>>> inf.seek(10)
10
>>> print(inf.read())
  #note that 10 bytes are skipped and then the remaining contents are displayed
on to File Operations
 Reading and writing operations in Python are easy
 We enjoy it
 This line is written in file
 This is a last line of this file
>>>
```

Directory Methods

In this section we will discuss various directory methods.

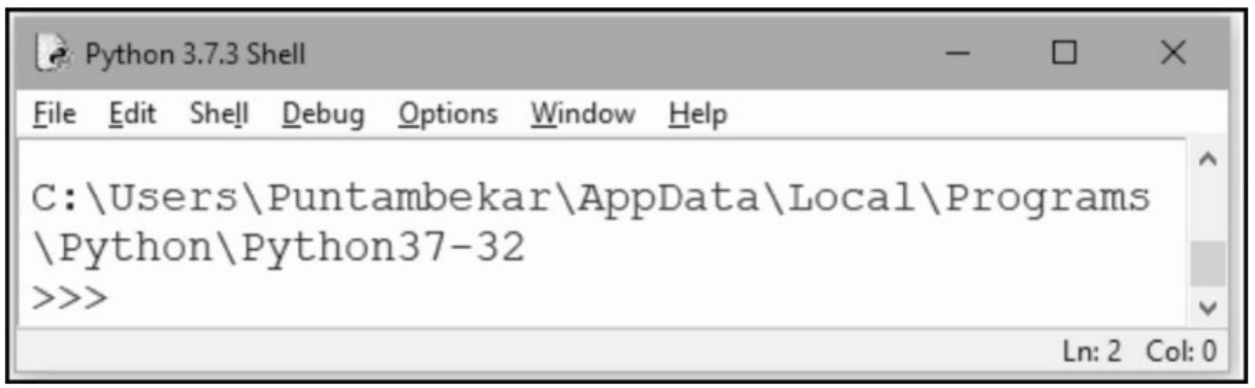
1) Getting current working directory

For getting the name of current working directory we use the function named **getcwd()** method. This method returns the name of current working directory. For example –

getwdDemo.py

```
import os
print(os.getcwd())
```

Output



2) Displaying all the files and sub-directories inside a directory

For listing the contents of a directory we use the method **listdir().** For example –

```
import os
print(os.listdir())
```

3) Creating a new Directory

We can create a new directory using mkdir() method

mkdirDemo.py

```
import os
os.mkdir("mypythonPrograms") #creates a folder named mypythonPrograms
print(os.listdir()) #displaying the directory contents
```

4) Removing the directory or a file

A file can be deleted using remove() method.

The rmdir() method removes an empty directory.

For example

```
import os
os.remove("test.txt")#removes file test.txt
print(os.listdir()) #displays the directory contents
os.rmdir("mypythonPrograms") #removes the directory named 'mypythonPrograms'
print(os.listdir())#displays the directory contents
```

6.2.6 Programs on File Handling

Example 6.2.9 : *Write a program in Python to count number of vowels and consonants present in the text file*
Solution :

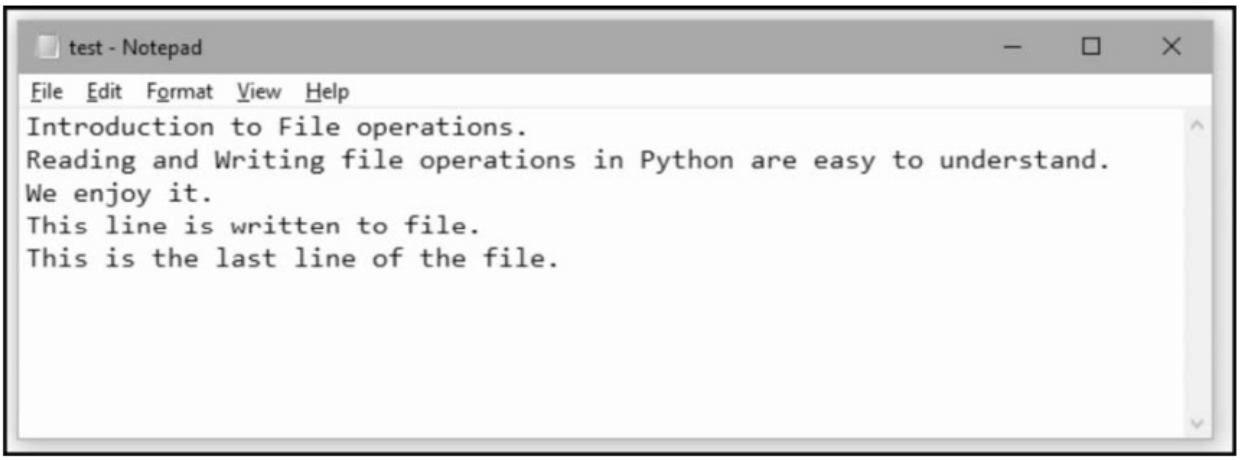
```
infile = open("d:\\test.txt", "r")
vowels = set("AEIOUaeiou")
cons = set("bcdfghjklmnpqrstvwxyzBCDFGHJKLMNPQRSTVWXYZ")
count_Vowels = 0
count_Cons = 0
for c in infile.read():
   if c in vowels:
      count_Vowels += 1
   elif c in cons:
      count_Cons += 1
print("Number of vowels in a file are: ",count_Vowels)
print("Number of consonants in a file are: ",count_Cons)
infile.close()
```

Output

```
Number of vowels in a file are:  59
Number of consonants in a file are:  85
>>>
```

Example 6.2.10 : *Write a Python program to count total number of lines in a file*
Solution :

Step 1 : Create a text file named test.txt as follows



Step 2 : Now write the python code to count the total number of lines in a file

```
count=0
with open("D:\\test.txt",'r') as f:
   for line in f:
      word=line.split()
```

```
        count+=1
f.close()
print("Total number of lines in file are: ",count)
```

Output

```
Total number of lines in file are:  5
>>>
```

Example 6.2.11 : *Write a Python program to count total number of words in a file*
Solution :

Step 1 : Create a text file names test.txt as follows –

Introduction to File operations.

Reading and Writing file operations in Python are easy to understand.

We enjoy it.

This line is written to file.

This is the last line of the file.

Step 2 : Now write a Python code to count number of words in the file. The code is as follows –

```
count=0
with open("D:\\test.txt",'r') as f:
   for line in f:
      word=line.split()
      count+=len(word)
f.close()
print("Total number of words in file are: ",count)
```

Output

```
Total number of words in file are:  32
>>>
```

Example 6.2.12 : *Write a Python program to count total number of characters in a file*
Solution :

Step 1 : Create a text file names test.txt as follows –

Introduction to File operations.

Reading and Writing file operations in Python are easy to understand.

We enjoy it.

This line is written to file.

This is the last line of the file.

Step 2 : Now write a Python code to count number of characters in the file. The code is as follows –

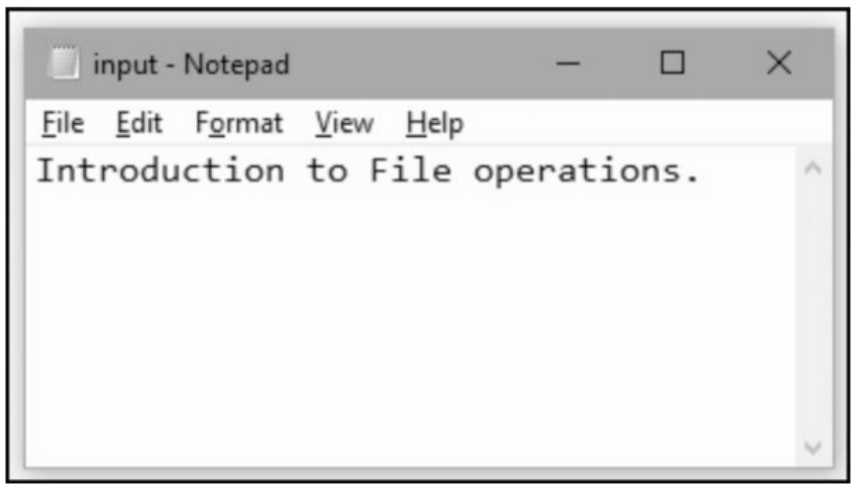
```
count=0
with open("D:\\test.txt",'r') as f:
   for line in f:
      word=line.split()
      count+=len(line)
f.close()
print("Total number of characters in file are: ",count)
```

Output

```
Total number of characters in file are:  181
>>>
```

Example 6.2.13 : *Write a Python program to count the total number of blank spaces in the file*

Solution : First of all create a text file and write some contents to it. Your contents must have some blank characters. Following is input.txt file



Then write the python code to count the number of blanks. Note that it will count newline character as a blank character as well.

```
with open("D:\\input.txt",'r') as f:
   count=0
   for line in f:
      word = line.split()#getting each word of every line
      for letter in word: #getting each letter from each word
            if(letter.isspace):#checking if it is blank
               count=count+1
print("Total Number of blank spaces are: ",count)
```

Output

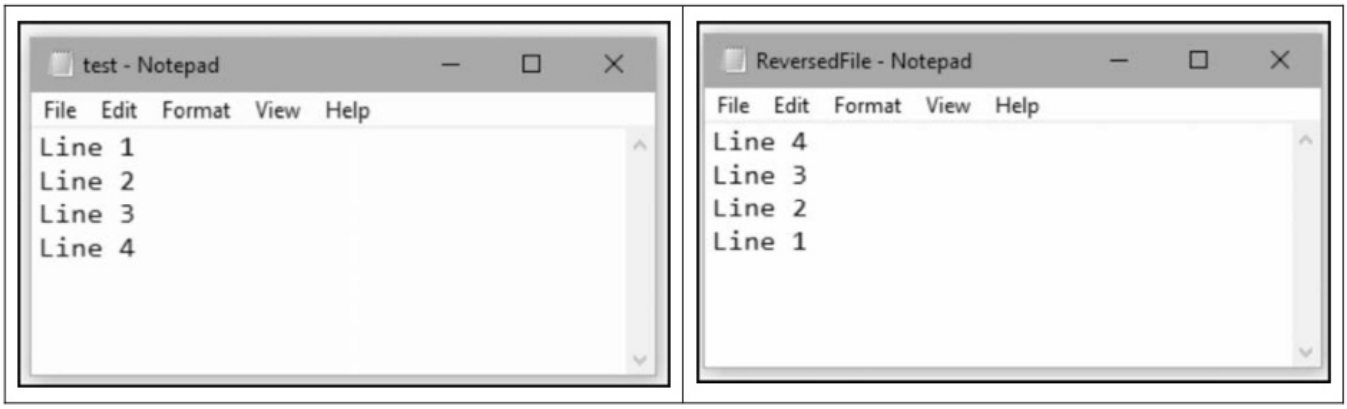
```
Total Number of blank spaces are:  4
>>>
```

Example 6.2.14 : *Write a Python program to write the contents of one file to another file from end.*

Solution :

```
lines = []
with open('d:\\test.txt') as f:
   lines = f.readlines()
f.close()
with open('d:\\ReversedFile.txt', 'w') as f:
   for line in reversed(lines):
      f.write(line)
f.close()
```

Output



Review Questions

1. *What is file ? Explain the concept of Relative and Absolute Path.*
2. *Explain the file open method.*
3. *Explain different functions used with respect to file positions*
4. *What is file ? Explain how to perform file read and write operations in Python ?*

6.3 Exception Handling

- **Definition of exception :** An exception is an event, which occurs during the execution of a program that disrupts the normal flow of the program.
- In general, when a python script encounters a situation that it cannot cope with, it raises an exception.
- When a python script raises an exception, it must either handle the exception immediately otherwise it terminates and quits.

6.3.1 try:Except Statement

- The exception handling mechanism using the **try…except…else** blocks.
- The suspicious code is placed in try block.
- After try block place the except block which handles the exception elegantly.
- If there is no exception then the else block statements get executed.
- **Syntax of try…except…else**

```
try:
                    write the suspicious code here
except Exception 1:
                    If Exception 1 occurs then execute this code
except Exception 2:
                    If Exception 2 occurs then execute this code
…
…
else:
                    If there is no exception then execute this code.
```

- Example : python program to perform division of two numbers.

```
try :
  a=int(input("Enter value of a: "))
   b=int(input("Enter value of b: "))
```

```
    c=a/b
except ValueError:
    print("You have entered wrong data")
except ZeroDivisionError:
    print("Divide by Zero Error!!!")
else:
    print("The result: ",c)
```

6.3.2 The Raise Statement

- Raising an exception is a technique for interrupting the normal flow of execution in a program, signaling that some exceptional circumstance has arisen, and returning directly to an enclosing part of the program that was designated to react to that circumstance.
- The Python interpreter raises an exception each time it detects an error in an expression or statement.
- However, Users can also raise exceptions with raise statement.
- **Syntax**

```
Raise [Exception [, args [,traceback]]]
```

Here,

Exception is a type of exception (For example NameError)

args is value for exception argument.

traceback is object of traceback

- **Example**

```
def NegativeNum( num ):
  if num < 0:
    raise "Number is negative!", num
```

6.3.3 Standard Built in Exceptions

Python provide standard built in Exceptions. These exceptions are enlisted in the following table.

Name	Purpose
Exception	Base class for all exceptions
ArithmeticError	Base class for all errors that occur for numeric calculation.
OverflowError	Raised when a calculation exceeds maximum limit for a numeric type.
FloatingPointError	Raised when a floating point calculation fails.
ZeroDivisionError	Raised when division or modulo by zero takes place for all numeric types.
EOFError	Raised when there is no input from either the raw_input() or input() function and the end of file is reached.
ImportError	Raised when an import statement fails.
KeyboardInterrupt	Raised when the user interrupts program execution, usually by pressing Ctrl+c.

NameError	Raised when an identifier is not found in the local or global namespace.
IOError	Raised when an input/ output operation fails.
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.
ValueError	Raised when the built-in function for a data type has the valid type of arguments, but the arguments have invalid values specified.
RuntimeError	Raised when a generated error does not fall into any category.

6.3.4 User Defined Exception

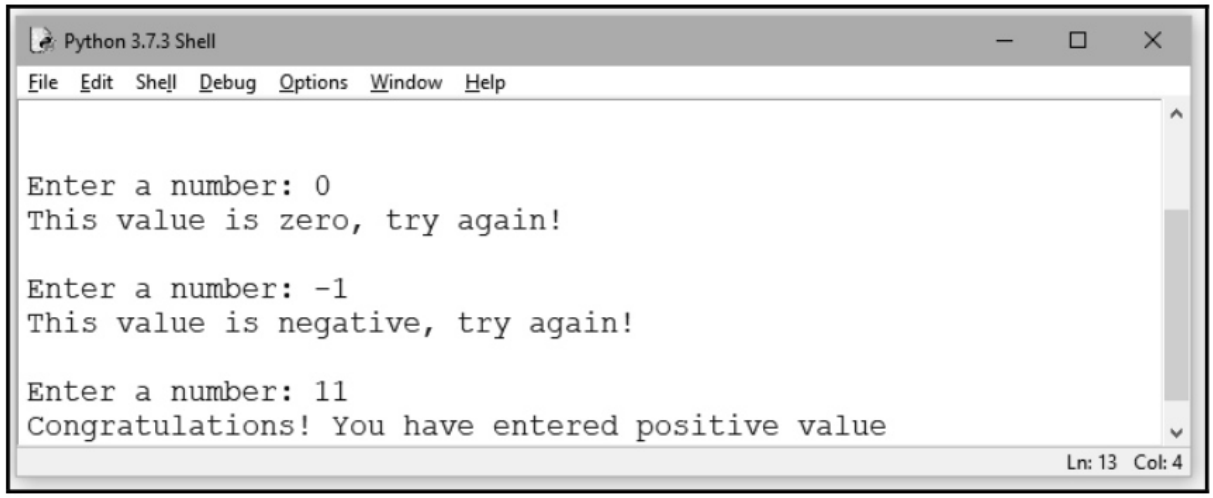
- Although standard exceptions in Python provide adequate functionality, It is advantageous to create user defined exception.
- With user defined exception, one can add the desired information to the exception handlers.
- **IOError** is a generic exception used for input/output problems, which may arise from invalid file accessor other forms of communication.
- Similarly, the user defined exception class can be derived from **Exception** class.
- **For example :** Following program makes use of user defined exception for accepting the positive value only. If user enters negative or zero value, then exception is raised.

usrEx.py

```
class Error(Exception):
  """Base class for other exceptions"""
  pass
class NegativeValue(Error):
  """Raised when the input value is negative"""
  pass
class ZeroValue(Error):
  """Raised when the input value is zero"""
  pass
# our main program
while True:
  try:
    i_num = int(input("Enter a number: "))
    if i_num < 0:
      raise NegativeValue
    elif i_num == 0:
      raise ZeroValue
    break
  except NegativeValue:
```

```
        print("This value is negative, try again!")
        print()
    except ZeroValue:
        print("This value is zero, try again!")
        print()
print("Congratulations! You have entered positive value")
```

Output



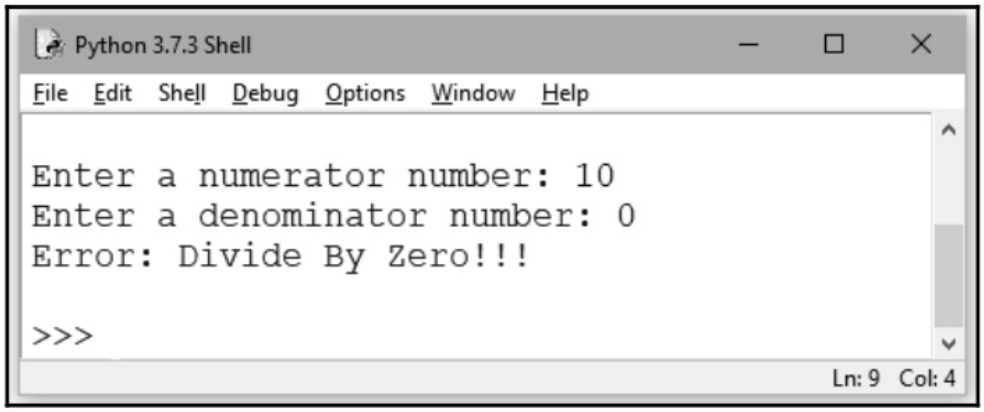
Example 6.3.1 : *Write a Python program to raise divide by zero error using user defined exception*

Solution:

UsrdefEx1.py

```
# define Python user-defined exceptions
class Error(Exception):
  """Base class for other exceptions"""
  pass
class DivideByZero(Error):
  """Raised when the denominator value is zero"""
  pass
# our main program
try:
  a = int(input("Enter a numerator number: "))
  b = int(input("Enter a denominator number: "))
  if b == 0:
    raise DivideByZero
  c=a/b
  print("The result = ",c)
except DivideByZero:
  print("Error: Divide By Zero!!!")
  print()
```

Output



Review Questions

1. *What is exception ?*
2. *Explain exception handling mechanism in Python.*
3. *Explain the use of raise statement in Python with suitable example.*
4. *Explain how to create and use the user defined exception.*

❑❑❑