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.

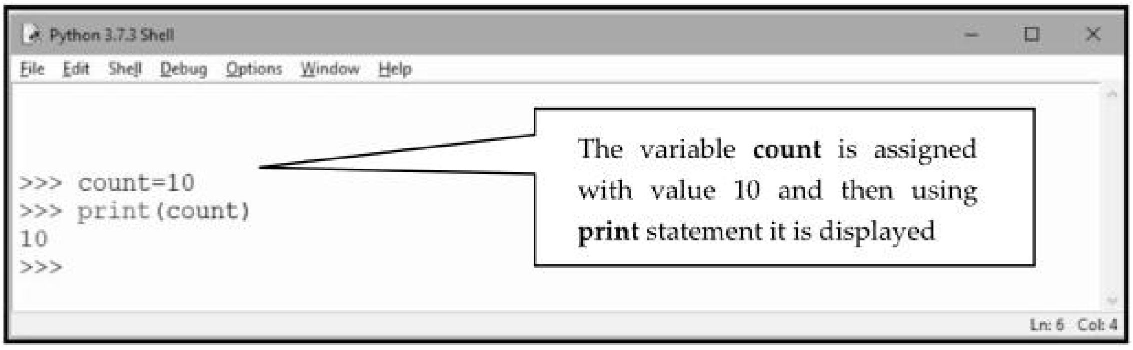
Review Question

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.



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

The variable **count** is assigned with value 10 and then using **print** statement it is displayed

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

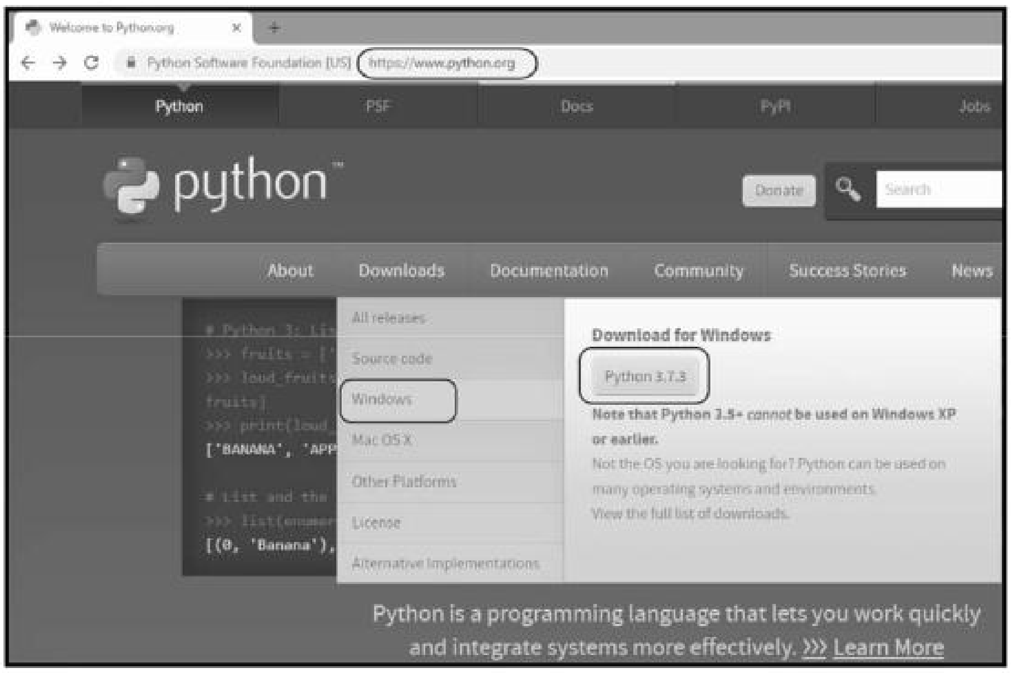
```
#This is
#another example
#of comment statement
```

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**



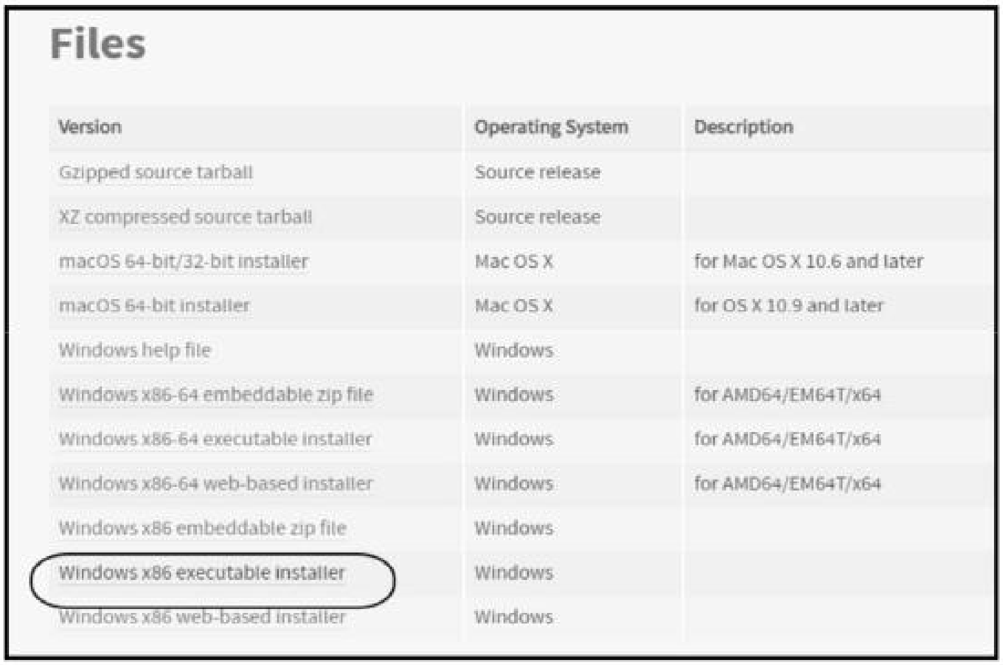
Note that Python 3.5+ *cannot* be used on Windows XP or earlier.

Not the OS you are looking for? Python can be used on many operating systems and environments.

View the full list of downloads.

Python is a programming language that lets you work quickly and integrate systems more effectively. >>> Learn More

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



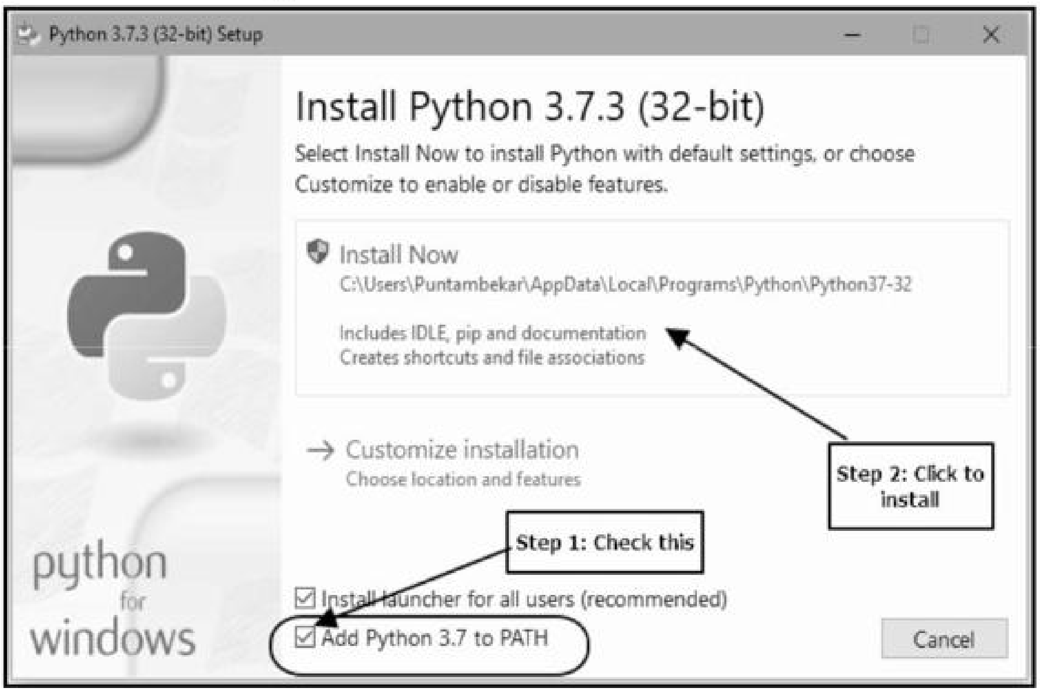
Files

Version	Operating System	Description
Gzipped source tarball	Source release	
XZ compressed source tarball	Source release	
macOS 64-bit/32-bit installer	Mac OS X	for Mac OS X 10.6 and later
macOS 64-bit installer	Mac OS X	for OS X 10.9 and later
Windows help file	Windows	
Windows x86-64 embeddable zip file	Windows	for AMD64/EM64T/x64
Windows x86-64 executable installer	Windows	for AMD64/EM64T/x64
Windows x86-64 web-based installer	Windows	for AMD64/EM64T/x64
Windows x86 embeddable zip file	Windows	
Windows x86 executable installer	Windows	
Windows x86 web-based installer	Windows	

of 184 − | + 70%

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.


Python 3.7.3 (32-bit) Setup

Install Python 3.7.3 (32-bit)

Select Install Now to install Python with default settings, or choose Customize to enable or disable features.

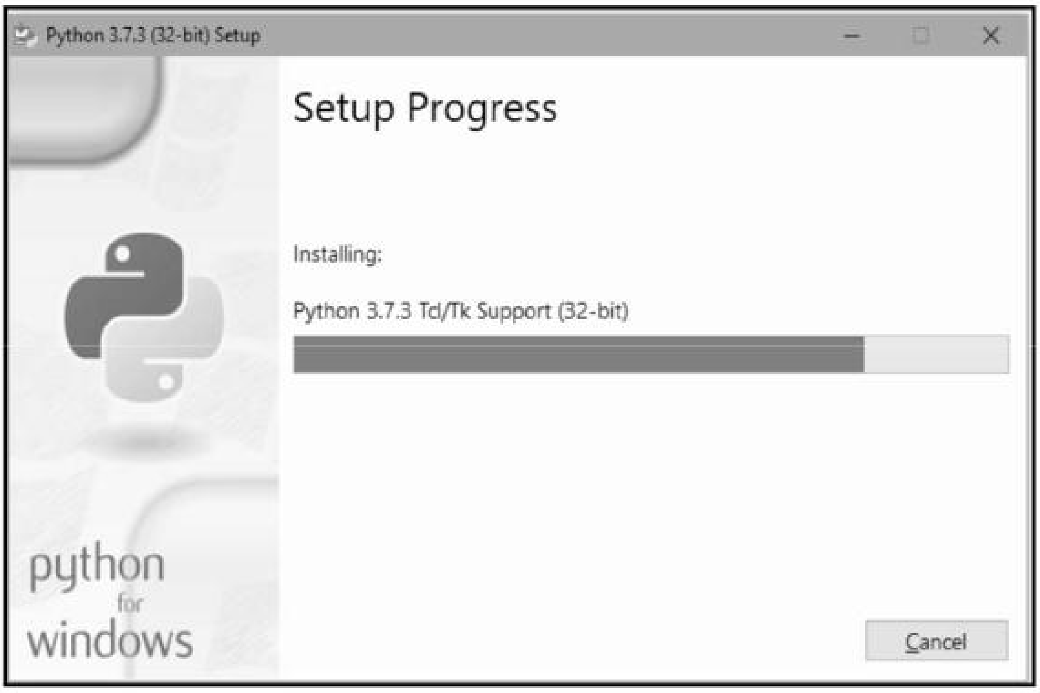
Install Now
C:\Users\Puntambekar\AppData\Local\Programs\Python\Python37-32

Includes IDLE, pip and documentation
Creates shortcuts and file associations

→ Customize installation
Choose location and features

☑ Install launcher for all users (recommended)
☑ Add Python 3.7 to PATH

The installation will progress.


Python 3.7.3 (32-bit) Setup

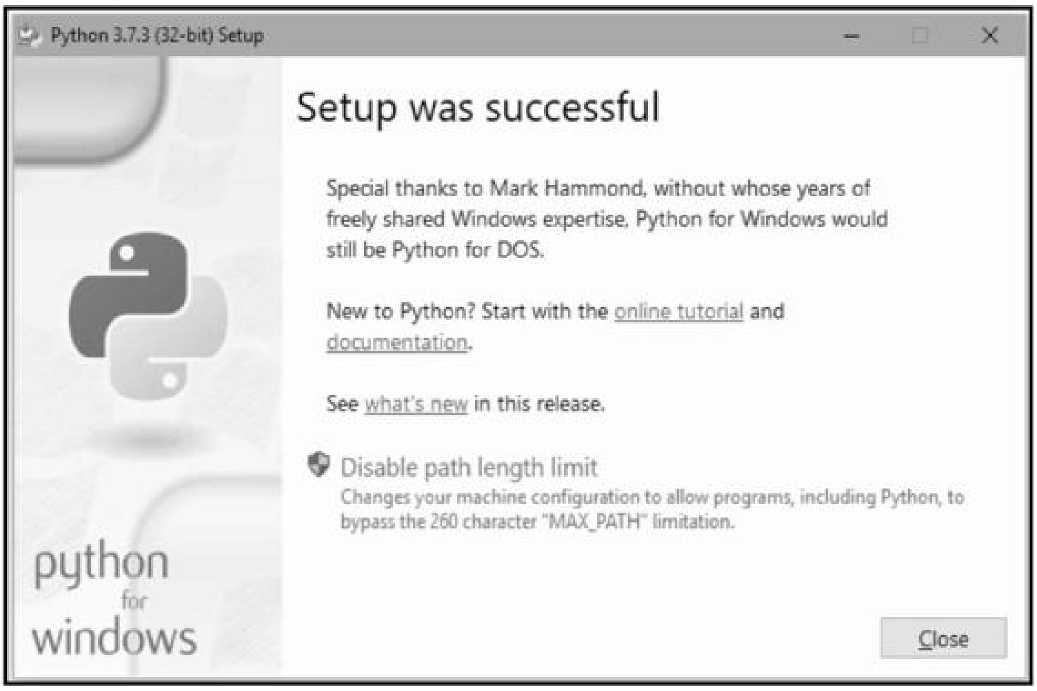
Setup Progress

Installing:

Python 3.7.3 Tcl/Tk Support (32-bit)

of 184 70%

Finally you will get successful installation message.



Python 3.7.3 (32-bit) Setup

Setup was successful

Special thanks to Mark Hammond, without whose years of freely shared Windows expertise, Python for Windows would still be Python for DOS.

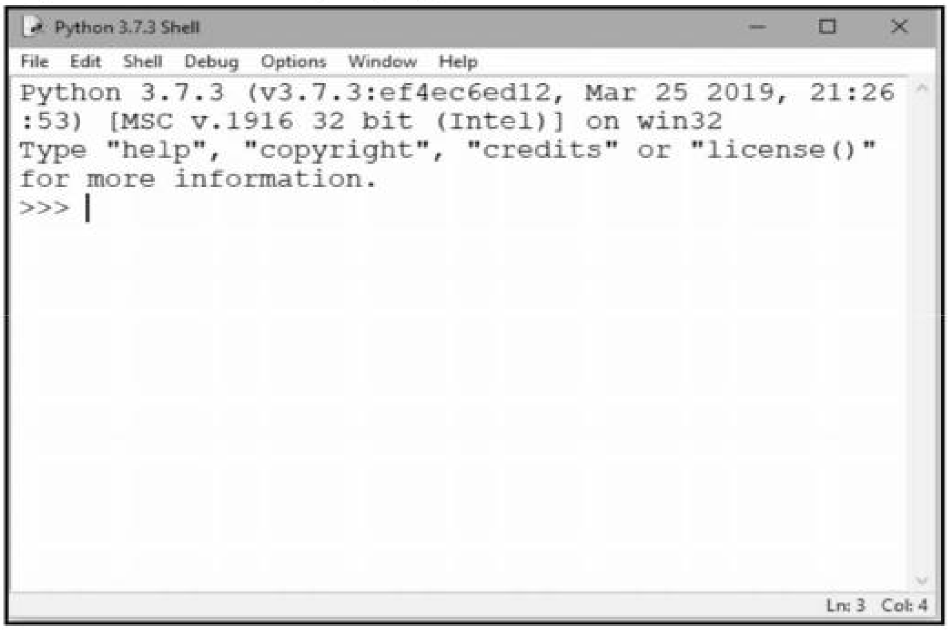
New to Python? Start with the online tutorial and documentation.

See what's new in this release.

Disable path length limit
Changes your machine configuration to allow programs, including Python, to bypass the 260 character "MAX_PATH" limitation.

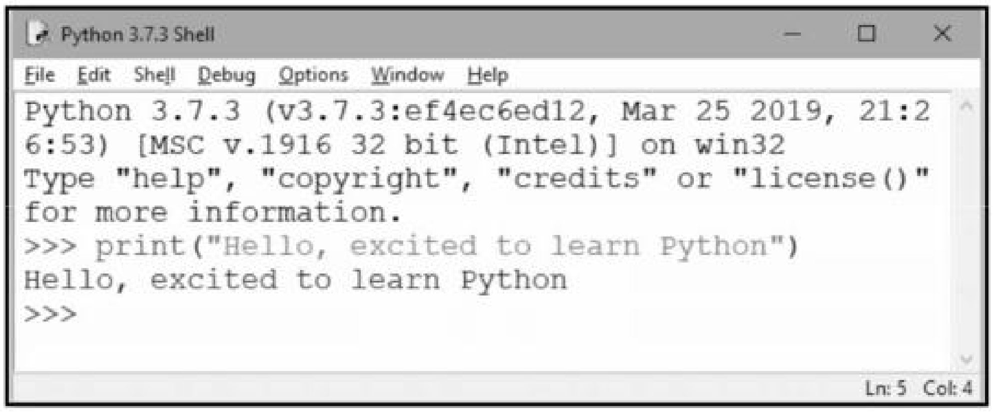
Congratulations!!!. The Python is installed on your PC. Now you can write and execute the programs in Python using Python Shell.

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 –



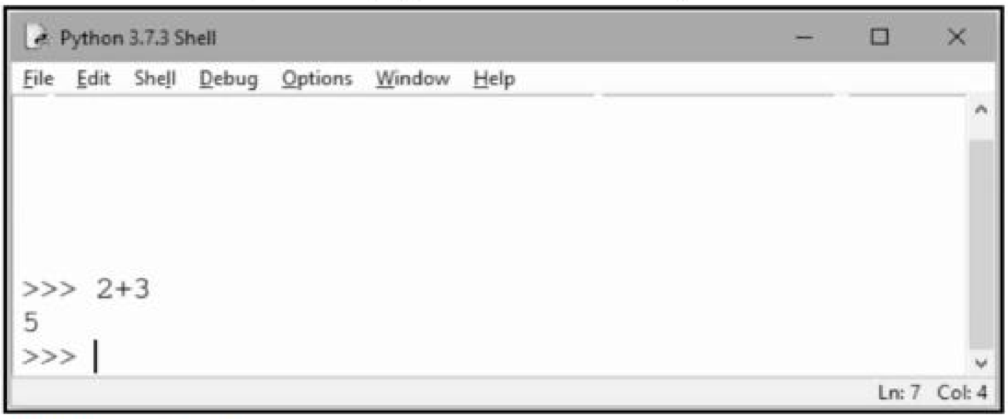
```
Python 3.7.3 (v3.7.3:ef4ec6ed12, Mar 25 2019, 21:26
:53) [MSC v.1916 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()"
for more information.
>>> |
```

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**



```
Python 3.7.3 (v3.7.3:ef4ec6ed12, Mar 25 2019, 21:2
6:53) [MSC v.1916 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()"
for more information.
>>> print("Hello, excited to learn Python")
Hello, excited to learn Python
>>>
```

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



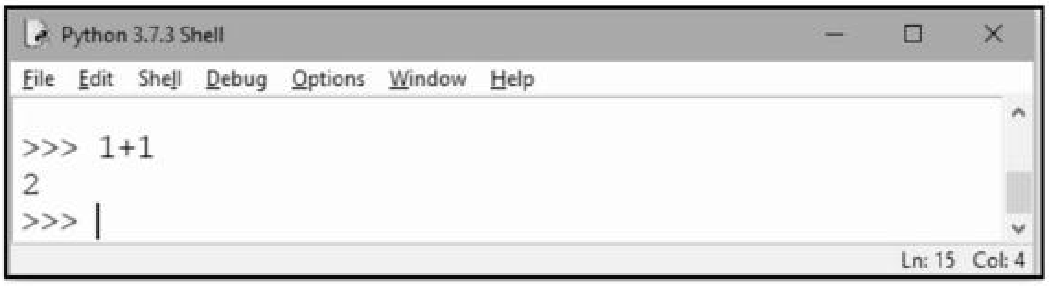
```
>>> 2+3
5
>>> |
```

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.



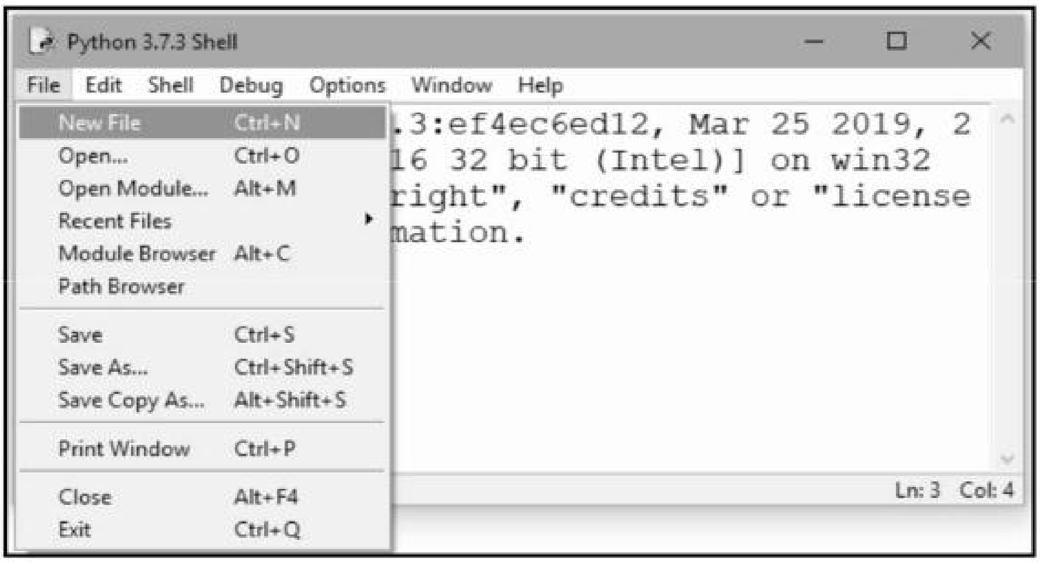
```
>>> 1+1
2
>>> |
```

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.

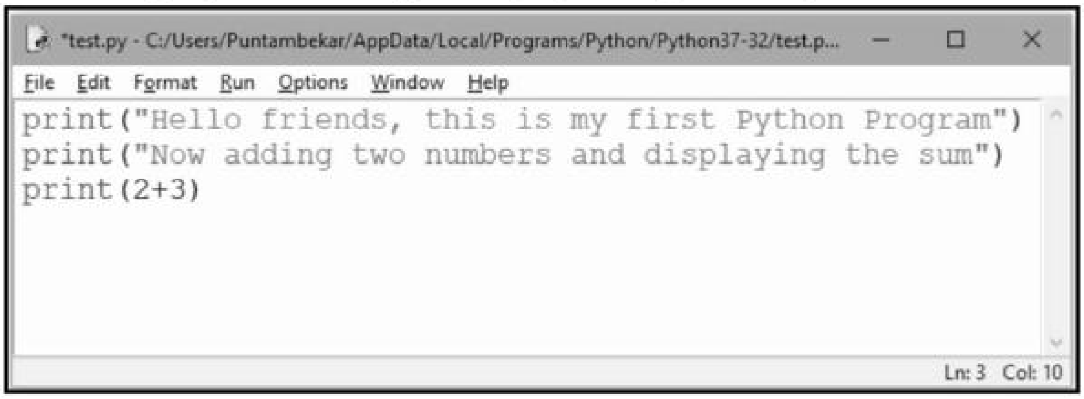
Step 2 : On File Menu Click on **New File** option.



```
.3:ef4ec6ed12, Mar 25 2019, 2
16 32 bit (Intel)] on win32
right", "credits" or "license
mation.
```

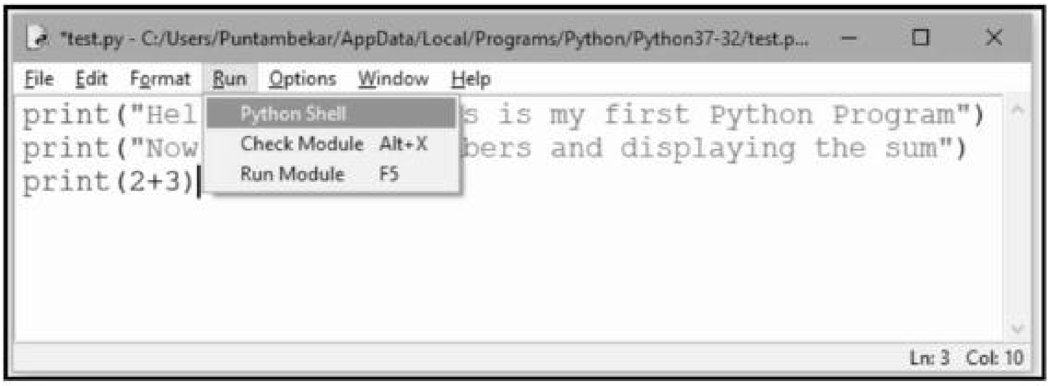
Step 3 : Give some suitable file name with extension .py (I have created **test.py**).

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


*test.py - C:/Users/Puntambekar/AppData/Local/Programs/Python/Python37-32/test.p...
File Edit Format Run Options Window Help
Ln: 3 Col: 10

```
print("Hello friends, this is my first Python Program")
print("Now adding two numbers and displaying the sum")
print(2+3)
```

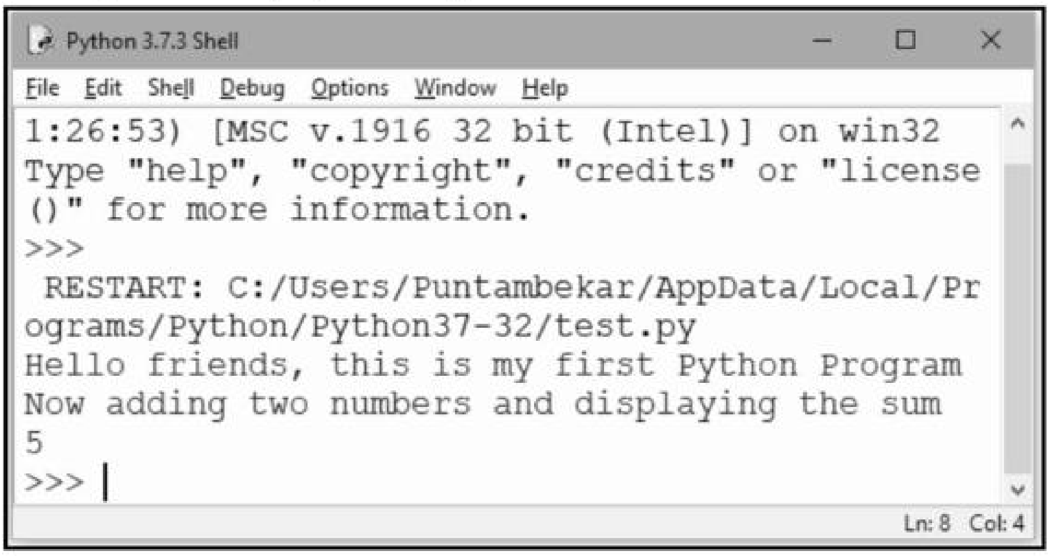
Step 5 : Now run your code by clicking on **Run ->Run Module** on Menu bar. Following screenshot illustrates it


*test.py - C:/Users/Puntambekar/AppData/Local/Programs/Python/Python37-32/test.p...
File Edit Format Run Options Window Help
Python Shell
Check Module Alt+X
Run Module F5
Ln: 3 Col: 10

```
print("Hel                  s is my first Python Program")
print("Now                  bers and displaying the sum")
print(2+3)
```

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



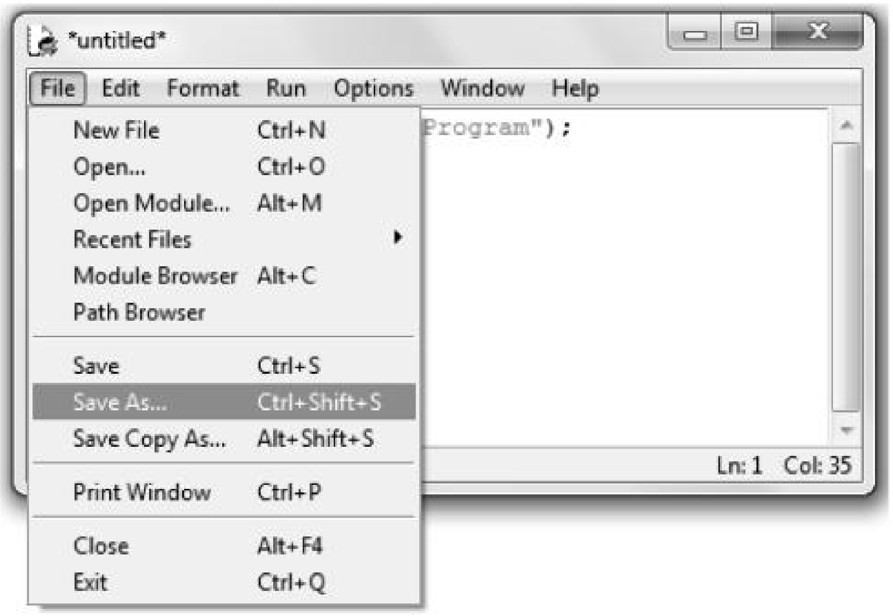
```
1:26:53) [MSC v.1916 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license
()" for more information.
>>>
 RESTART: C:/Users/Puntambekar/AppData/Local/Pr
ograms/Python/Python37-32/test.py
Hello friends, this is my first Python Program
Now adding two numbers and displaying the sum
5
>>> |
```

1.4 Running Simple Python Script to Display 'welcome' Message

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

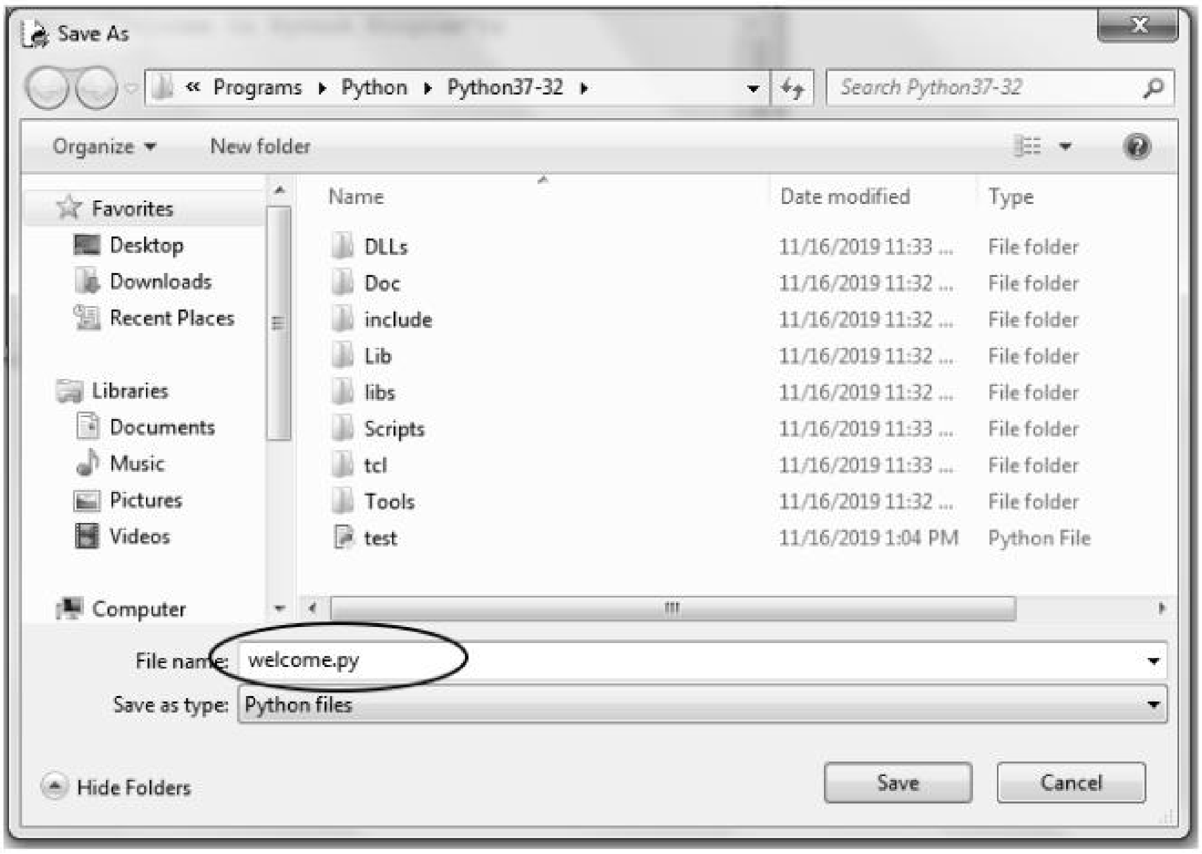
f 184 − + 70%

Step 4 : Now select File -> Save As Option



```
Program");
```

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



Name	Date modified	Type
DLLs	11/16/2019 11:33 ...	File folder
Doc	11/16/2019 11:32 ...	File folder
include	11/16/2019 11:32 ...	File folder
Lib	11/16/2019 11:32 ...	File folder
libs	11/16/2019 11:32 ...	File folder
Scripts	11/16/2019 11:33 ...	File folder
tcl	11/16/2019 11:33 ...	File folder
Tools	11/16/2019 11:32 ...	File folder
test	11/16/2019 1:04 PM	Python File

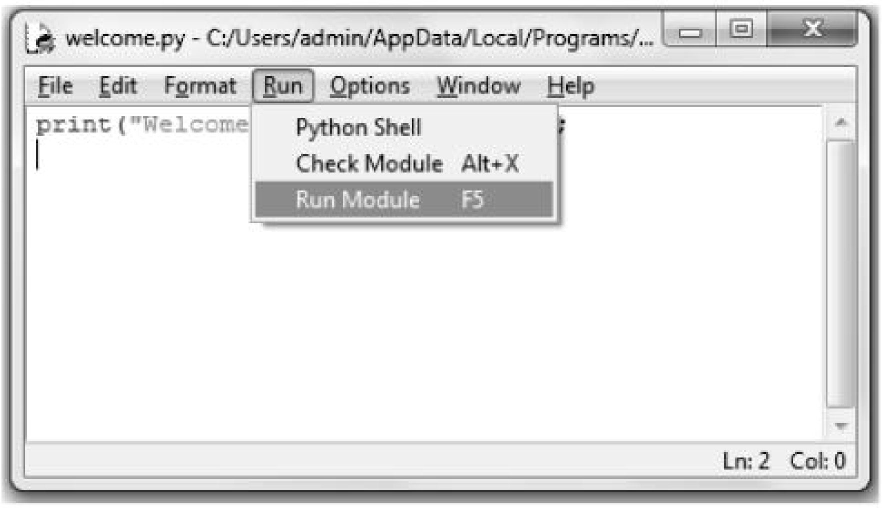
File name: welcome.py
Save as type: Python files

of 184 − + 70%

welcome.py

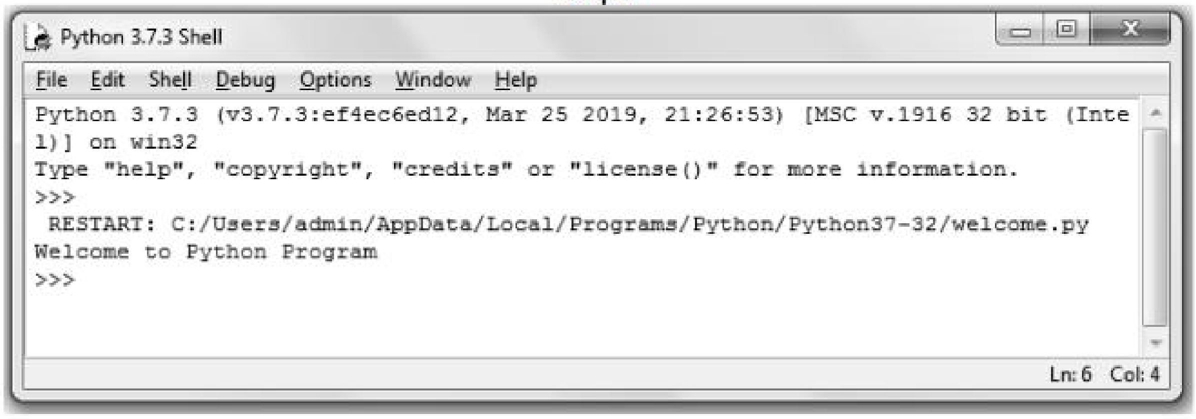
print('Welcome to Python Program');

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


welcome.py - C:/Users/admin/AppData/Local/Programs/...
File Edit Format Run Options Window Help
Python Shell
Check Module Alt+X
Run Module F5
Ln: 2 Col: 0

```
print("Welcome
```

Output



```
Python 3.7.3 (v3.7.3:ef4ec6ed12, Mar 25 2019, 21:26:53) [MSC v.1916 32 bit (Inte
l)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
 RESTART: C:/Users/admin/AppData/Local/Programs/Python/Python37-32/welcome.py
Welcome to Python Program
>>>
```

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 –

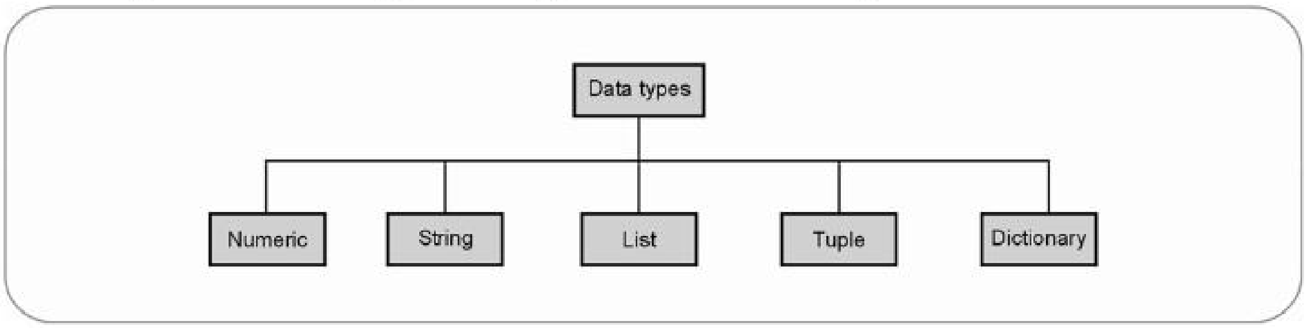
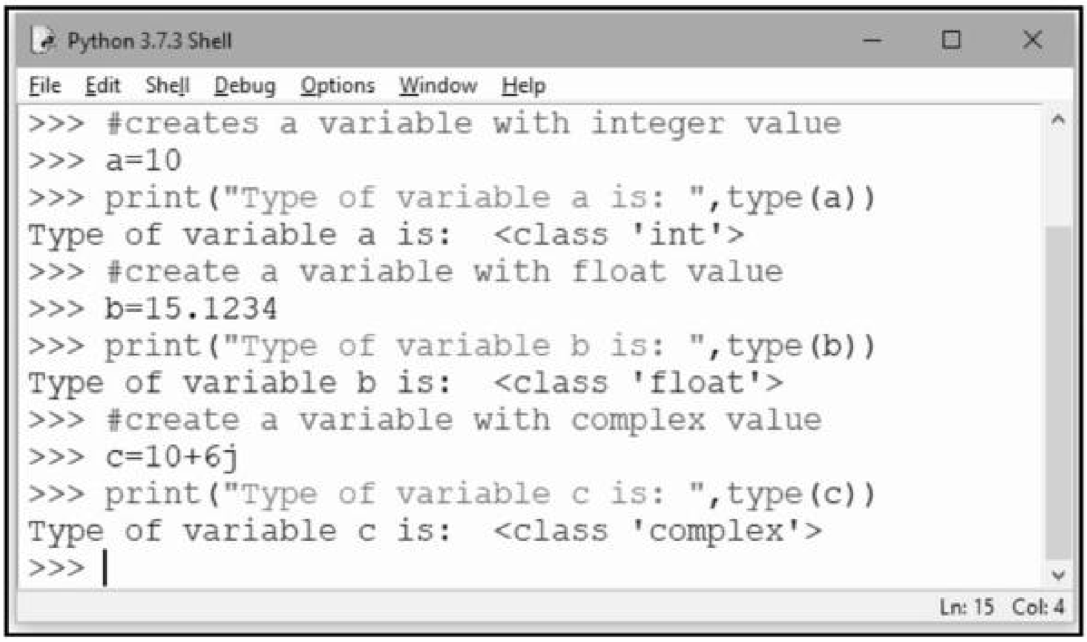


Fig. 1.5.1 Data types in Python

(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 -

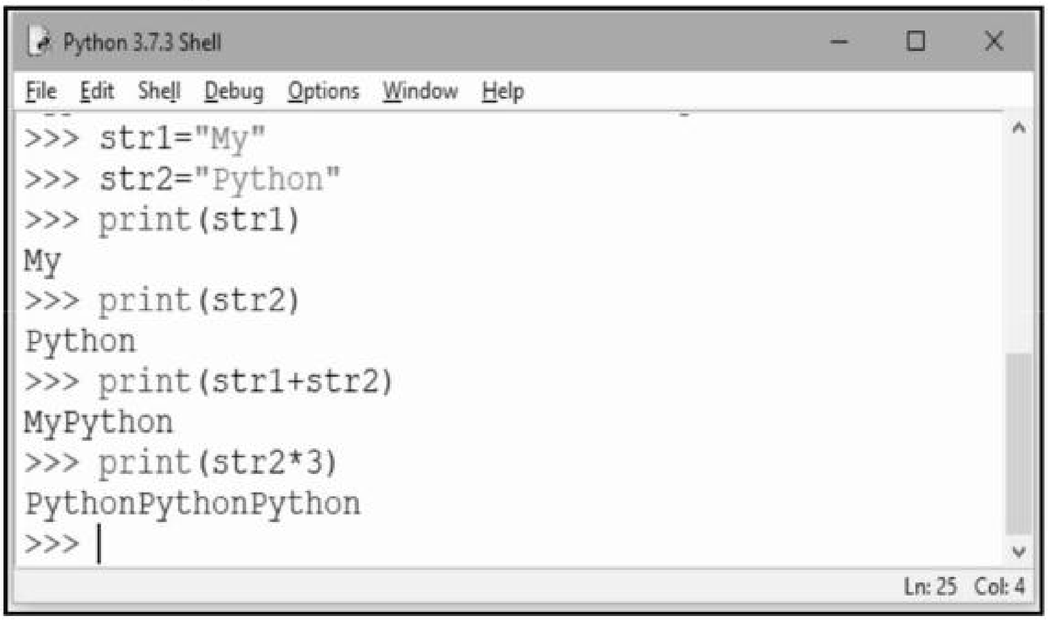


```
>>> #creates a variable with integer value
>>> a=10
>>> print("Type of variable a is: ",type(a))
Type of variable a is:  <class 'int'>
>>> #create a variable with float value
>>> b=15.1234
>>> print("Type of variable b is: ",type(b))
Type of variable b is:  <class 'float'>
>>> #create a variable with complex value
>>> c=10+6j
>>> print("Type of variable c is: ",type(c))
Type of variable c is:  <class 'complex'>
>>> |
```

(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 *.

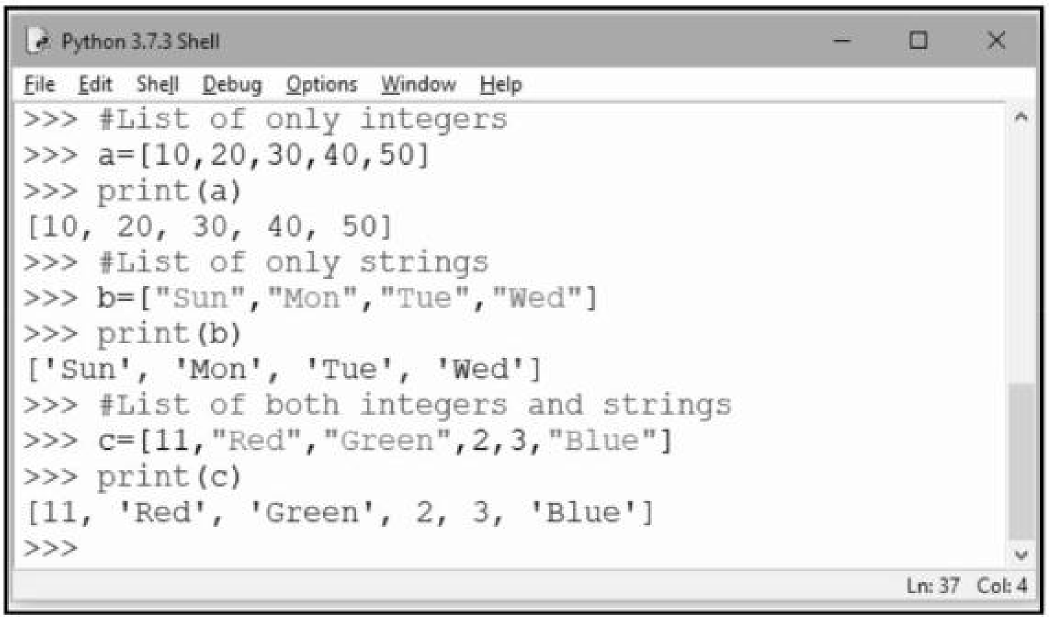
- The + operator is used to concatenate the two strings. While * operator is used as a repetition operation. Following execution illustrates it -



```
>>> str1="My"
>>> str2="Python"
>>> print(str1)
My
>>> print(str2)
Python
>>> print(str1+str2)
MyPython
>>> print(str2*3)
PythonPythonPython
>>> |
```

(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 -



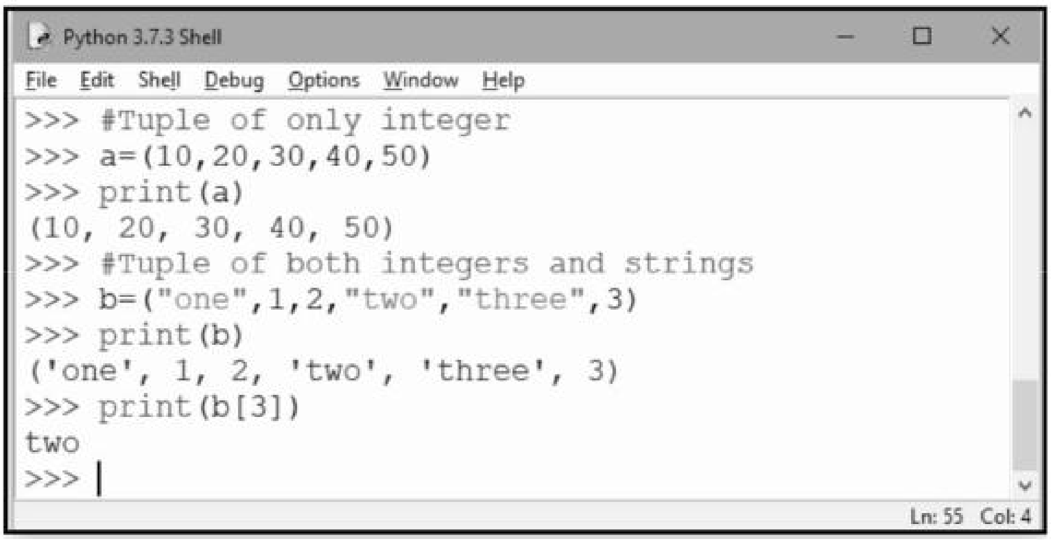
```
>>> #List of only integers
>>> a=[10,20,30,40,50]
>>> print(a)
[10, 20, 30, 40, 50]
>>> #List of only strings
>>> b=["Sun","Mon","Tue","Wed"]
>>> print(b)
['Sun', 'Mon', 'Tue', 'Wed']
>>> #List of both integers and strings
>>> c=[11,"Red","Green",2,3,"Blue"]
>>> print(c)
[11, 'Red', 'Green', 2, 3, 'Blue']
>>>
```

(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.

f 184 − | + 70%

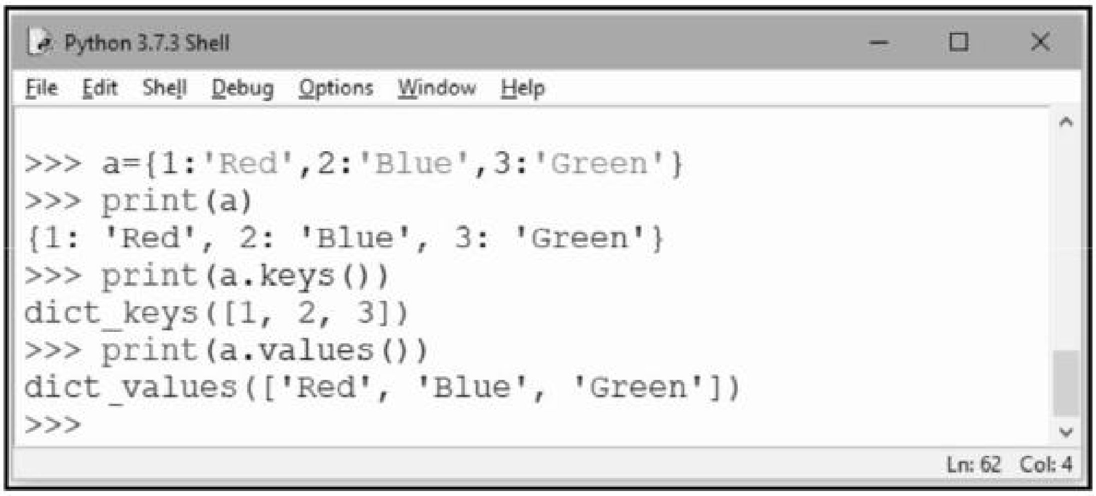
- Tuples are immutable or read-only. That means we can not modify the size of tuple or we cannot change the value of items of the tuple.
- Here are examples of tuples -



```
>>> #Tuple of only integer
>>> a=(10,20,30,40,50)
>>> print(a)
(10, 20, 30, 40, 50)
>>> #Tuple of both integers and strings
>>> b=("one",1,2,"two","three",3)
>>> print(b)
('one', 1, 2, 'two', 'three', 3)
>>> print(b[3])
two
>>> |
```

(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 –



```
>>> a={1:'Red',2:'Blue',3:'Green'}
>>> print(a)
{1: 'Red', 2: 'Blue', 3: 'Green'}
>>> print(a.keys())
dict_keys([1, 2, 3])
>>> print(a.values())
dict_values(['Red', 'Blue', 'Green'])
>>>
```

Review Question

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

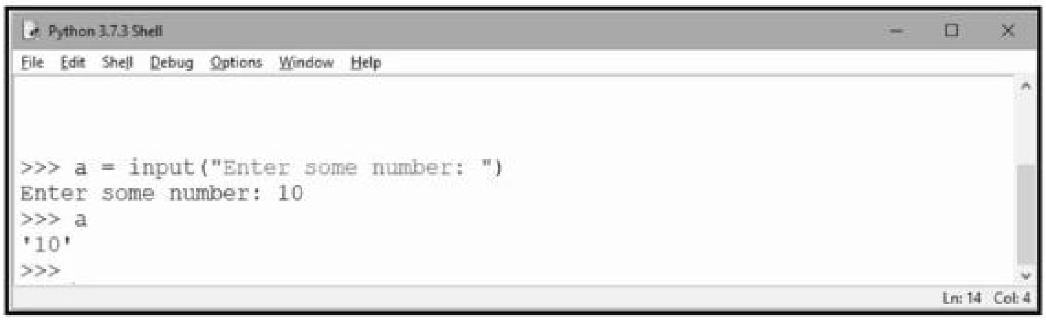
1.6 Input through Keyboard

- 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*

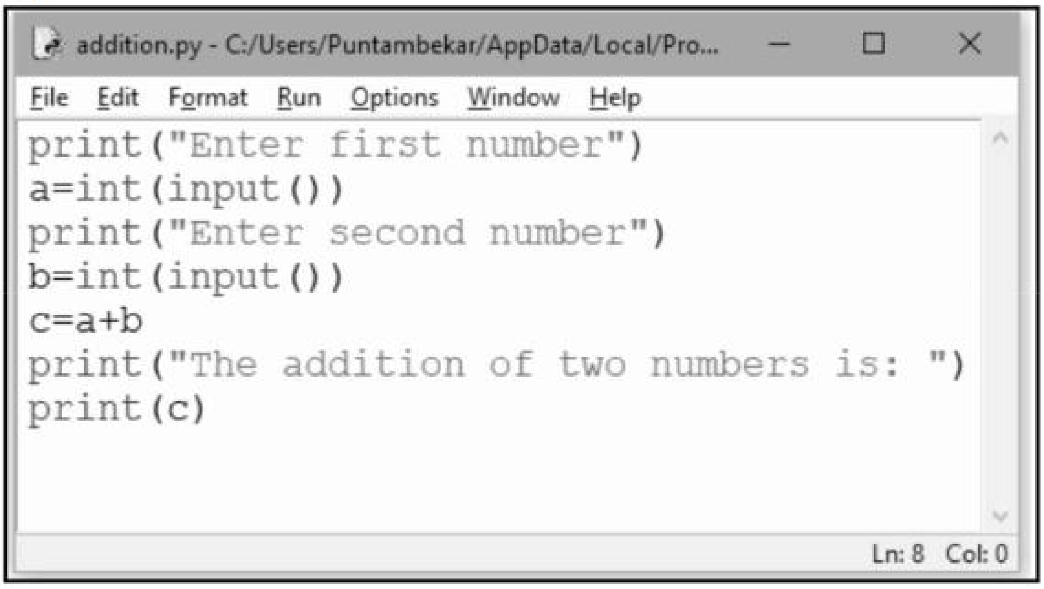


```
>>> a = input("Enter some number: ")
Enter some number: 10
>>> a
'10'
>>>
```

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

Solution :

addition.py

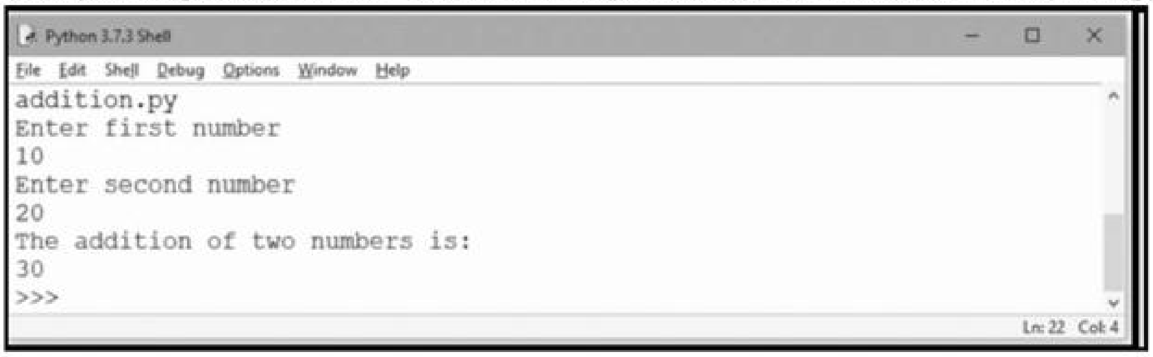

addition.py - C:/Users/Puntambekar/AppData/Local/Pro...
File Edit Format Run Options Window Help
Ln: 8 Col: 0

```
print("Enter first number")
a=int(input())
print("Enter second number")
b=int(input())
c=a+b
print("The addition of two numbers is: ")
print(c)
```

f 184 — | + 70%

Output

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



```
addition.py
Enter first number
10
Enter second number
20
The addition of two numbers is:
30
>>>
```

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.*

of 184 — + 70%

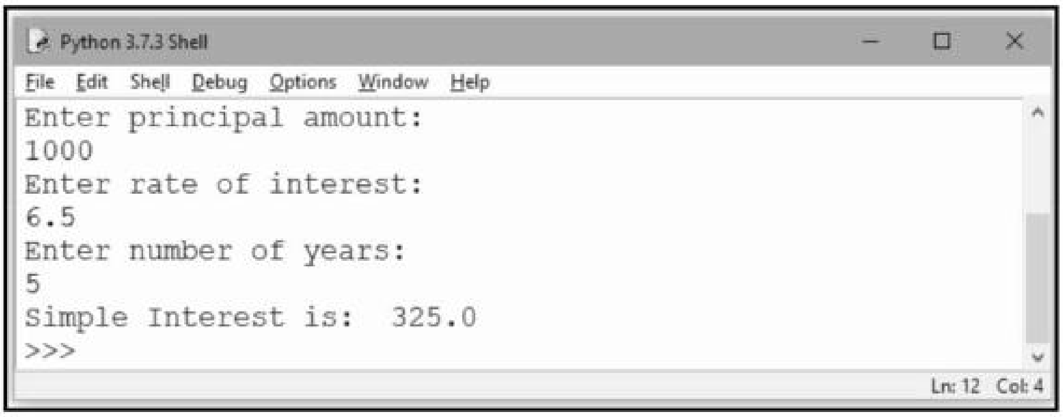
Solution :

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



```
Enter principal amount:
1000
Enter rate of interest:
6.5
Enter number of years:
5
Simple Interest is:  325.0
>>>
```

❑❑❑

UNIT - II

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.

f 184 − | + 70%

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

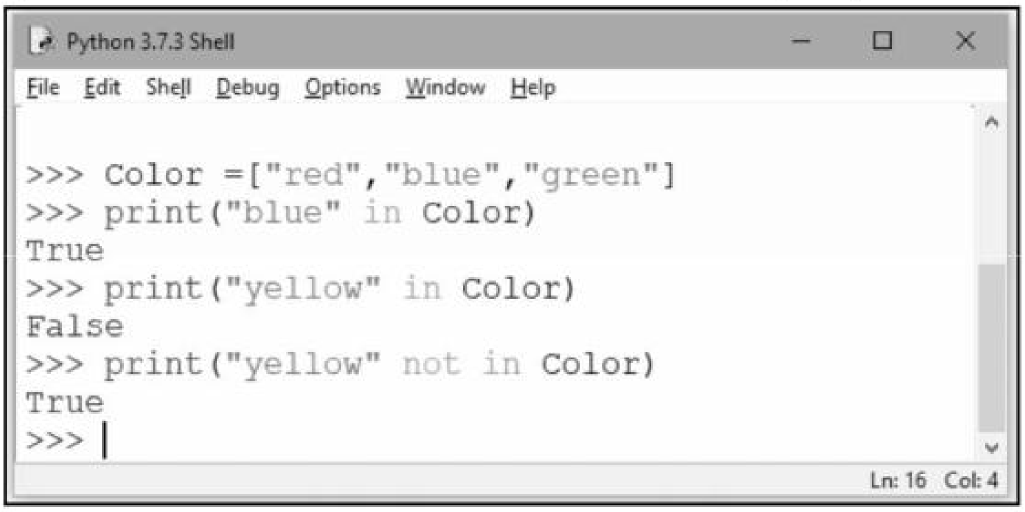
There are two types of membership operators – in and not in

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.

of 184 − | + 70%



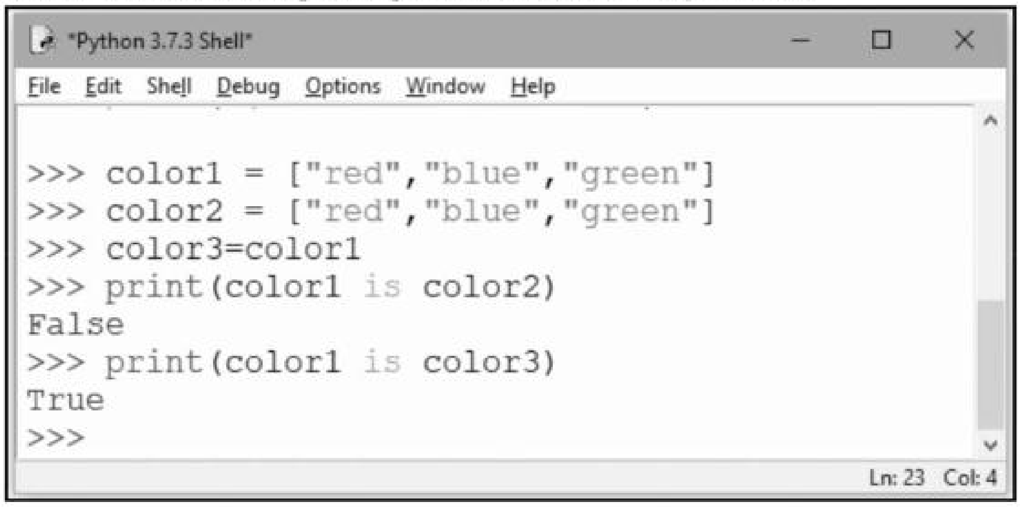
```
>>> Color =["red","blue","green"]
>>> print("blue" in Color)
True
>>> print("yellow" in Color)
False
>>> print("yellow" not in Color)
True
>>> |
```

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.



```
>>> color1 = ["red","blue","green"]
>>> color2 = ["red","blue","green"]
>>> color3=color1
>>> print(color1 is color2)
False
>>> print(color1 is color3)
True
>>>
```

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
```

f 184 − | + 70%

2.1.11 Programs using Operators

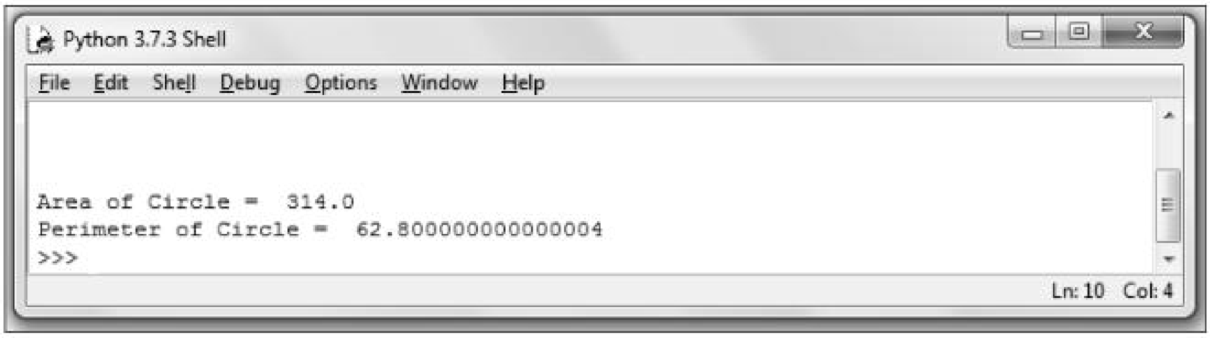
Example 2.1.1 : Write a program in Python to print area and perimeter of a circle.

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



```
Area of Circle =  314.0
Perimeter of Circle =  62.800000000000004
>>>
```

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

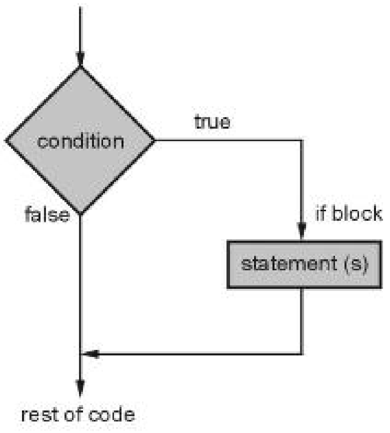
2.3 Conditional Statements

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.



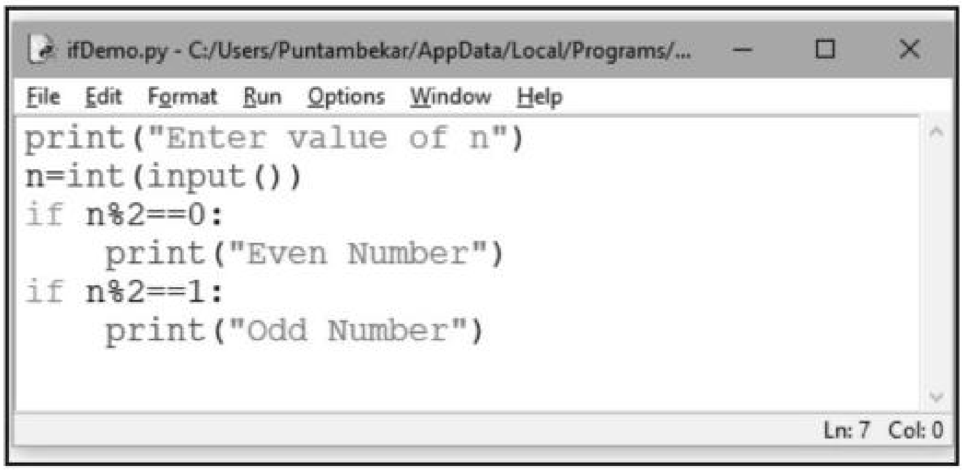
Syntax :

```
if condition:
        statement
```

Example:

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

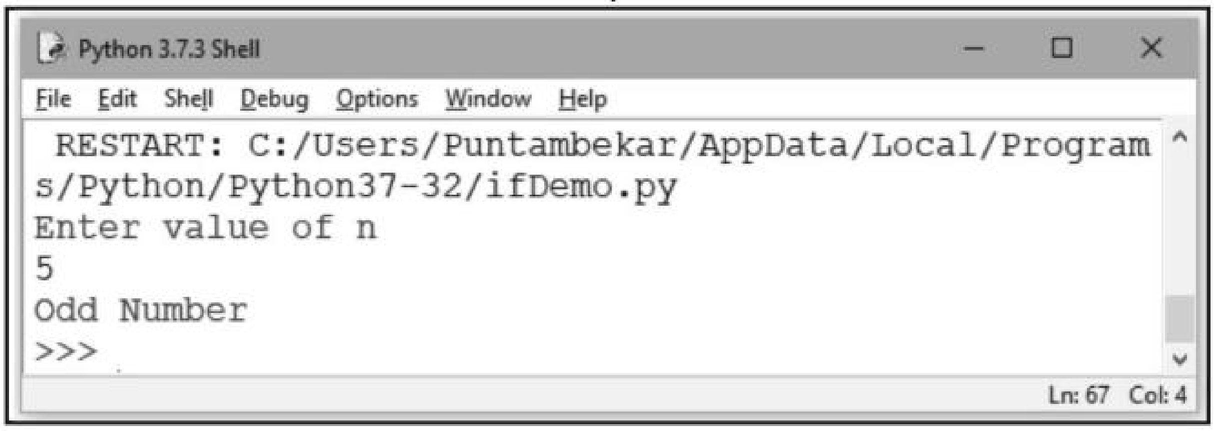
Example 2.3.1 : Write a Python program to check whether given number is even or odd.
Solution :


ifDemo.py - C:/Users/Puntambekar/AppData/Local/Programs/...
File Edit Format Run Options Window Help
Ln: 7 Col: 0

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

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

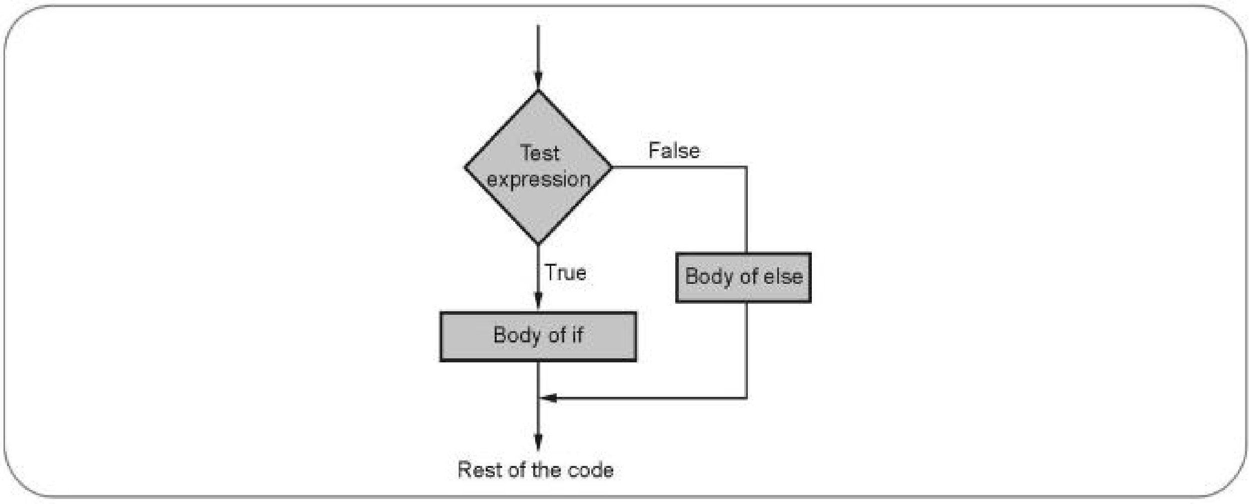
Output



```
 RESTART: C:/Users/Puntambekar/AppData/Local/Program
s/Python/Python37-32/ifDemo.py
Enter value of n
5
Odd Number
>>>
```

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
```

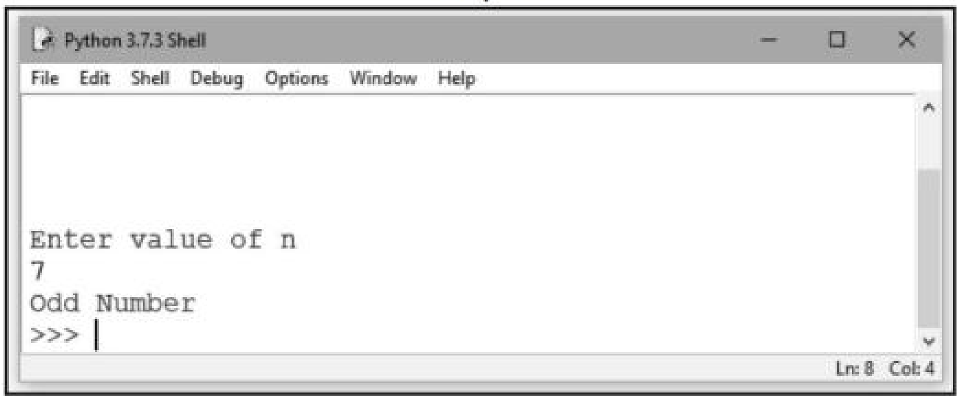
- If the condition is true, then the if-block is executed. Otherwise, the else-block is executed.

ifelseDemo.py

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

f 184 − + 70%

Output



```
Enter value of n
7
Odd Number
>>>
```

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:                       ├ Nested if-else
      if a>b:               ┘
      print("a is greater than b")
```

Output

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

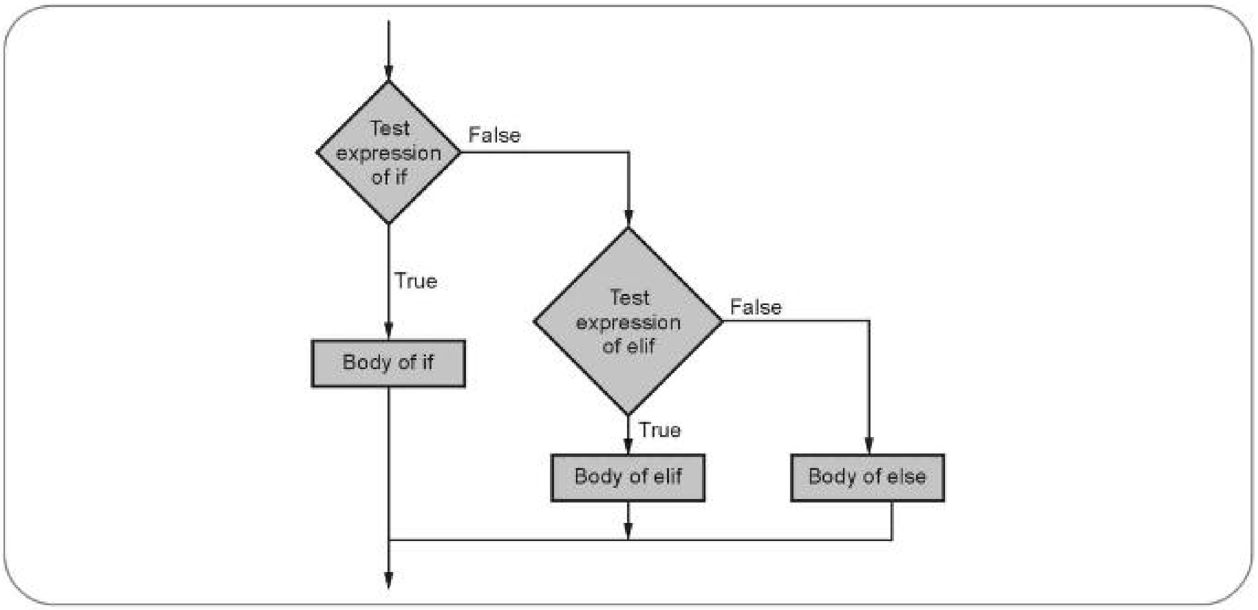
4. If-elif-else Statement

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

f 184 − + 70%

```
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")
```

f 184 − + 70%

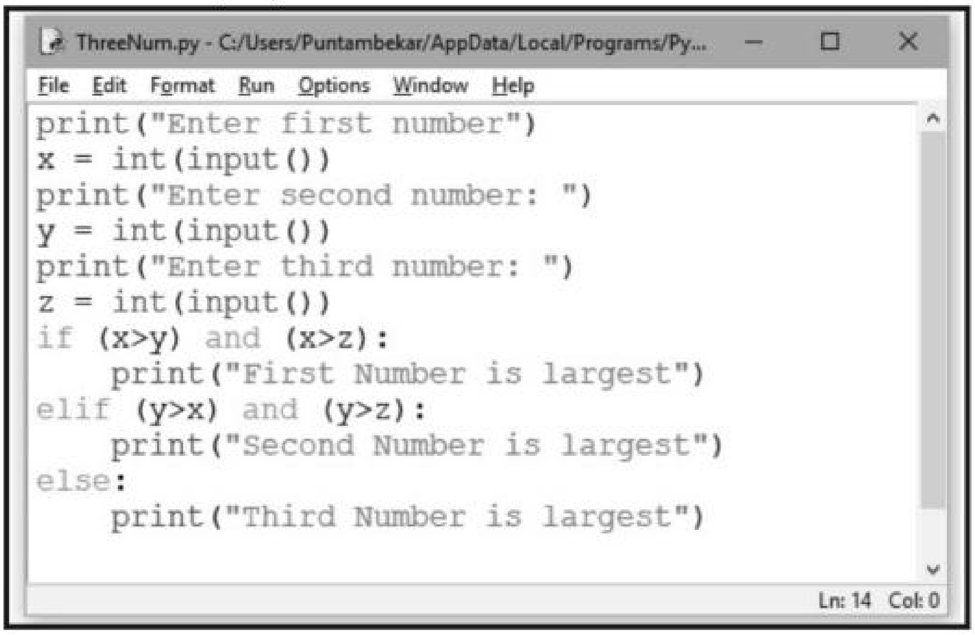
Output

```
Enter your marks
45
Grade : Pass Class
```

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

Solution :

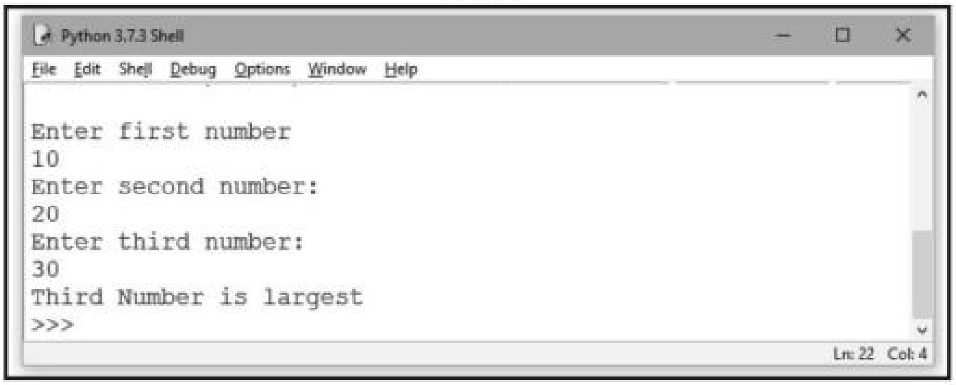
```
print("Enter first number")
x = int(input())
print("Enter second number:")
y = int(input())
print("Enter third number:")
z=int(input())
if (x>y) and (x>z):
    print("First number is largest")
elif (y>x) and (y>z):
    print("Second number is largest")
else:
    print("Third number is largest")
```



ThreeNum.py - C:/Users/Puntambekar/AppData/Local/Programs/Py...
File Edit Format Run Options Window Help
Ln: 14 Col: 0

```
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



```
Enter first number
10
Enter second number:
20
Enter third number:
30
Third Number is largest
>>>
```

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.

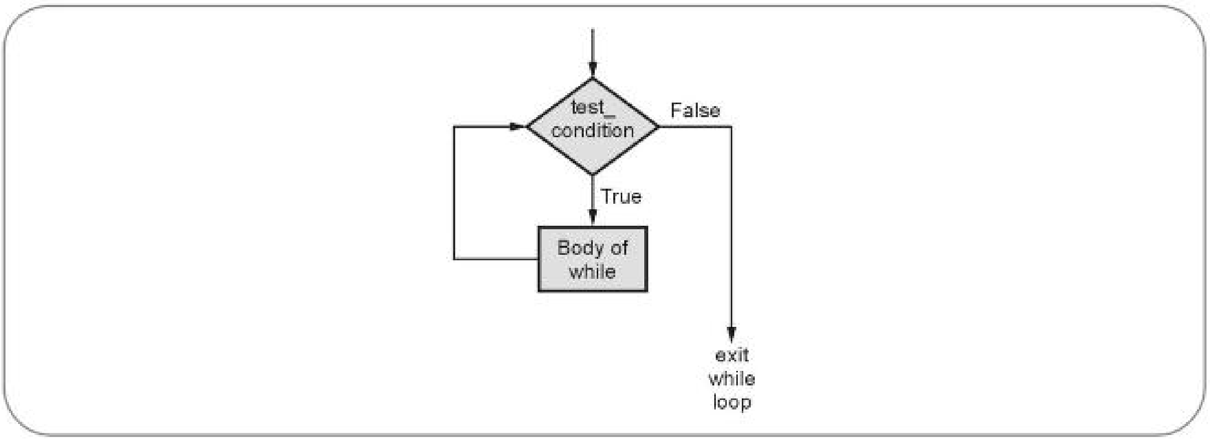
2.4.1 While Loop

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

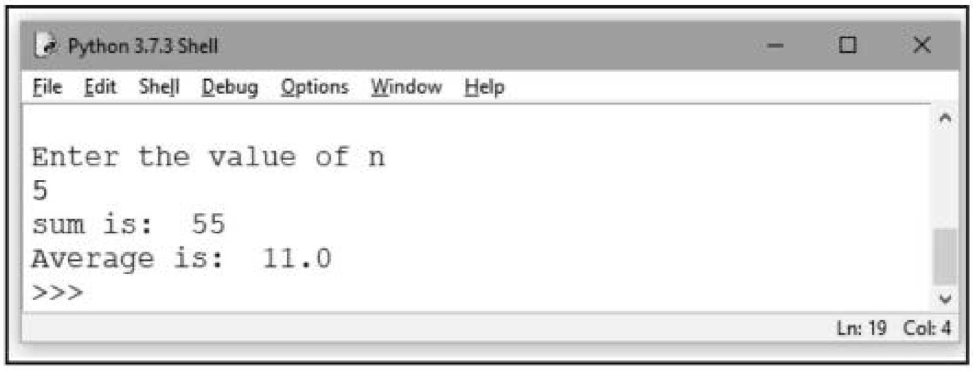
whileDemo.py

```
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.



```
Enter the value of n
5
sum is:  55
Average is:  11.0
>>>
```

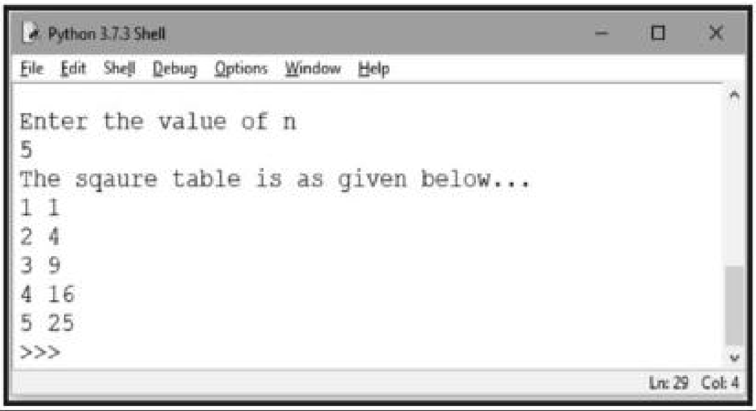
Example 2.4.2 : Write a python program to display square of n numbers using while loop.

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



```
Enter the value of n
5
The sqaure table is as given below...
1 1
2 4
3 9
4 16
5 25
>>>
```

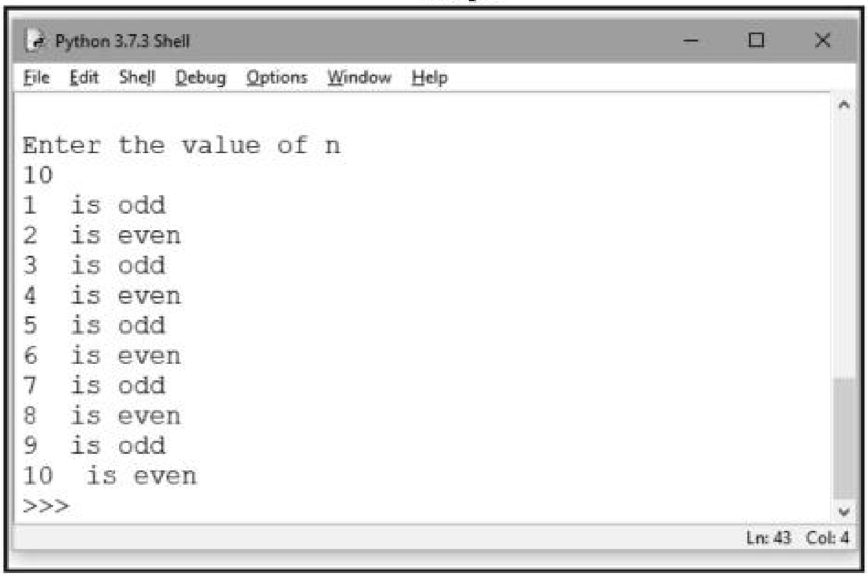
Example 2.4.3 : Write a python program for displaying even or odd numbers between 1 to n.

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



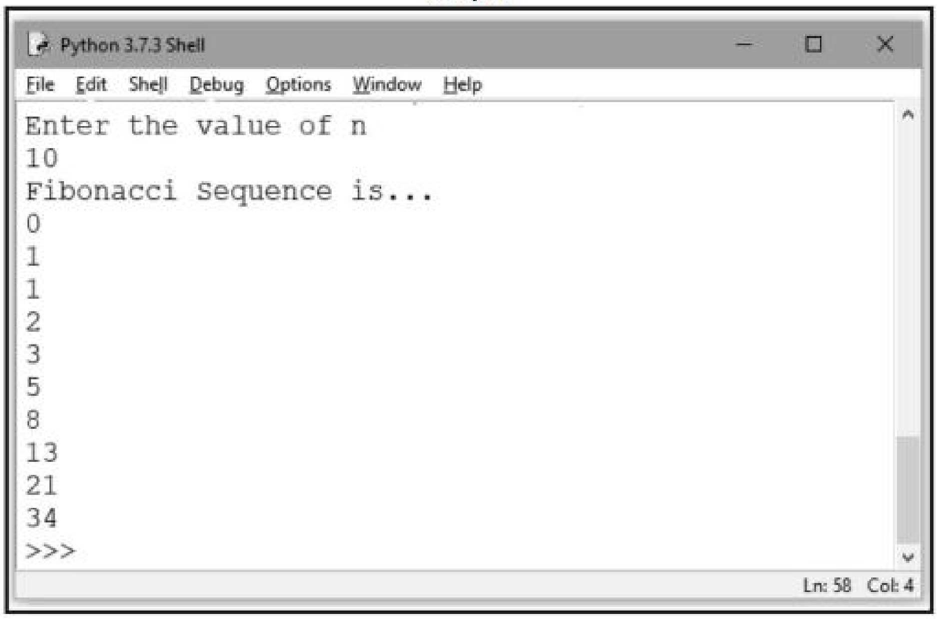
```
Enter the value of n
10
1  is odd
2  is even
3  is odd
4  is even
5  is odd
6  is even
7  is odd
8  is even
9  is odd
10  is even
>>>
```

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



```
Enter the value of n
10
Fibonacci Sequence is...
0
1
1
2
3
5
8
13
21
34
>>>
```

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")
```

of 184 − + 70%

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
```

f 184 70%

```
# 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

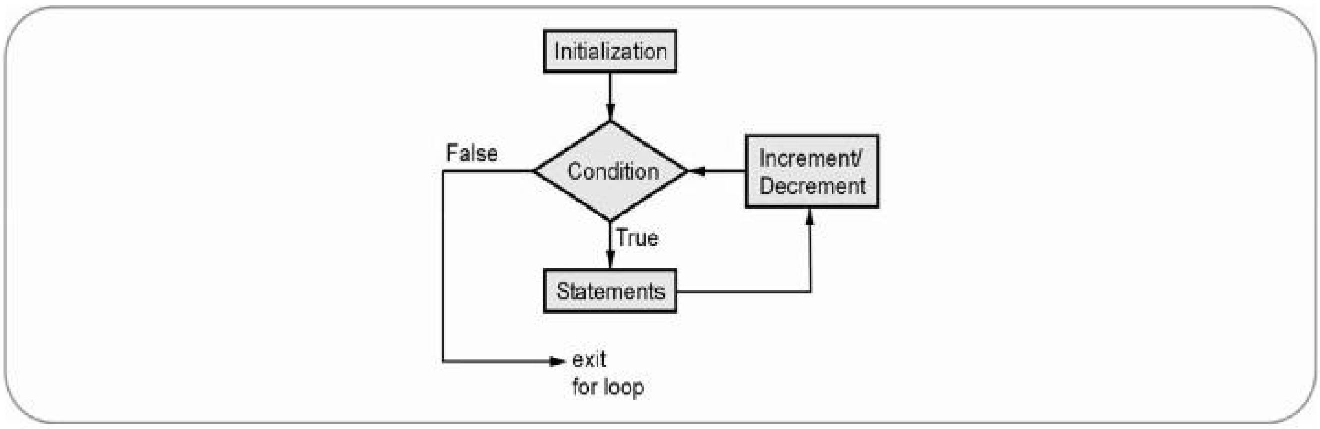
- The for loop is another popular way of using iteration. The syntax of for loop is
- **Syntax**

```
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 –



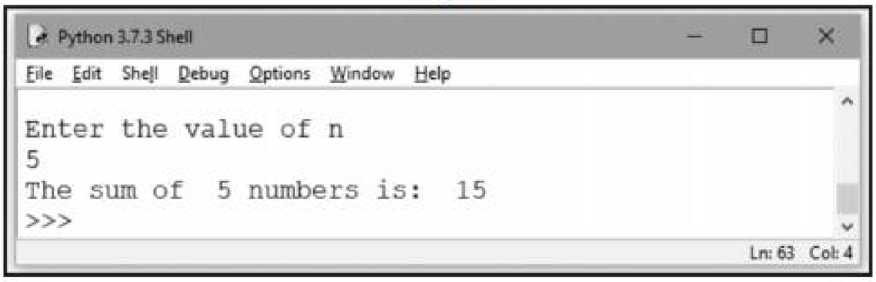
of 184 − + 70%

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)
```

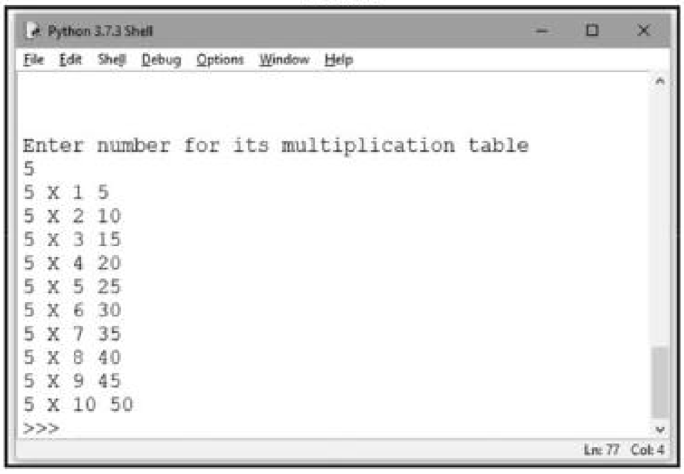
Output



```
Enter the value of n
5
The sum of  5 numbers is:  15
>>>
```

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

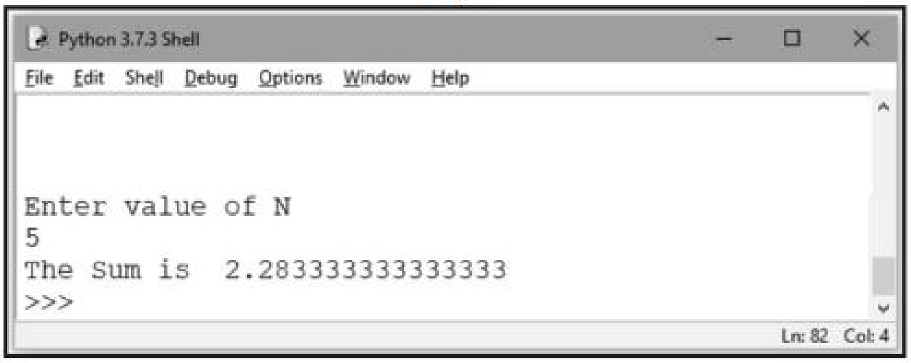


```
Enter number for its multiplication table
5
5 X 1 5
5 X 2 10
5 X 3 15
5 X 4 20
5 X 5 25
5 X 6 30
5 X 7 35
5 X 8 40
5 X 9 45
5 X 10 50
>>>
```

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



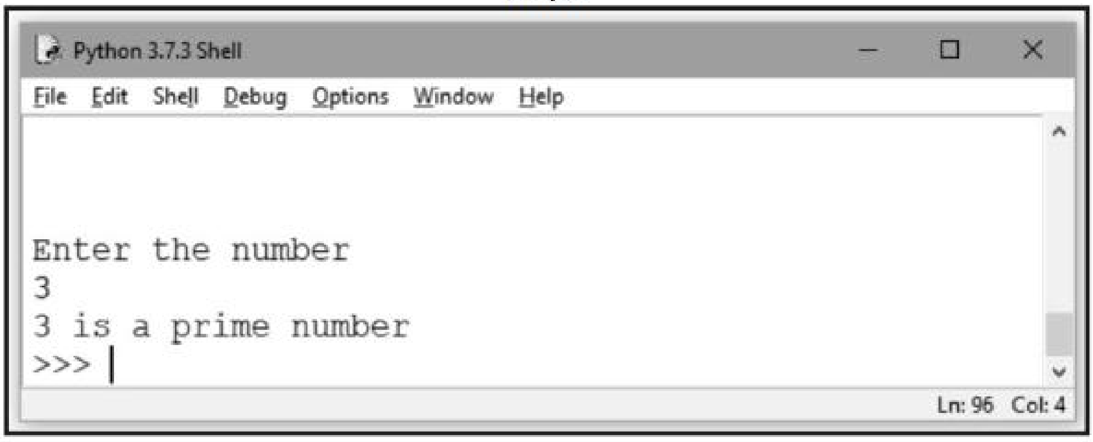
```
Enter value of N
5
The Sum is  2.283333333333333
>>>
```

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



```
Enter the number
3
3 is a prime number
>>> 
```

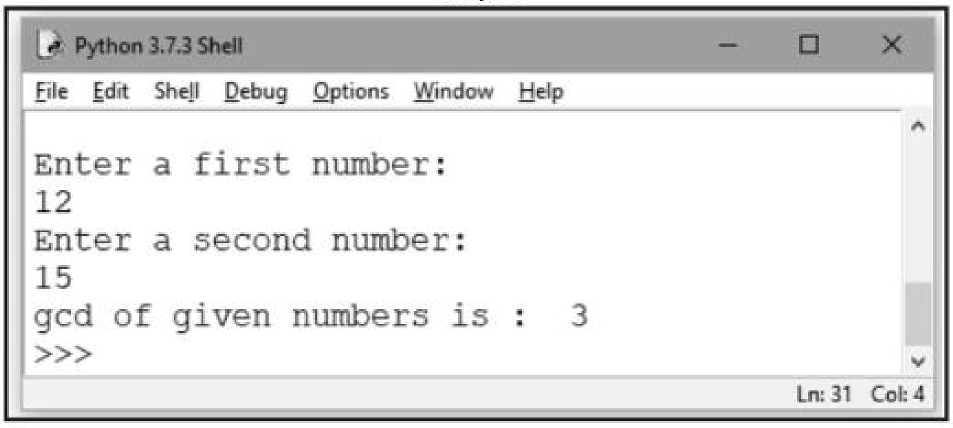
Example 2.4.11 : Write a python program for finding the GCD of two numbers.
Solution :

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)
```

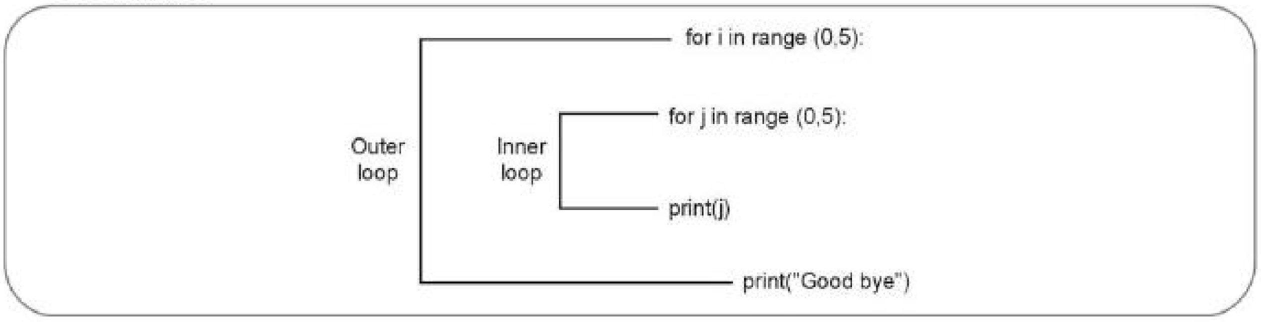
Output



```
Enter a first number:
12
Enter a second number:
15
gcd of given numbers is :  3
>>>
```

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

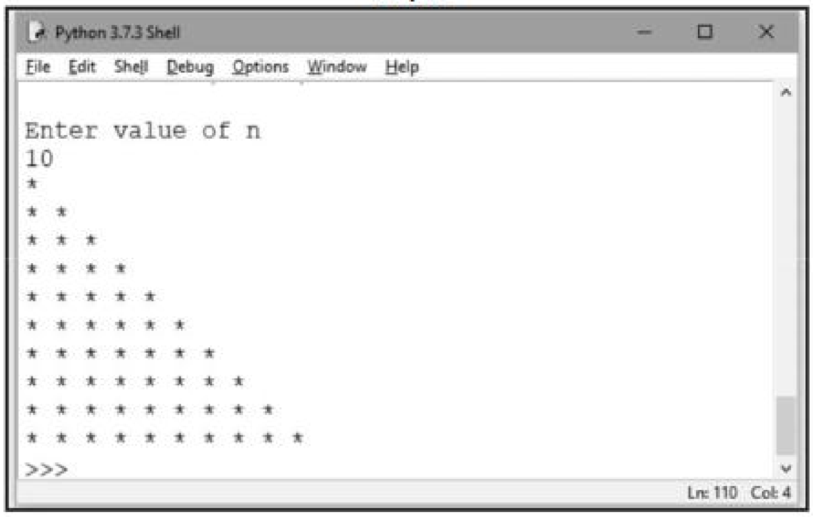
```
*
* *
* * *
* * * *
* * * * *
...
...
```

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



```
Enter value of n
10
*
* *
* * *
* * * *
* * * * *
* * * * * *
* * * * * * *
* * * * * * * *
* * * * * * * * *
* * * * * * * * * *
>>>
```

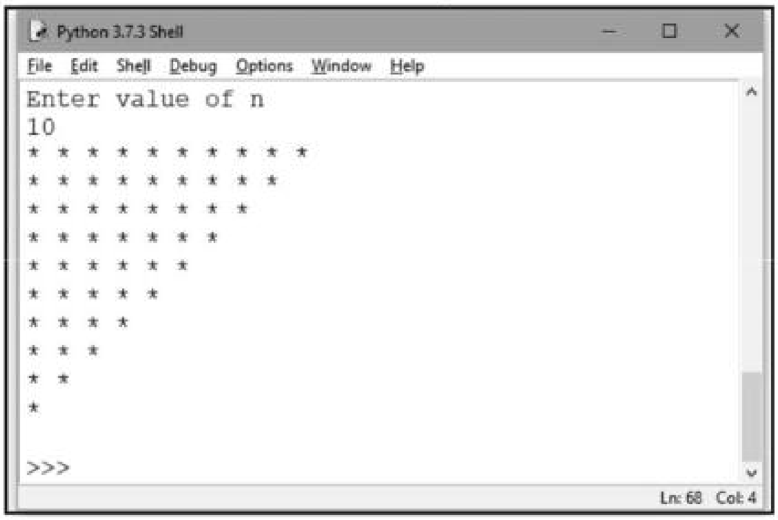
Example 2.4.13 : Write a python program to display a star pattern as

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

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



```
Enter value of n
10
* * * * * * * * * *
* * * * * * * * *
* * * * * * * *
* * * * * * *
* * * * * *
* * * * *
* * * *
* * *
* *
*


>>>
```

f 184 − + 70%

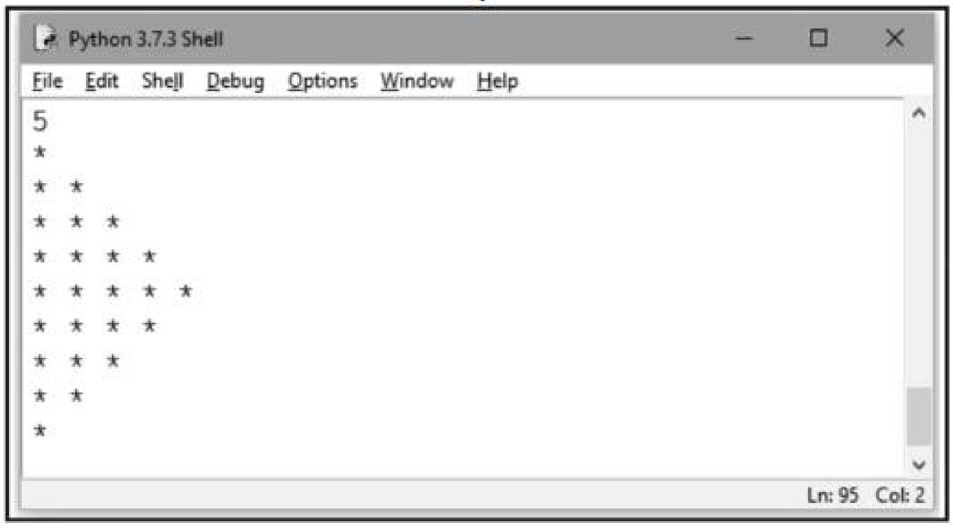
Example 2.4.14 : Write a python program to display the star pattern as

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

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



```
5
*
*  *
*  *  *
*  *  *  *
*  *  *  *  *
*  *  *  *
*  *  *
*  *
*
```

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
```

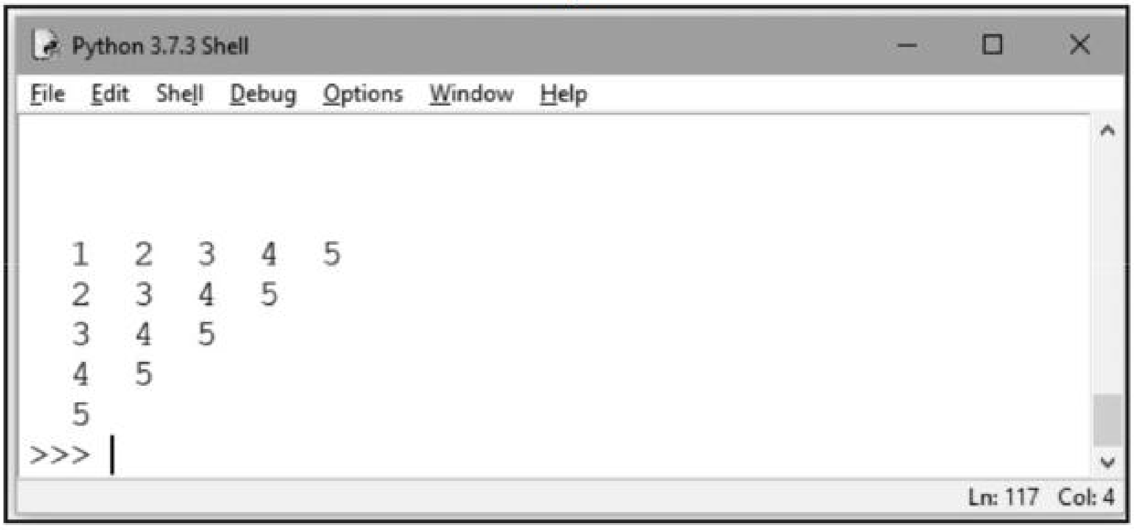
Solution :

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



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

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

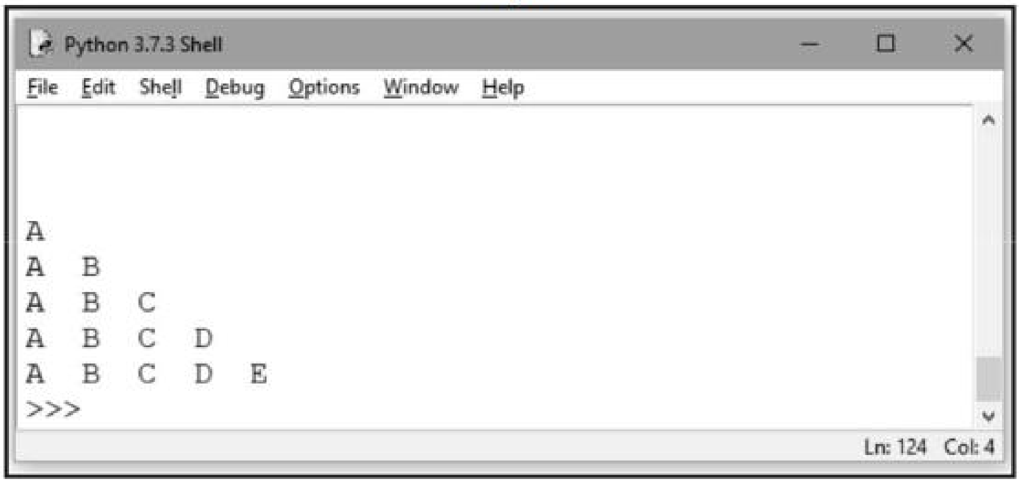
```
A
A B
A B C
A B C D
A B C D E
```

Solution :

Pattern3.py

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

Output



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

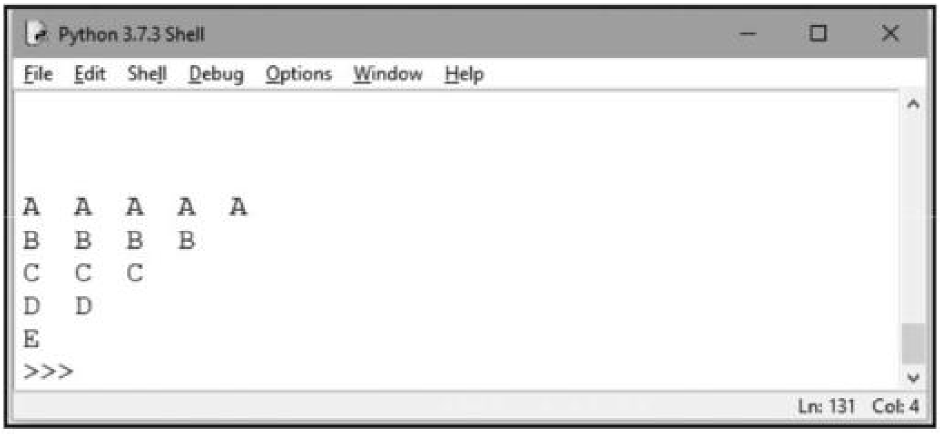
Example 2.4.17 : Write a python program to print the pattern for alphabets.

```
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();
```



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

Review Question

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

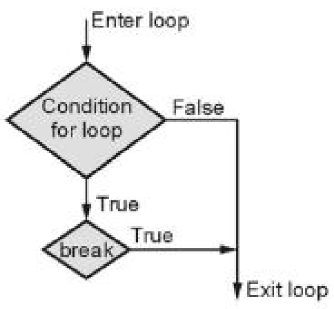
2.5 Loop Manipulation

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



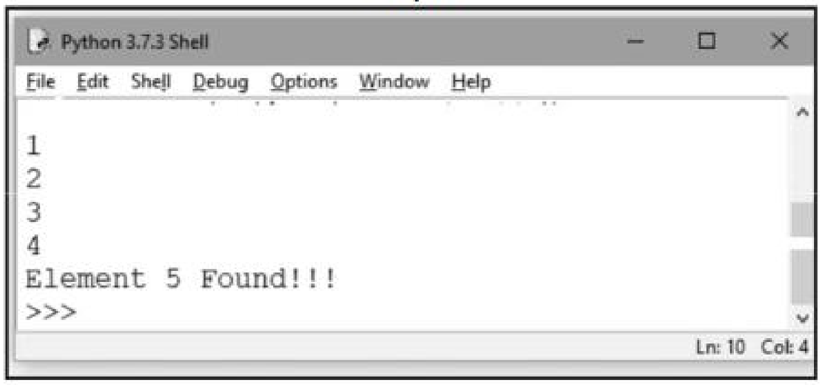
- **Syntax**

  break

- **For example**

```
for i in range(1,11):
   if i==5:
        print("Element {} Found!!!".format(i))
        break
    print(i)
```

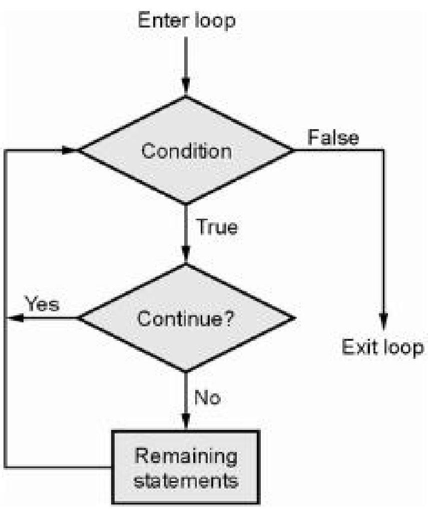
Output



```
1
2
3
4
Element 5 Found!!!
>>>
```

(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**

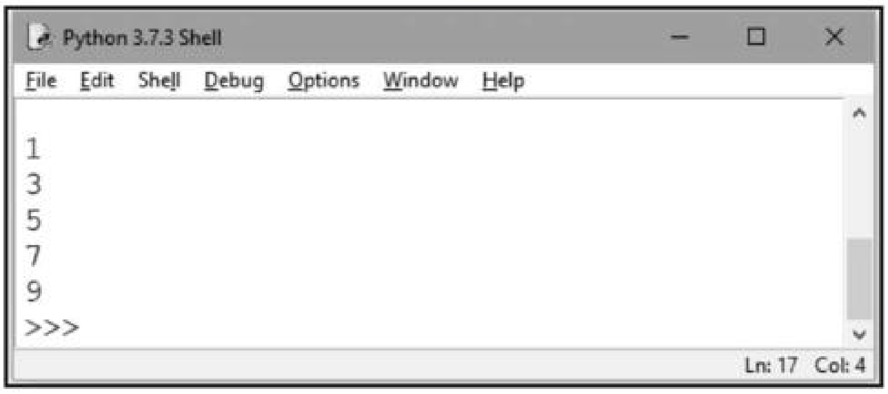
```
continue
```

- **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



```
1
3
5
7
9
>>>
```

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

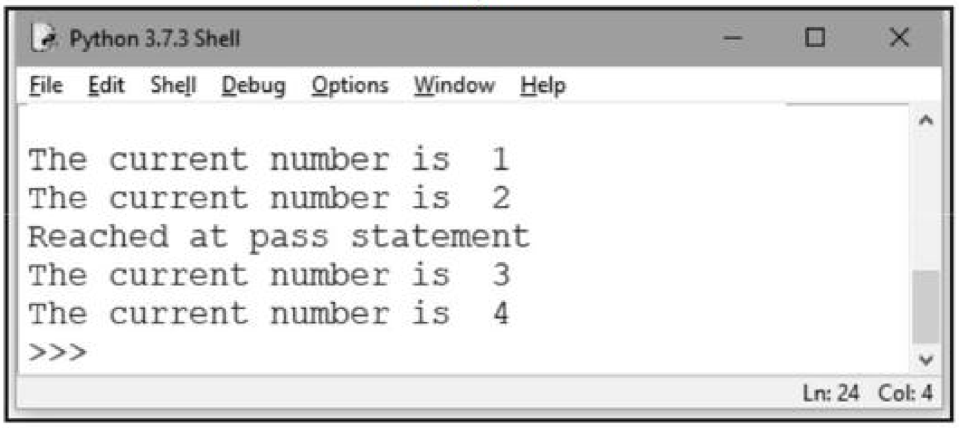
- The **pass** statement is used when we do not want to execute any statement. Thus the desired statements can be bypassed.
- **Syntax**

```
Pass
```

- **Example**

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

Output



```
The current number is  1
The current number is  2
Reached at pass statement
The current number is  3
The current number is  4
>>>
```

(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.*

❑❑❑

Notes

f 184 — | + 70%

UNIT - III

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]
>>>
```

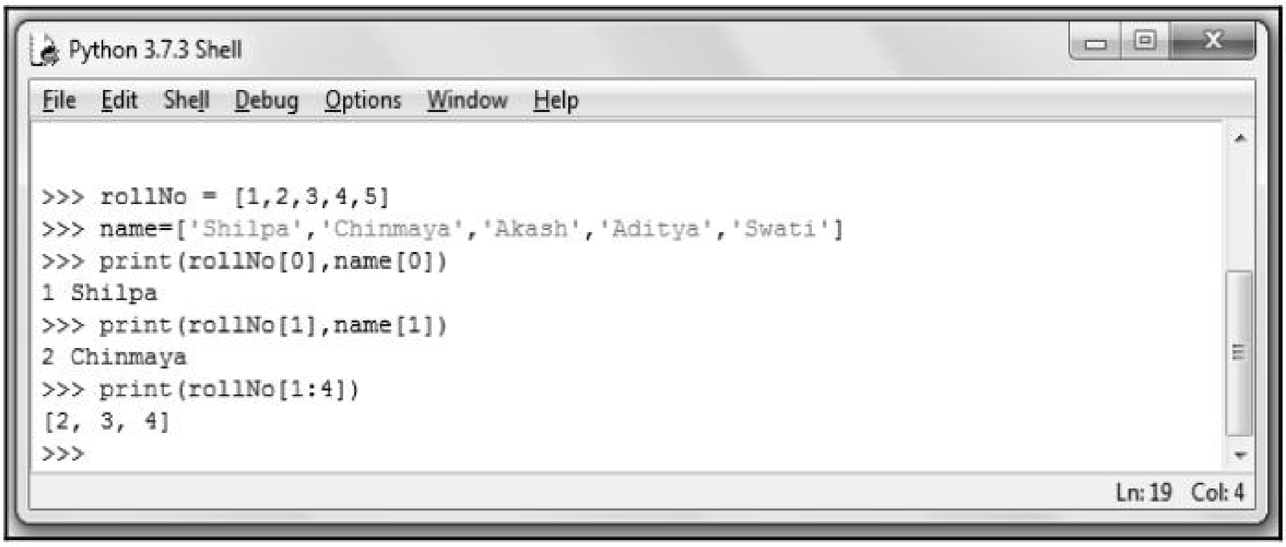
Here a and b are the lists data structures.

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 -



```
>>> 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]
>>>
```

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)):
```

of 184 − + 70%

```
        a[i]=a[i]+1   #incremented each number by one
>>> a
[11, 21, 31, 41]
>>>
```

(ii) Using enumerate() function

Using **enumerate()** function we can print both index and item of the list.

Example 3.1.1 : Write a program to iterate through list using enumerate() function.

Solution :

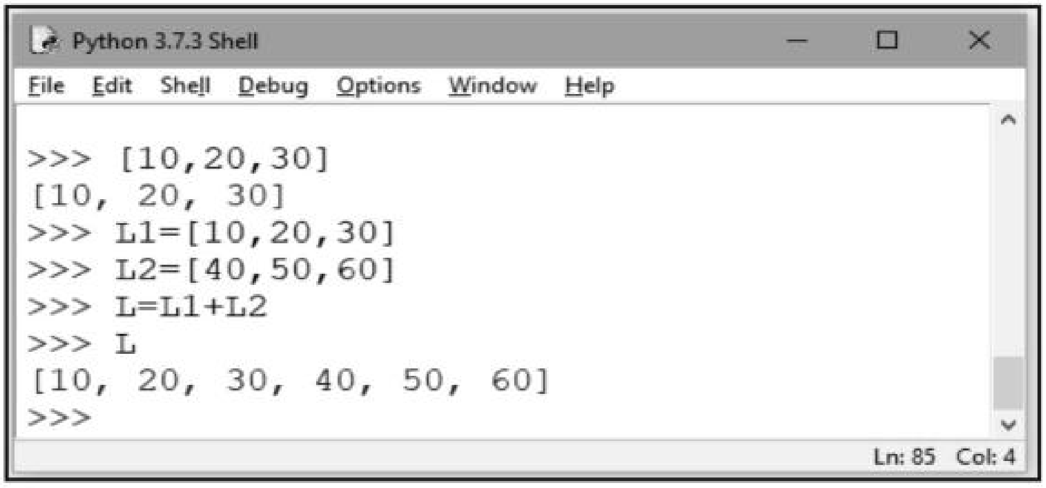
```
myList =[10,20,30]
for item in enumerate(myList):
   print(item)
```

Output

```
(0, 10)
(1, 20)
(2, 30)
```

(2) Concatenation using +

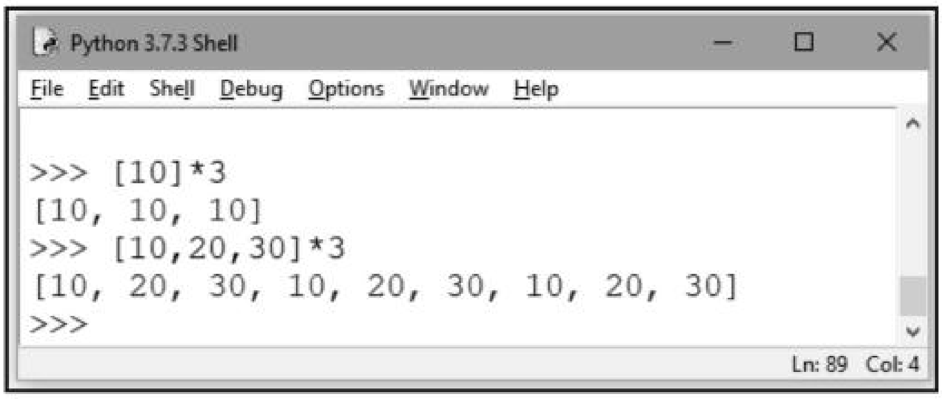
The two lists can be created and can be joined using + operator. Following screenshot illustrates it



```
>>> [10,20,30]
[10, 20, 30]
>>> L1=[10,20,30]
>>> L2=[40,50,60]
>>> L=L1+L2
>>> L
[10, 20, 30, 40, 50, 60]
>>>
```

(3) Repetition using *

The * is used to repeat the list for number of times. Following screenshot illustrates it.

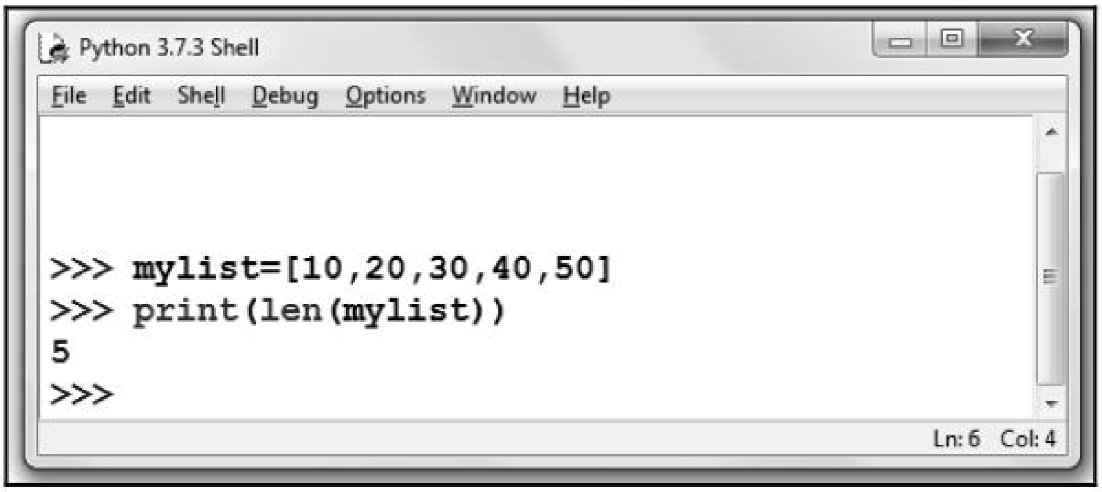


```
>>> [10]*3
[10, 10, 10]
>>> [10,20,30]*3
[10, 20, 30, 10, 20, 30, 10, 20, 30]
>>>
```

f 184 − | + 70%

(4) Finding Length of a list

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



```
>>> mylist=[10,20,30,40,50]
>>> print(len(mylist))
5
>>>
```

(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']
>>>
```

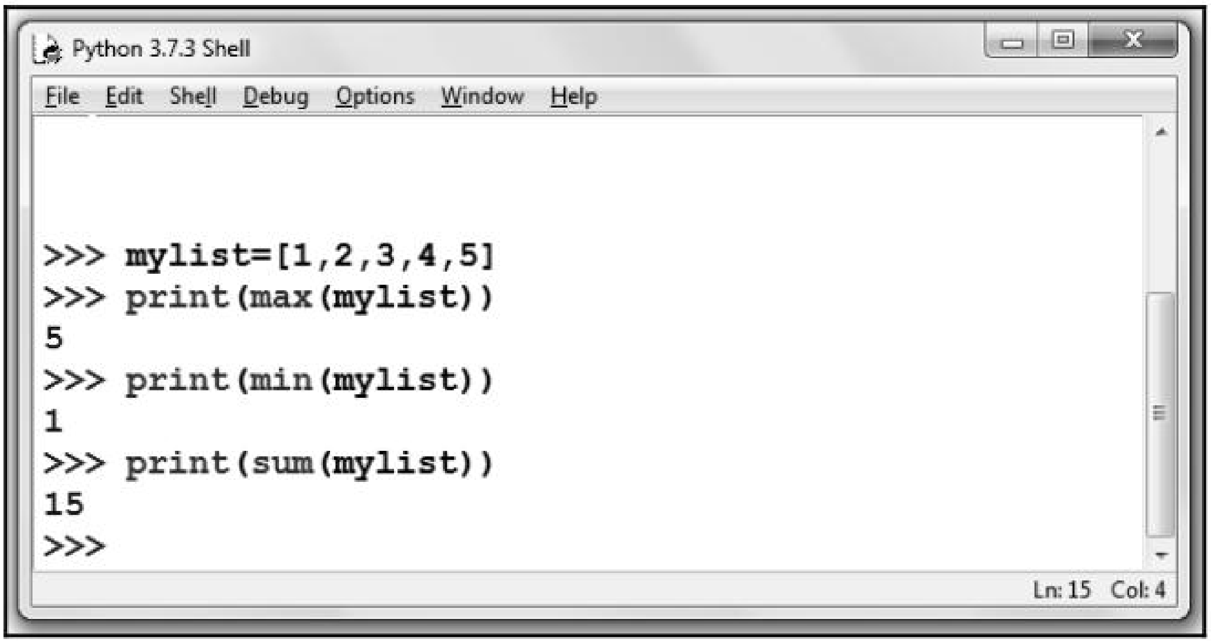
The methods append, extend and sort does not return any value. These methods simply modify the list. These are void methods.

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



```
>>> mylist=[1,2,3,4,5]
>>> print(max(mylist))
5
>>> print(min(mylist))
1
>>> print(sum(mylist))
15
>>>
```

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.

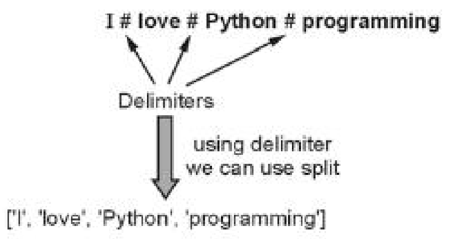
For example -

```
>>> 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']
>>>
```


I # love # Python # programming
Delimiters
using delimiter we can use split
['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)
>>>
```

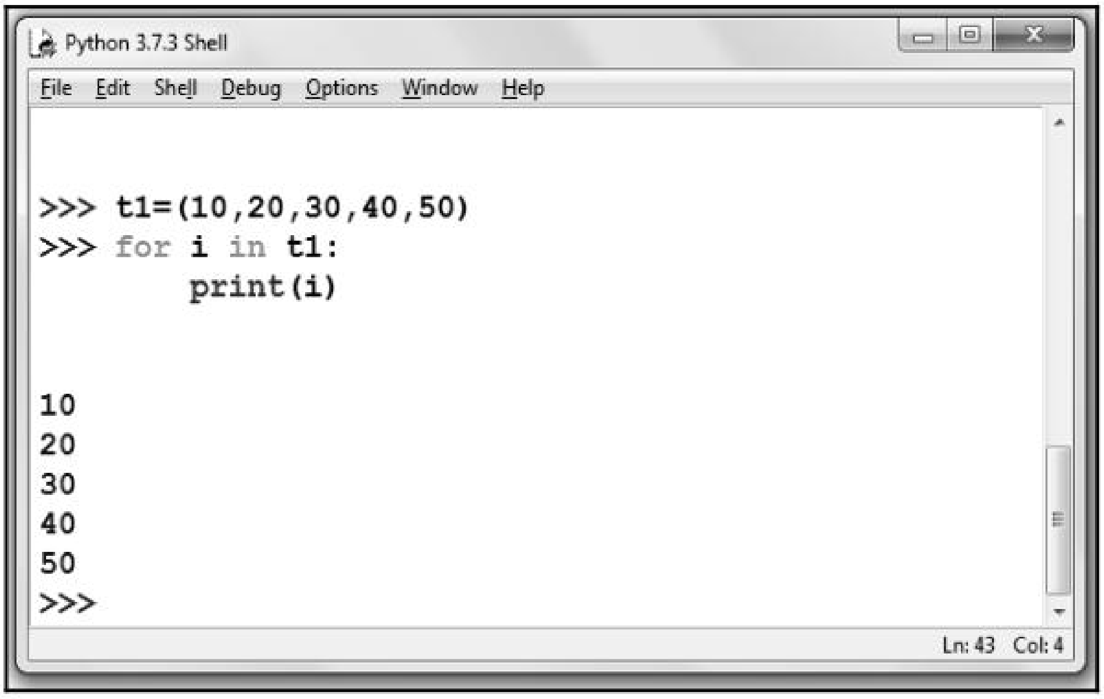
(3) Repetition : For repetition of the elements in the Tuple, we use * operator. For example

```
>>> 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



```
>>> t1=(10,20,30,40,50)
>>> for i in t1:
        print(i)


10
20
30
40
50
>>>
```

3.2.3 Built in Tuple Function

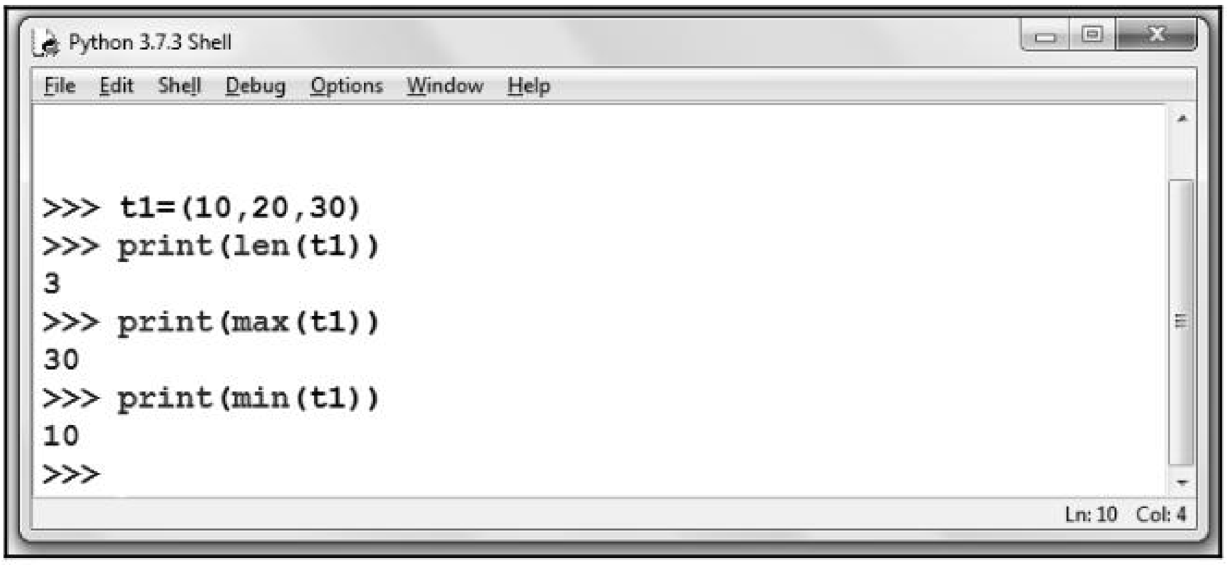
There are some useful built in Tuple functions. These functions are enlisted in the following table -

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.

f 184 − + 70%

The illustration of above functions is as shown below –

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



```
>>> t1=(10,20,30)
>>> print(len(t1))
3
>>> print(max(t1))
30
>>> print(min(t1))
10
>>>
```

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'])
```

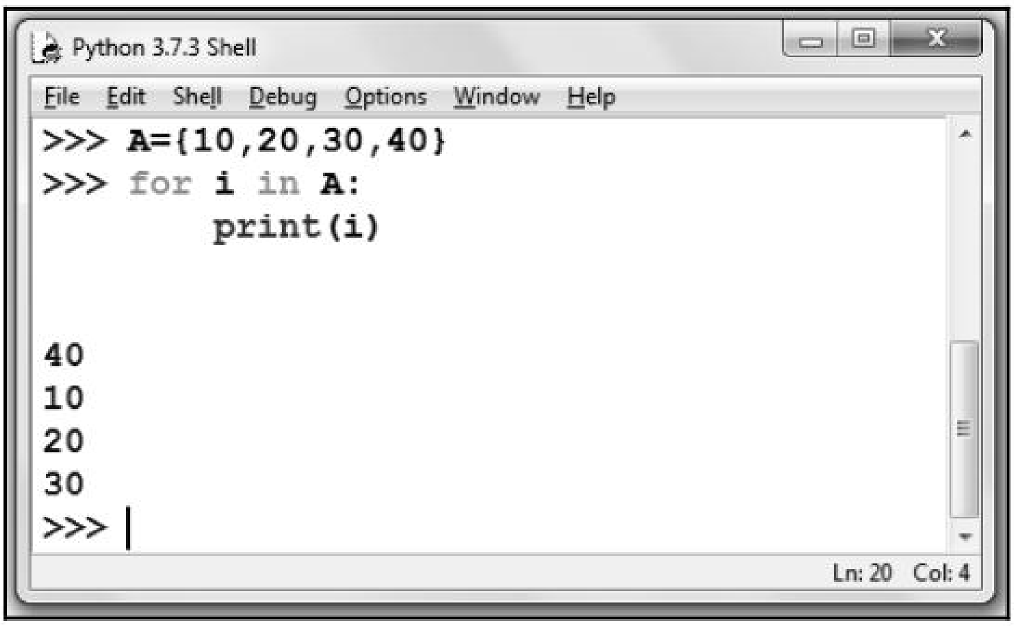
3.3.1.1 Accessing Values in Set

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

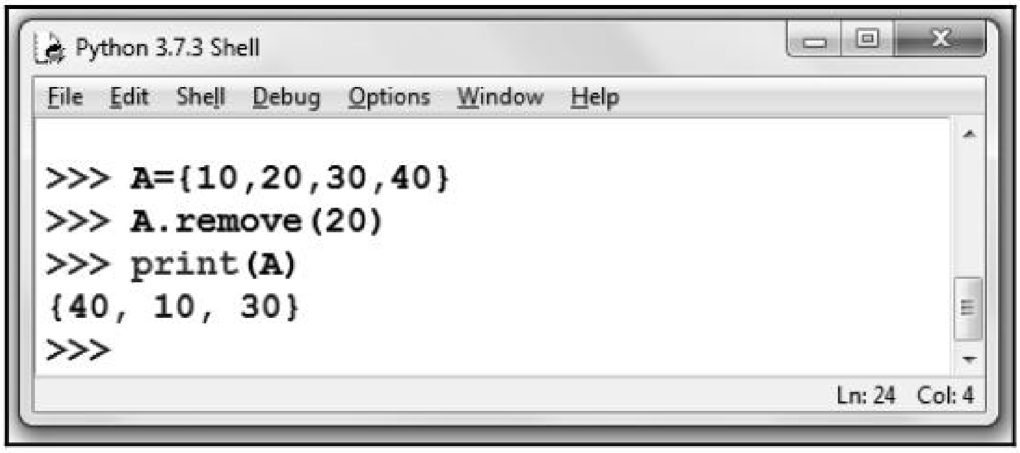


```
>>> A={10,20,30,40}
>>> for i in A:
        print(i)


40
10
20
30
>>> 
```

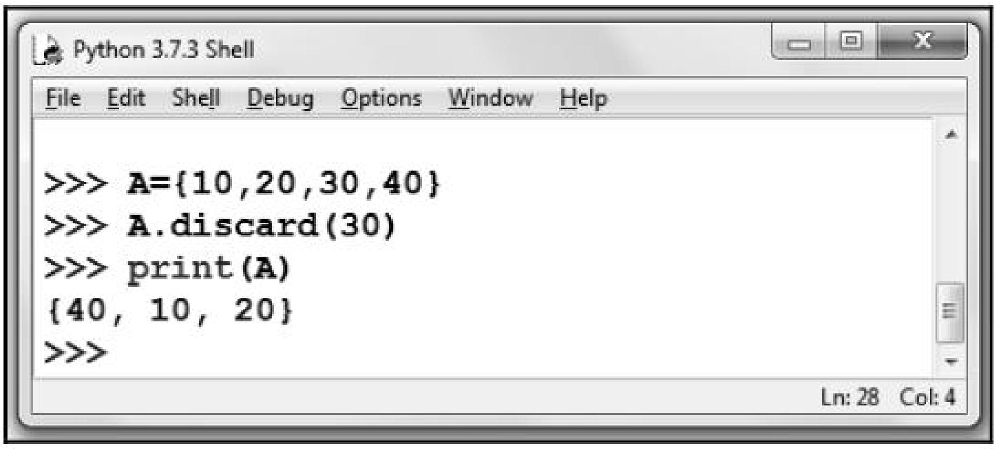
3.3.1.2 Deleting Values in Set

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



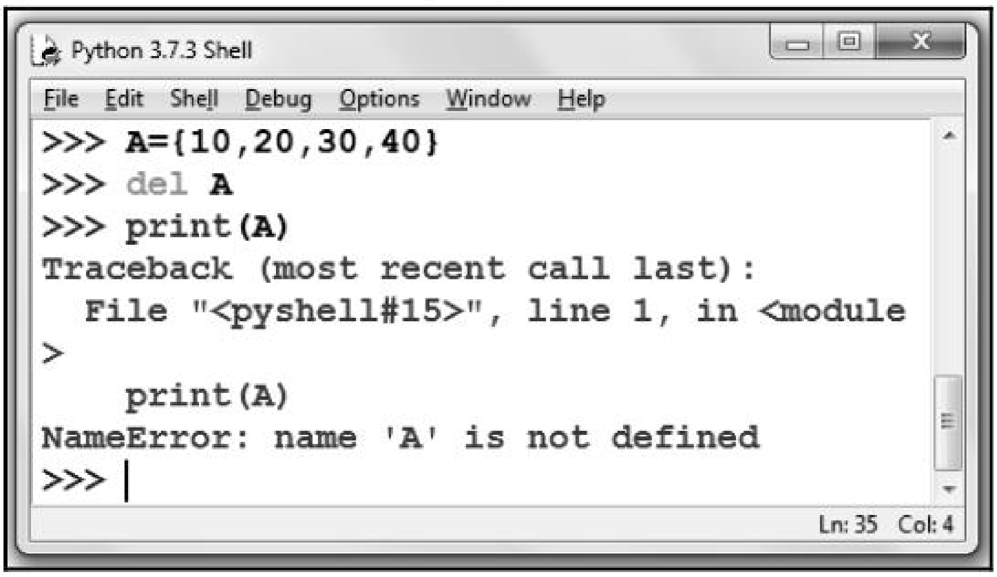
```
>>> A={10,20,30,40}
>>> A.remove(20)
>>> print(A)
{40, 10, 30}
>>>
```

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



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

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



```
>>> A={10,20,30,40}
>>> del A
>>> print(A)
Traceback (most recent call last):
  File "<pyshell#15>", line 1, in <module
>
    print(A)
NameError: name 'A' is not defined
>>>
```

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}
>>>
```

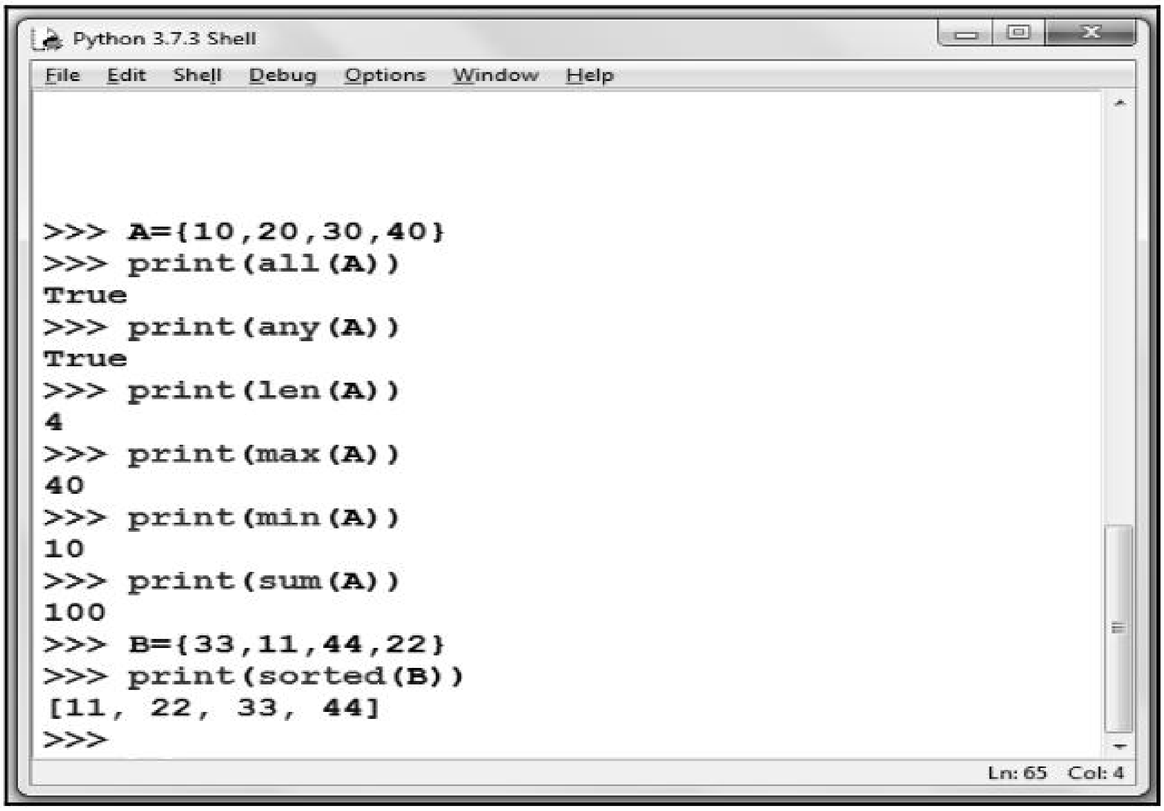
of 184 − + 70%

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 -



```
>>> A={10,20,30,40}
>>> print(all(A))
True
>>> print(any(A))
True
>>> print(len(A))
4
>>> print(max(A))
40
>>> print(min(A))
10
>>> print(sum(A))
100
>>> B={33,11,44,22}
>>> print(sorted(B))
[11, 22, 33, 44]
>>>
```

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.

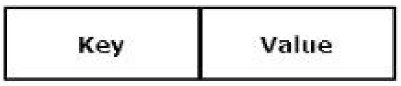


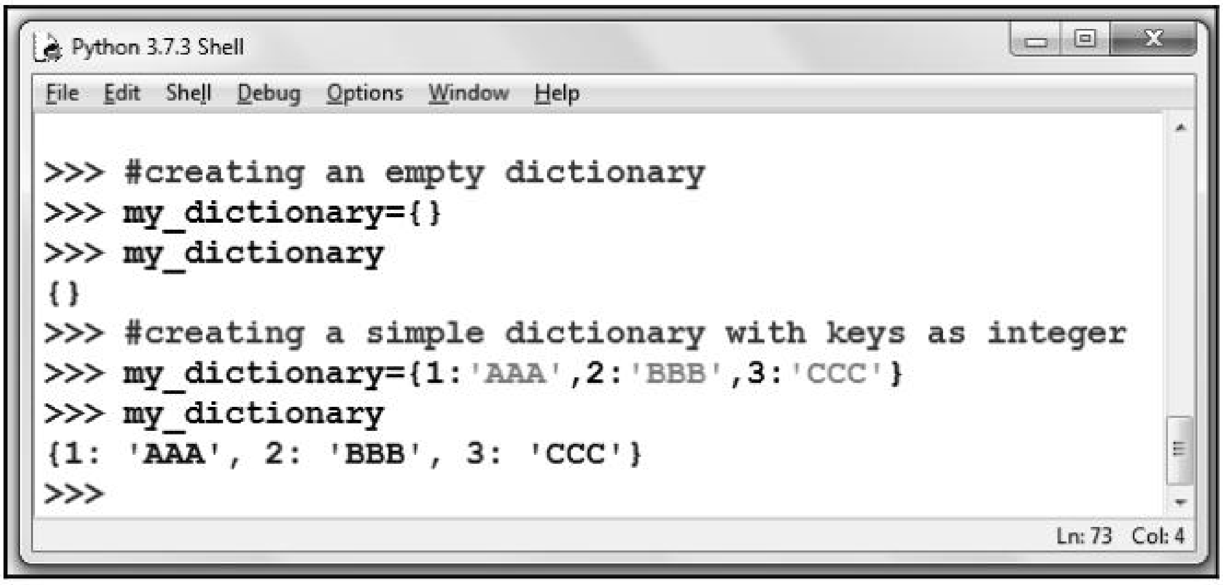
Fig. 3.4.1 Dictionary element

- 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



```
>>> #creating an empty dictionary
>>> my_dictionary={}
>>> my_dictionary
{}
>>> #creating a simple dictionary with keys as integer
>>> my_dictionary={1:'AAA',2:'BBB',3:'CCC'}
>>> my_dictionary
{1: 'AAA', 2: 'BBB', 3: 'CCC'}
>>>
```

- 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

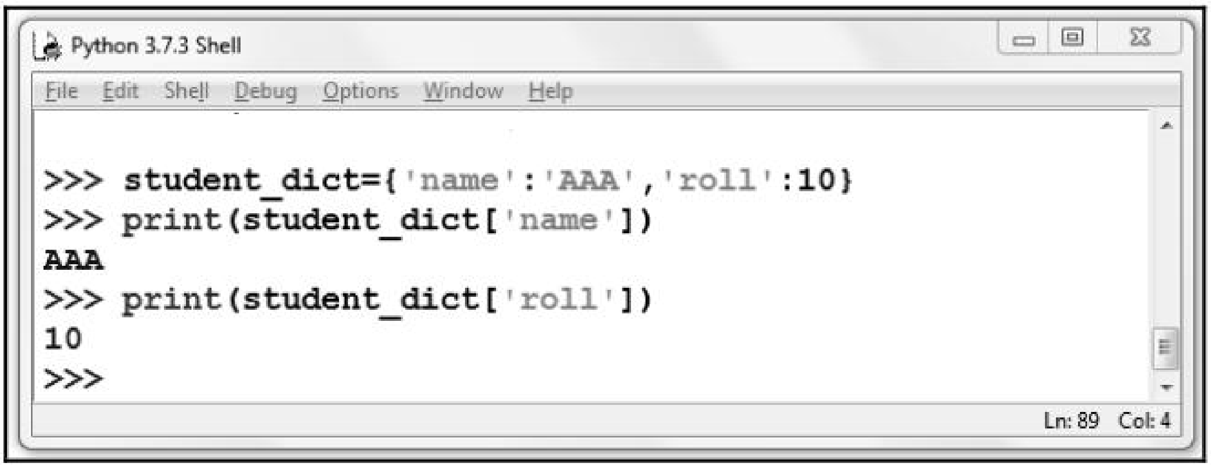
We can access the element in the dictionary using the keys. Following script illustrates it

DictionaryDemo.py

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

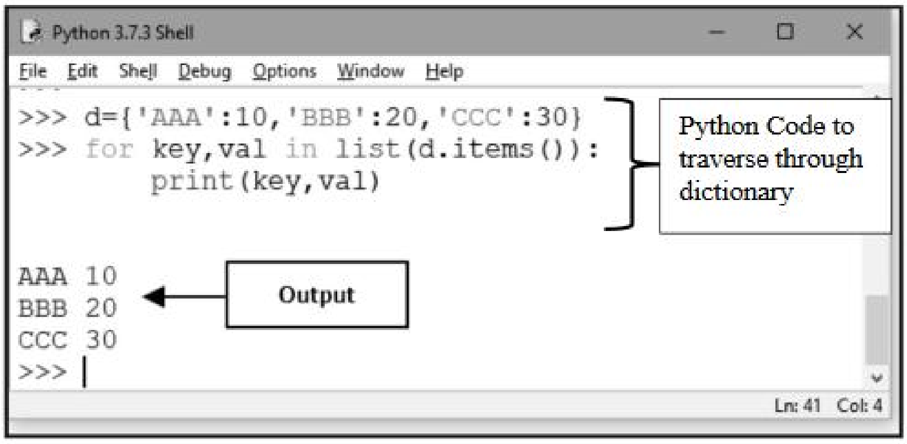
Output

```
AAA
10
```



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


Python 3.7.3 Shell
File Edit Shell Debug Options Window Help
Python Code to traverse through dictionary
Output
Ln: 41 Col: 4

```
>>> d={'AAA':10,'BBB':20,'CCC':30}
>>> for key,val in list(d.items()):
        print(key,val)


AAA 10
BBB 20
CCC 30
>>> |
```

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 -

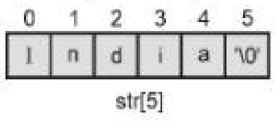


Fig. 3.5.1 String indexes

Review Question

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

3.6 String Operations

3.6.1 Finding Length of String

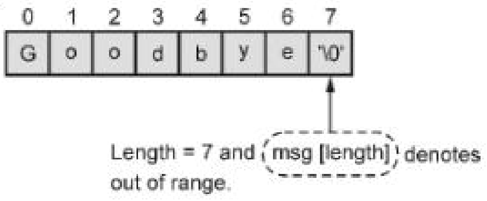
There is an in-built function to find length of the string. This is **len** function. For example

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

of 184 − + 70%

```
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 -

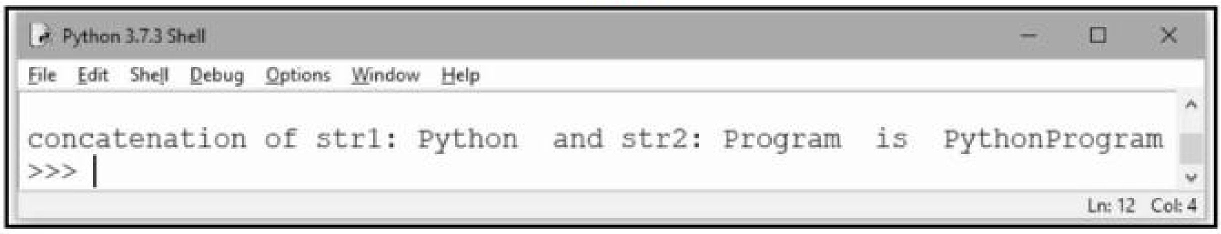
Example 3.6.1 : Write a python program to concatenate two strings.

Solution :

Concatenate.py

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

Output



```
concatenation of str1: Python  and str2: Program  is  PythonProgram
>>> 
```

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 -

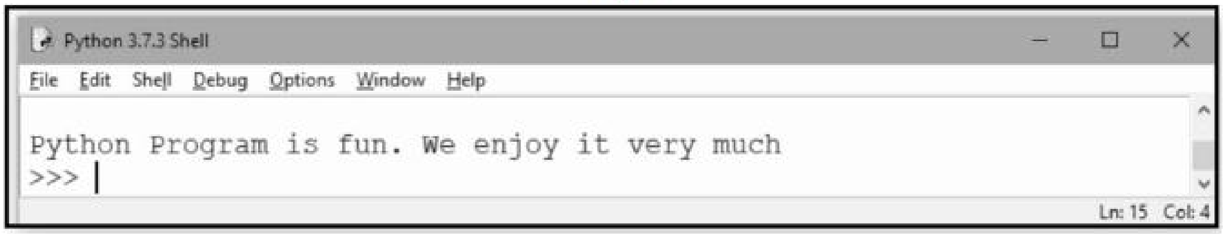
Example 3.6.2 : Write a python program to demonstrate the use of += operator form concatenation of the string.

Solution :

ConcatenateDemo.py

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

Output



```
Python Program is fun. We enjoy it very much
>>> |
```

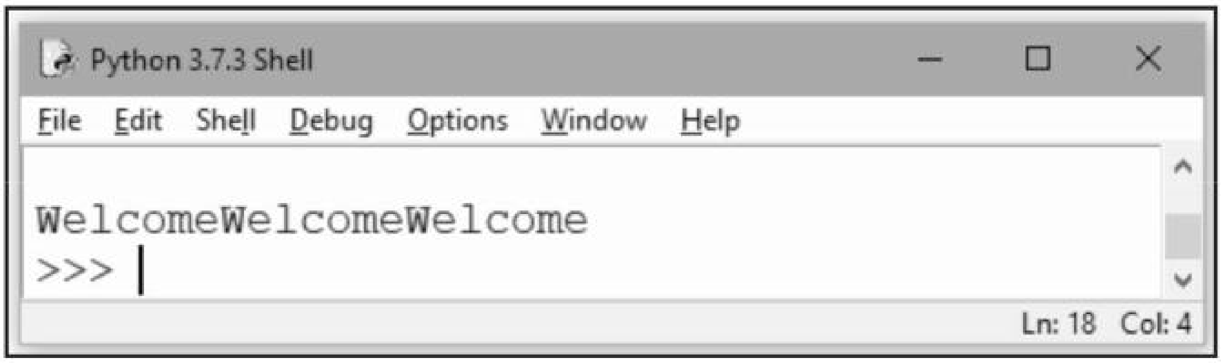
3.6.4 Multiplying Strings

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



```
WelcomeWelcomeWelcome
>>> |
```

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.*

of 184 − + 70%

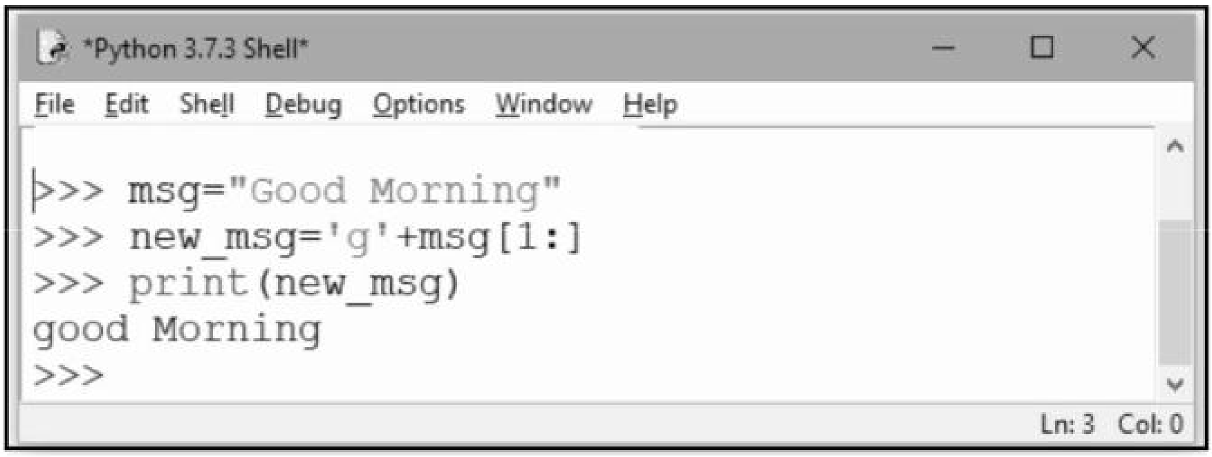
3.7 Strings are Immutable

- 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.



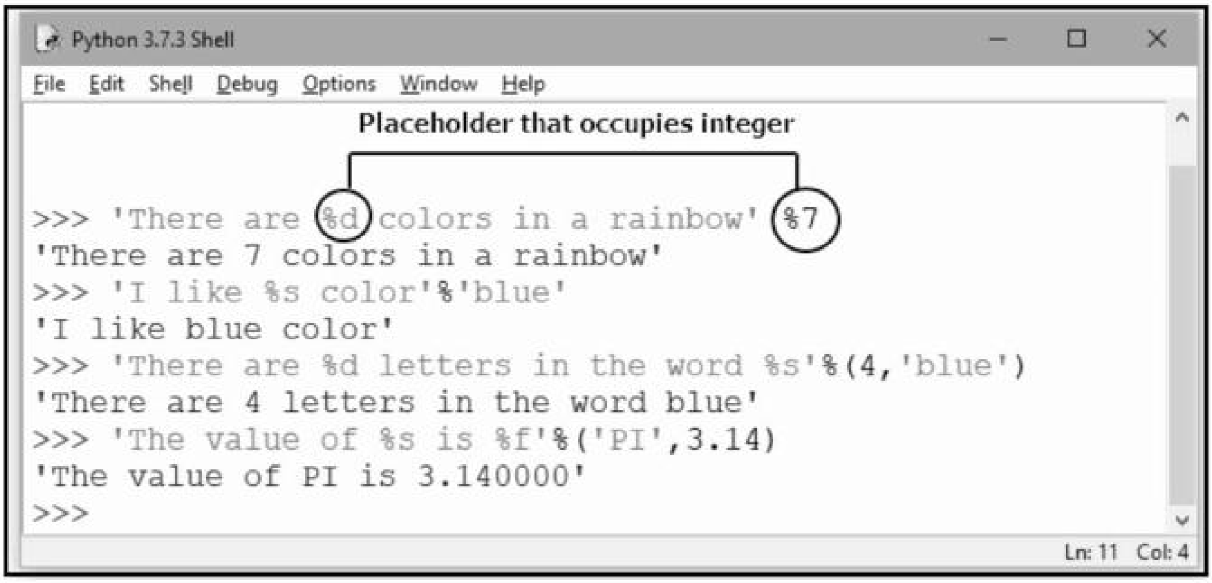
```
>>> msg="Good Morning"
>>> new_msg='g'+msg[1:]
>>> print(new_msg)
good Morning
>>>
```

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.


Python 3.7.3 Shell
File Edit Shell Debug Options Window Help
Placeholder that occupies integer
Ln: 11 Col: 4

```
>>> 'There are %d colors in a rainbow'%7
'There are 7 colors in a rainbow'
>>> 'I like %s color'%'blue'
'I like blue color'
>>> 'There are %d letters in the word %s'%(4,'blue')
'There are 4 letters in the word blue'
>>> 'The value of %s is %f'%('PI',3.14)
'The value of PI is 3.140000'
>>>
```

- 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

of 184 − + 70%

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
```

```
    for j in word_list:  #Iterating through the list
      if(len(j)>longest):  #if any word with longest length is found
          longest=len(j)# then update length of longest variable
         temp=j  # storing that word with longest length in temp variable
    return temp  #returning longest length word from function

print("The word with longest length: ",my_function()) #calling the function
              #to find the word with longest length
```

Output

```
Enter the number of words in list:5
Enter the word:Rupali
Enter the word:Asawari
Enter the word:Swati
Enter the word:Jai
Enter the word:Ashwini
The word with longest length:  Asawari
```

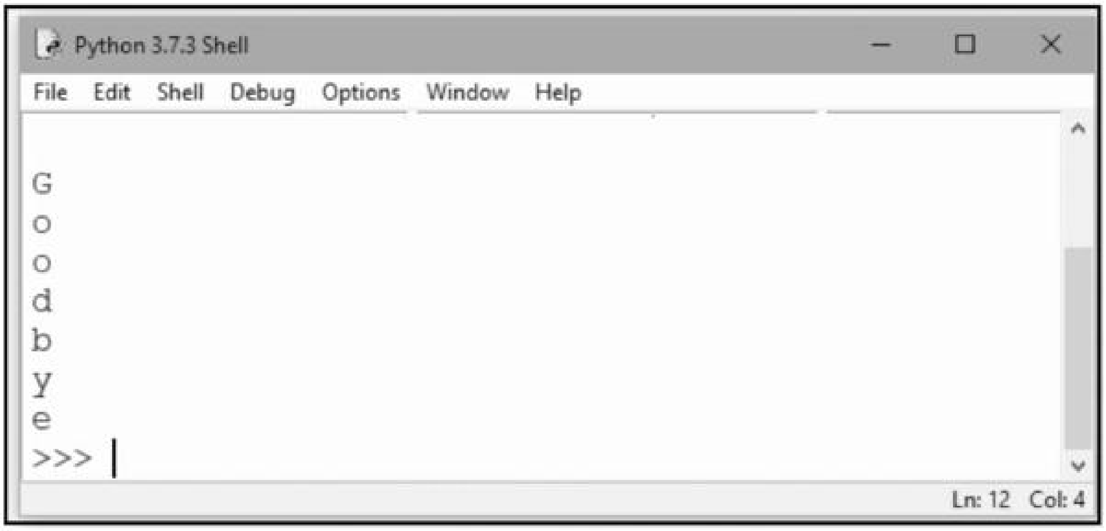
Example 3.9.2 : Write a program in python to traverse through string using while loop.

Solution :

StringTraverse.py

```
msg='Goodbye'
index = 0
while index<len(msg):
    letter=msg[index]
    print(letter)
    index=index+1
```

Output



```
G
o
o
d
b
y
e
>>> 
```

Example 3.9.3 : *Write a program in python to traverse through string using for loop.*

Solution :

StringTraverse1.py

```
msg='Goodbye'
index = 0
for index in range (0,len(msg)):
  letter=msg[index]
  print(letter)
  index=index+1
```

The output of this program is same as previous example.

In python, it is possible to count the number of times particular letter appeared in the string. Following example shows this idea.

Example 3.9.4 : Write a python program to display how many times particular letter appears in the string.

Solution :

CountDemo.py

```
def fun():
  str = 'aaabbbccddeecccddd'
  count = 0
  for i in str:
    if i == 'b':
      count = count + 1
  print("The string is ",str,"in which letter b appears ",count," times")
```

Output

```
>>> fun()
The string is  aaabbbccddeecccddd in which letter b appears  3  times
```

❑❑❑

f 184 − + 70%

Notes