INDEX

1) Language Fundamentals 2

2) Operators 31

3) Input and Output Statements 45

4) Flow Control 54

5) String Data Type 68

6) List Data Structure 87

7) Tuple Data Structure 104

8) Set Data Structure 112

9) Dictionary Data Structure 118

10) Functions 128

11) Modules 153

12) Packages 165

Introduction to Python

1.Under the leadership of "ANDREW STUART TANENBAUM" a group of employees developed distributed operating system.
2.group of employees used ABC scripting language to develop distributed operating system.
3."ABC" scripting language is very simple and easy to learn and work.
4."GUIDO VAN ROSSUM" is a member in that group and he likes ABC scripting language very well as it is very simple and easy.
5.In christmas holidays , Guido Van Rossum started developing a new language to be simpler and easy compare to ABC scripting language and all other existing language.
6.finally he developed a new language.
7.He likes "Nonty python's Flying Circus" english daily serial very well.
8.So finally Guido Van Rossum takenword python from that serial and kept for his language.
9.So finally Guido van Rossum developed python scripting language at the national research institute for mathematics & computer science in Netherland in 1989 and it available to public in 20th feb 1991.
10.Python is now maintained by a core development team at the institute,although Guido Van Rossum still hold the vital role in directing its progress.
11.Python 1.0 was released in january 1994.
12.Python 2.0 was released in october 2000 any python 2.7.11 in the last edition of python 2.
13.Meanwhile, Python 3.0 was released in december 2008.
14.Python is general purpose high level programming language.
15.Python is recommended as first programming language for beginners.
16.Guido Van Rossum has developed python language by taking almost all programming features from different languages.
17.The most of the syntaxes in python is taken from C and ABC languages.

Python:

--> Python is genral purpose programming language implemented by Guido Van Rossum in 1989 @ mathematics and science research center called CWI.CWI is located @ netherland.
-->Guido Van Rossum implemented python language by taking the different varities of language features like:

=> Procedure Oriented Programming Languages.
=> Object Orientyed programming languages.
=>Scripting Languages.
=>Modular Programming Languages.

NARESH IT, 2nd Floor, Durga Bhavani Complex, Satyam Threater Rd, Ameerpet, HYD
Telangana-500016 | Tel:+91 40 2374 6666 | Ph:+91 90009 94007 | www.nareshit.in

-->Guido Van Rossum make it available python to public in 1991. by using python we can implement different varities of application like :

1) Automation Application.
2) Data Analytics.
3)Web Application.
4)Scientific application.
5)Web Scrapping.
6)N/W with IOT.
7)Test Cases.
8)Gaming Application.
9)GUI Application.
10)Admin Application.
11)animation applications and so on.....

-->Currently copy right of pythons are registered with an community & non profitable organisation called Python S/W foundation.
-->Python S/W which is provided by python s/w foundation is known as "cpyhton".
There are so many distributions are available for cpython

1) Active Python.
2) Anaconda Python.
3) iPython.
4)Pocket Python.
5)PYPY Python.

Flavors of Python

1. CPython:

It is the standard flaver of python.It can be used to work with c language applications.

2. Jython or JPython:

It is for java applications. it can run on jvm.

3.IronPython:

it is for C#.net platform.

4.PYPY:

The main advantage of PYPY is performance will be improved because JIT compiler is available inside PVM.

5.RubyPython:

For Ruby Platforms.

6.Anaconda Python:

It is specially designed for handling large volume of data processing.

-->The current version of CPython is 3.x(3.1,3.2,.....3.7)
-->Download the required python 3.7 version from "http://python.org" website

-->NTFS file system format in windows (No proper security,But it is user friendly)
-->APFS file system in MAC OS.
-->EXT3 file system in linux (user, group, author level security is provided in linux, but not user friendly.)
-->HDFS this file system is similar to linux but with limited differences.

How to convert one OS file to System to the anther file system?

1) By using 3rd party S/w like FileZilla we can convert one file into another.
2) By using Shares directories also we can convert one file system to another.

we can develop the python programs in two modes:

1)Interactive mode
2) Batch mode------>|--->Editors
|--->IDE's

1) Interactive Mode:

--> The concept submitting 1-by-1 python statement explicitly to the python interpritter is known as an interactive mode.
--> We can submit 1-by-1 python statement explicitly to the python interpritter by using the python command line shell.
--> We can open the python cmd line shell, by executing the python command on cmd prompt or terminal.

Drawbacks:

-->Once if we exit from python cmd line prompt, the work which we done is going to loss.
-->Interactive mode is suitable for learning python & to test predefined functions functionalities, but it is not suitable for application development.

Batch Mode:

Concept of writting group of python statements in a file saving that file with extension.py file and submitting entire file to interpritter is known as batch mode.

-->We can develop the python files by using the "editors" or "IDE's".
-->Different editors are notepad, notepad++, editplus, nano, VI, GEdit etc...

-->Different IDE's are pycharm(90% usage), eclipse, netbeans, eric etc...

-->Open Notepad save file with .py in any drive and type code .
-->To run the program open cmd then type python filename.py or py filename.py

Editors Dis-Advantages:

-->By using editors takes longer time(we have to remeber every thing while writting program).
-->By development the python files by using editors we can't use code generation tools (Smart help like ctrl+space)
-->We can't perform the automatic debugging operations on python files by using editors.

ID's Advantages:

To overcome the problems we use IDE's. Different IDE's are:

- -->Pycharm
- -->Eric
- -->Eclipse
- -->Netbeans

-->Download and install pycharm
-->Open pycharm IDE by clicking on shortcut.

Features of Python:

1).Simple and easy to learn:

-->Python is a simple programming language, when we read python program, we can fell like reading english statements.
-->The syntaxes are very simple and only 30+ keywords are available.
-->When compared with other languages , we can write programs with very less number of lines. Hence more readability and simplicity.
-->We can reduce development time and cost of the project.

2).Freeware & open source:

-->We can use python s/w without any licence and it is a freeware.
-->It's source code is open, so that we can customize based on our requirement.
Ex: Jython is customized version of python to work with java applications.

3).High level programming language:

-->Python is a high level programming language hence it is a programmer friendly language.
-->Being a programmer we are not required to concentrate low level activities like memory management and security etc....

4).Platform Independent:

-->Once we write the python program, it can run on any platform without rewritting once again.
-->Internally PVM is responsible to convert into machine understandable form.\

5).Portability:

-->Python programs are portable. ie we can migrate from one platform to other platform very easily.
-->Python programs will provide same results on any platform.

**6).Dynamically Typed:

-->In python we are not required to declare type for variables.
-->Whenever we are assigning the value, based on value, type will be allocated automatically.
-->Hence python is considered as dynamically typed language.
-->But java, C are statically typed languages bcoz we have to provide type at the beginning only.
-->This dynamically typing nature will provide more flexibility to the programmer.

7).Both procedure oriented and object oriented:

-->Python language supports both procedure oriented(Like C, pascal etc) and object oriented (Like C++, Java) features.
-->Hence we can get benfits of both security & reusability.

8).Intrepreted:

-->We are not required to compile python programs explicitly. Internally python interpreter will take care that compilation.

9).Extensible:

-->We can use other language programs in python.
-->The main advantage of this approach are:
1).We can use already existing legacy non-python code.
2).We can improve performance of the application.

10).Embeded:

-->We can use python programs in any other language program.
-->i.e we can embedded python programs any where.

11).Extensive Library:

-->Python has a rich inbuilt library.
-->Being a programmer we can use this library directly and we are not responsible to implement the functionality.

Limitations of Python:

-->Performance wise not up to the mark bcoz it is interpreted language.
-->Not suitable for mobile applications.

Identfiers:

-->A name in the python program is called as identifier.
-->It can be a class name or function or module name or variable name.

```
Ex:     a=10
        def f():
        class test:
```

Rules to define identifiers in python:

1).The only allowed characters in python are
-->Alphabates(Either lower case or upper case)
-->Digits(0-9)
-->Underscore symbol(_)

-->By mistake if we are using any other symbol like $ then we will get syntax error.

Ex: cash=10(Valid)
ca$h=10(Invalid)

2).Identifiers should not start with digit.
Ex: 123total=500(Invalid)
total123=500(valid)

3).Identifiers are case sensitive. Of course python language is case sensitive language.
Ex: total=500
TOTAL=500

Note:

-->If identifier starts with underscore(_) then it indicates it is private.
-->We can't use reserved words as identifiers
Ex: def=10
-->There is no length limit for python identifiers. But not recommended to use too lengthy identifiers.
-->Dollar ($) symbol is not allowed in python.
-->If idetifier starts with (_ _)it indicates that strongly private identifiers.
-->If the identifier starts and ends with two underscores symbols then the isentifier is language defined special name, which is also known as magic methods.
Ex: __add__

Reserved Words:

->In python some words are reserved to represent some meaning or functionality. Such type of words are called as Reserved words.
-->There are 30+ reserved words in python
-->True, False, None
-->and, or, not, is
-->if, else, elif
-->while,for,break,continue,return,in,yield
-->try,except,finally,raise,assert
-->import,from,as,class,def,global,nonlocal,lambda,del,with

Note:

-->All the reserved words in python contain only alphabates symbols.
-->Except the following 3 reserved words, all contain only lower case alphabates.
True, False,None
Ex: a=true(invalid)
a=True(valid)

```
>>> import keyword
>>> keyword.kwlist
```

Data Types:

-->Any organization use cases or scenarios are 2-parts.
1).Data
2).Operations
-->Data of any organization usecase can be represented in any programming language by using data types and variables.
-->Every programming language supports data types & variables, but the data type of any programming language are not going to be same with the data types & variables of other programming language.
-->Operations of any organization use case can be represented in any programming language by using functions or methods.
-->Every programming language supports functions or methods or both.

Datatype:

-->Data types are nothing but some of the keywords of programming language, which are used to specify what type of data has to be store in the variable.
-->Without specifying data types variable memory allocation with not take place for variables.
-->Without allocating memory space for the variable we can't store the data into variables.
-->Programming languages supports 2-types of data types.
1).Static
2).Dynamic

1).Static Data Types:

-->In static type programming languages programmer should define the data type to the variable explicitly.
-->Once if we define data type to the variable explicitly, we can't modifythe the data type of that variable through out the program execution.
-->In static DT supported languages, one variable can store only one veraity of data.
-->C,C++,.Net,Java.....languages supports static data types.

2).Dynamic Data Types:

-->In dynamic data type supported languages, programmer should not define the data types to the variables explicitly.
-->At the time of execution of rogram, data type of variable is decided based on the data which is assigned to the variable.
-->Dynamic DT supported languages, one variable can store differenet varities of data.

-->Python, java script.....languages are supported dynamic data types.
-->Every data type in python language is internally implemented as a class.
-->Python data types are catagorised into 2-parts.
-->Fundamental types
-->Collection types

Python contains the following inbuilt data types:

Data Type	Description	Immutable	Example

int	We can use to represent the whole / integral values	Immutable	>>> a=10
			>>> type(a)
			<class 'int'>

Float	We can use to represent the decimal / floating point numbers	Immutable	>>> b=10.2
			>>> type(b)
			<class 'float'>

Complex	We can use to represent the complex numbers	Immutable	>>> c=10+5j
			>>> type(c)
			<class 'complex'>
			>>> c.real
			10.0
			>>> c.imag
			5.0

Bool	WE can use to represent the logical values(Only allowed values are True and False)	Immutable	>>> flag=True
			>>> flag=False
			>>> type(flag)
			<class 'bool'>

str	To represent sequence of characters	Immutable	>>> s='mahesh'
			>>> type(s)
			<class 'str'>
			>>> s="mahesh"
			>>> s='''Durga Software Solutions'''
			>>> type(s)
			<class 'str'>

bytes	To represent a sequence of byte values from 0-256	Immutable	>>> list=[10,20,30,40]
			>>> b=bytes(list)
			>>> type(b)
			<class 'bytes'>

bytearray	To represent a sequence of byte values from 0-256	Muttable	>>> list=[10,20,30,40]
			>>> ba=bytearray(list)
			>>> type(ba)
			<class 'bytearray'>

NARESH IT, 2nd Floor, Durga Bhavani Complex, Satyam Threater Rd, Ameerpet, HYD
Telangana-500016 | Tel:+91 40 2374 6666 | Ph:+91 90009 94007 | www.nareshit.in

range	To represent a range of values	Immutable	>>> r=range(10)
			>>> r1=range(10,20)
			>>> r2=range(10,50,5)

list	To represent an ordered collection of objects	Muttable	>>> l=[10,20,30,40]
			>>> type(l)
			<class 'list'>

tuple	To represent an ordered collection of objects	Immutable	>>> t=(1,2,3,4,5)
			>>> type(t)
			<class 'tuple'>

set	To represent an unordered collection of unique objects	Muttable	>>> s={1,2,3,4,5,6}
			>>> type(s)
			<class 'set'>

frozenset	To represent an unordered collection of unique objects	Immutable	>>> s={1,2,3,'Mahesh',100,'Durga'}
			>>> fs=frozenset(s)
			>>> type(fs)
			<class 'frozenset'>

dict	To represent a group of key value pair	Muttable	>>> d={100:'Mahesh',102:'Durga',103:'Sunny'} >>> type(d) <class 'dict'>

1.int
2.float
3.complex
4.bool
5.str
6.bytes
7.bytesarray
8.range
9.list
10.tuple
11.set
12.frozenset
13.dict
14.None

```
>>> a=10
>>> id(a)
140731593971008
>>> b=10
>>> id(b)
140731593971008
>>> a=20
>>> id(a)
140731593971328
```

Note:
Python contains several inbuilt functions

1).type():
To check the type of variable.
2).id():
To get the address of object.
-->In python every thing is an object
3).print():
To print the value

int Data Type:

-->We can use int data type to represent whole numbers(Integral values).
Ex:

```
a=10
type(a)#int
```

Note:

-->In python-2 we have long data type to represent very large integer values.
-->But in python-3 there is no long type explicitly we can represent long values also by using int type only.

-->We can represent int values in the following ways.
1).Decimal form
2).Binary form
3).Octal form
4).Hexa decimal form

1).Decimal Form(Base-10):

-->It is the default number system in python.
-->The allowed digits are :0 to 9.
Ex: a=10

2).Binary form(Base-2):

-->The allowed digits are : 0 & 1.
-->Literal value should be prefixed with 0b or 0B.
Ex: a=0b1111(valid)
a=0B1111(valid)
a=0B123(Invalid)

3).Octal form(Base-8):

-->The allowed digits are :0 to 7
-->Literal value should be prefixed with 0o or 0O
Ex: a=0o123(Valid)
a=0O786(Invalid)

4).Hexa Decimal Form(Base-16):

-->The allowed digits are :0 to 9, a-f(Both lower and upper case are allowed).
-->Literal value should be prefixed with 0x or 0X.
Ex: a=0XFACE(Valid)
a=0xBeef(Valid)
a=0XBeer(Invalid)

Note:
Being a programmer we can specify literal values in decimal, binary, octal and hexa decimal forms, But PVM will always provide values in decimal form.
Ex:

```
1) >>> a=10
2) >>> b=0b10
3) >>> c=0o10
4) >>> d=0x10
5) >>> print(a):10
6) >>> print(b):2
7) >>> print(c):8
8) >>> print(d):16
```

Base Conversions:

Python provides the following in-built functions for base conversions

1).bin():

We can use bin() to convert from any base to binary

```
>>> bin(15)
'0b1111'
>>> bin(0o11)
'0b1001'
>>> bin(0x10)
'0b10000'
```

2).oct():

We can use oct() to convert from any base to octal.

```
>>> oct(10)
'0o12'
>>> oct(0b1111)
'0o17'
>>> oct(0x123)
'0o443'
```

3).hex():

We can use hex() to convert any base to hexa decimal.

```
>>> hex(100)
'0x64'
>>> hex(0B111111)
'0x3f'
>>> hex(0o12345)
'0x14e5'
```

2).Float:

-->We can use float data type to represent floating point values(decimal values).

```
>>> f=1.234
>>> type(f)
<class 'float'>
```

-->We can also represent floating point values by using exponential form(Scintific notation)

```
1.          >>> f=1.2e3
2.      >>> f
3.      1200.0
```

Note: instead of 'e' we can use 'E'
Ex: f=1.2E3

-->The main advantage of exponential form is we can represent big values in less memory.

Note:

We can represent int values in decimal, binary,octal and hexa decimal forms. But we can represent float values by using decimal form only.

```
1.  f=0B11.10(Invalid)
2.  f=0o123.234(Invalid)
3.      f=0x12.face(Invalid)
```

3).Complex Data Type:

A complex is of the form
a+bj
a-->Real part
b-->Imaginary part
j-->sqrt(-1)
-->a & b contains integers or floating point values.
-->In the real part if we use int values then we can specify that either by decimal, octal, binary or hexa decimal form.
-->But imaginary part should be specified only by using decimal form.
Ex:

```
1.  a=3+4j
2.  a=1.5+2.5j
3.  a=3+0B1010j(Invalid)
4.  a=0b10101+3J(valid)
```

-->We can perform operations on complex type values

```
1.  >>> a=3+1.5j
2.  >>> b=4+2.5j
3.  >>> a+b
```

```
(7+4j)
>>> a-b
(-1-1j)
```

Note:
Complex data type has some inbuilt attributes to retrieve the real part and imaginary part.

```
>>> c=10.5+3.6j
    >>> c.real
    10.5
    >>> c.imag
    3.6
```

-->We can use complex type generally in scintific applications and electrical engineering applications.

4).Bool Data Type:

-->We can use this data type to represent boolean values.
-->The only allowed values for this data type are:
True and False
-->Internally python represents True as 1 and False as 0.
Ex:

```
a=True
type(a)==>bool
```

Ex:

```
        >>> a=10
    >>> b=20
    >>> c=a<b
    >>> print(c)
    True

    >>>True+True==>2
    >>>True-False==>1
```

5).str Data Type:

-->str represents String data type.
-->A string is a sequence of characters with in single quotes or double quotes.
Ex: s='Mahesh'

s="Mahesh"

-->By using single quotes or double quotes we can't represent multi line string literals.

Ex: s="Mahesh
Dasari"(Invalid)

-->For this requirement we should go for triple quotes(''') or triple double quotes(""")

Ex: s='''Mahesh
Dasari'''
s="""Durga
Soft"""

-->We can embeded one string in another string.

Ex: '''This "python classes very helpful" for java students'''.

Slicing of Strings:

-->Slice means a piece.

-->[:] is called as slice operator, which can be used to retrieve parts of string.

-->In python string follows zero based index.

-->The index can be either +ve or -ve.

-->+ve index means forward direction from Left to Right.

-->-ve index means backward direction from Right to Left

```
1.          >>> s="python"
2.      >>> s[0]
3.      'p'
4.      >>> s[1]
5.      'y'
6.      >>> s[-1]
7.      'n'
8.      >>> s[30]
9.      IndexError: string index out of range
10.     >>> s[1:40]
11.     'ython'
12.     >>> s[1:]
13.     'ython'
14.     >>> s[:4]
15.     'pyth'
16.     >>> s[:]
17.     'python'
18.     >>> s[-1:-4](Begin position should be lower than end position)
19.     ''
20.     >>> s[-4:-1]
21.     'tho'
22.     >>> s*3
```

```
'pythonpythonpython'
>>> len(s)
6
```

Note:

-->In the python the following data types are considered as fundamental data types.

- **-->int**
- **-->float**
- **-->complex**
- **-->bool**
- **-->str**

-->In python , we can represent char values also by using str type and explicitly char type not available.

```
>>> c='a'
>>> type(c)
<class 'str'>
```

-->long data type is available in python2 but not in python3. In python3 long values also we can represent by using int type only.

Type Casting:

-->We can convert one type to other type. This conversion is called as Typecasting or Type coersion.

-->The following are various in-built functions for type casting.

- **-->int()**
- **-->float()**
- **-->complex()**
- **-->bool()**
- **-->str()**

1).int():

We can use this function to convert values from other types to int.

```
>>> int(123.456)
123
>>> int(10+4j)
TypeError: can't convert complex to int
>>> int(True)
1
>>> int(False)
```

```
0
>>> int("10")
10
>>> int("10.5")
ValueError: invalid literal for int() with base 10: '10.5'
>>> int("ten")
ValueError: invalid literal for int() with base 10: 'ten'
>>> int("0B1010")
ValueError: invalid literal for int() with base 10: '0B1010'
```

Note:

-->We can convert from any type to int except complex type.

-->If we want to convert str type to int type, compalsury str should contains only integral value and should be specified in base-10.

2).float():

We can use float() to convert other type values to float type.

```
>>> float(10)
10.0
>>> float(10+4j)
TypeError: can't convert complex to float
>>> float(True)
1.0
>>> float(False)
0.0
>>> float("10")
10.0
>>> float("10.5")
10.5
>>>float("ten")
ValueError: could not convert string to float: 'ten'
```

Note:

-->We can convert any type value to float type expcept complex type.

-->Whenever we are trying to convert str type to float type compalsury str should be either integral or floating point literal and should be specified only in base-10.

3).complex():

We can use complex() function to convert other data type to complex type.

Form-1:complex(x)

We can use this function to convert x into complex number with real part x and imaginary part 0.

```
>>> complex(10)
(10+0j)
>>> complex(10.5)
(10.5+0j)
>>> complex(True)
(1+0j)
>>> complex(False)
0j
>>> complex("10")
(10+0j)
>>> complex("10.5")
(10.5+0j)
>>> complex("ten")
ValueError: complex() arg is a malformed string
```

Form-2:complex(x,y)

We can use this method to convert x and y into complex number such that x will be real part and y will be imaginary part.

```
>>> complex(10,2)
(10+2j)
>>> complex(10,-2)
(10-2j)
>>> complex(True,False)
(1+0j)
>>> complex("10","10.2")
TypeError: complex() can't take second arg if first is a string
```

4).bool():

We can use this function to convert other type to bool type

```
>>> bool(0)
False
>>> bool(1)
True
>>> bool(10)
True
>>> bool(0.123)
True
```

```
>>> bool(10.5)
True
>>> bool(0.0)
False
>>> bool(10+4J)
True
>>> bool(0+0j)
False
>>> bool("True")
True
>>> bool("")
False
>>> bool(None)
False
>>> bool(" ")
True
```

5).str():

We can use this method to convert other type values to str type.

```
>>> str(10)
'10'
>>> str(10.5)
'10.5'
>>> str(True)
'True'
>>> str(10+4j)
'(10+4j)'
```

Fundamental Data types vs Immutability:

-->All the fundamental data types are immutable. i.e once we creates an object, we can't perform any changes in that object. If we are trying to change then with those changes a new object will be created. This non-changable behaviour is called as immutability.

-->In python if a new object is required, then PVM won't create object immediately. First it will check if any object is available with the required content or not. If available then existing object will be re-used. If it is not available then only a new object will be created. The advantage of this approach is memory utilization and performance will be improved.
-->But the problem in this approach is, several references pointing to the same object, by using one reference if we are allowed to change the content in the existing object then the remaining references will be effected. To prevent this immutability concept is required.

Accordint to this once we create an object we are not allowed to change content. If we are trying with those changes a new object will be created.

```
>>> a=10
>>> b=10
>>> id(a)
140723035886912
>>> id(b)
140723035886912
>>> a is b
True
```

```
>>> a=10+5j
>>> b=10+5j
>>> id(a)
2560497332016
>>> id(b)
2560497331568
>>> a is b
False
```

No-Reusability

```
>>> a=True
>>> b=True
>>> a is b
True
>>> id(a)
140723035355472
>>> id(b)
140723035355472
```

```
>>> a='mahesh'
>>> b='mahesh'
>>> id(a)
3003052190160
>>> id(b)
3003052190160
>>> a is b
True
```

```
>>> a=10.123
>>> b=10.123
>>> id(a)
3003011087360
>>> id(b)
3003011087408
>>> a is b
False
```

No-Reusability

```
Note:
-------
-->There is no Re-Usability for float & complex types.
-->Re-Usability is applicable for int only the range 0-256.
```

Ex:

```
1) >>> a=10
2) >>> b=10
3) >>> id(a)
4) 140723035886912
5) >>> id(b)
6) 140723035886912
7) >>> a is b
8) True
9)
```

Note:

-->There is no Re-Usability for float & complex type.

-->Re-Usability is applicable for int only the range 0-256.

6).bytes Data Type:

bytes data type represents a group of byte numbers just like an array.

Ex:

```
1) >>> x=[10,20,30,40]
2) >>> b=bytes(x)
3) >>> type(b)
4) <class 'bytes'>
5) >>> b[0]
6) 10
7) >>> b[-1]
```

```
8)  40
9)  >>> for i in b:print(i)
10) 10
11) 20
12) 30
13) 40
```

Conclusion 1:

The only allowed values for byte data type are 0 to 256. BY mistakes if we are trying to provide any other values then we will get value error.

```
Ex:     >>> x=[10,20,300,40]
        >>> b=bytes(x)
        ValueError: bytes must be in range(0, 256)
```

Conclusion 2:

Once we create bytes data type value, we can't change it values, if we are trying to change we will get an error.

```
Ex:     >>> x=[10,20,30,40]
        >>> b=bytes(x)
        >>> b[2]=50
        TypeError: 'bytes' object does not support item assignment
```

7).bytearray Data type:

bytearray is exactly same as bytes data type except that its elements can be modified.

```
1)  >>> x=[10,20,30,40]
2)  >>> b=bytearray(x)
3)  >>> for i in b:print(i)
4)  10
5)  20
6)  30
7)  40
8)  >>> b[2]=50
9)  >>> for i in b:print(i)
10) 10
11) 20
12) 50
13) 40
```

8).List Data Type:

If we want to represent a group of values as a single entity where insertion order is required to preserve and duplicates are allowed then we should go for list data type.

-->Insertion order is preserved.
-->Heterogeneous objects are allowed.
-->Duplicates are allowed.
-->Growable in nature.
-->Values should be enclosed within square brackets [ ].

Ex:

```
1) >>> list=[10,20,'mahesh',True]
2) >>> print(list)
3) [10, 20, 'mahesh', True]
```

Ex:

```
1) >>> list=[10,20,30,40]
2) >>> list[0]
3) 10
4) >>> list[-1]
5) 40
6) >>> list[1:3]
7) [20, 30]
8) >>> list[0]=100
9) >>> for i in list:print(i)
10) 100
11) 20
12) 30
13) 40
```

-->List is growable in nature. i.e based on our requirement we can increase or decrease the size.
Ex:

```
1) >>> list=[10,20,30]
2) >>> list.append('mahesh')
3) >>> list
4) [10, 20, 30, 'mahesh']
5) >>> list.remove(20)
6) >>> list
7) [10, 30, 'mahesh']
8) >>> list2=list*2
```

```
>>> list2
[10, 30, 'mahesh', 10, 30, 'mahesh']
```

Note:

An order, mutable, heterogenous collection of elements is nothing but a list, where duplicates also allowed.

9).tuple data type:

-->tuple data type is exactly same as list data type except that it is immutable. i.e we can't change values.

-->tuple elements can be represented within parenthesis ().

Ex:

```
>>> t=(10,20,30,40)
>>> type(t)
<class 'tuple'>
>>> t[0]
10
>>> t[0]=100
TypeError: 'tuple' object does not support item assignment
>>> t.append('mahesh')
AttributeError: 'tuple' object has no attribute 'append'
>>> t.remove(10)
AttributeError: 'tuple' object has no attribute 'remove'
>>> t[1:3]
(20, 30)
```

10).range data type:

-->Range data type represents a sequence of numbers.

-->The elements present in range data type are not modifiable. i.e range data type is immutable.

Form-1:range(10)

->Generates numbers from 0-9

```
Ex:     r=range(10)
        >>> for i in r:print(i)
```

Form-2: range(10,20)

-->Generates numbers from 10-19

```
Ex:     r=range(10,20)
        >>> for i in r:print(i)
```

Form-3: range(10,50,5)

-->5 means step(increment value)

```
>>> for i in r:print(i)
>>>10 15 20 25 30 35 40 45
```

-->We can access elements present in the range data type by using index.

Ex:

```
1) >>> r=range(10,20)
2) >>> r[0]
3) 10
4) >>> r[15]
5) IndexError: range object index out of range
6) >>> r[0]=100
7) TypeError: 'range' object does not support item assignment
```

-->We can create a list of values with range data type

Ex:

```
1) >>> l=list(range(10))
2) >>> l
3) [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

11).set Data Type:

If we want to represent a group of values without duplicates where order is not important then we should go for set data type.

-->Insertion order is not preserved.

-->Duplicates are not allowed.

-->Heterogeneous objects are allowed.

-->Index concept is notr allowed.

-->It is a mutable collection.

-->Growable in nature.

Ex:

```
1) >>> s={10,20,30}
2) >>> s
3) {10, 20, 30}
4) >>> s[0]
5) TypeError: 'set' object does not support indexing
6) >> s={10,20,30,'mahesh'}
```

```
>>> s
{'mahesh', 10, 20, 30}
```

-->set is a growable nature, based on our requirement we can increase or decrease the size.

```
>>> s.add(40)
>>> s
{40, 10, 20, 'mahesh', 30}
>>> s.remove(20)
>>> s
{40, 10, 'mahesh', 30}
```

12).frozenset Data Type:

-->It is exactly same as set except that it is immutable.
-->Hence we can't use add or remove functions.

Ex:

```
>>> s={10,20,30,40}
>>> fs=frozenset(s)
>>> type(fs)
<class 'frozenset'>
>>> fs
frozenset({40, 10, 20, 30})
>>> for i in fs:print(i)
40
10
20
30
>>> fs.add(50)
AttributeError: 'frozenset' object has no attribute 'add'
>>> fs.remove(20)
AttributeError: 'frozenset' object has no attribute 'remove'
```

13).dict Data Type:

-->If we want to represent a group of values as key-value pairs then we should go for dict data type.

Ex:

```
>>> d={100:'mahesh',102:'durga',103:'sunny'}
>>> type(d)
```

```
3) <class 'dict'>
4) >>> d[100]
5) 'mahesh'
```

-->Duplicate keys are not allowed but values can be duplicated. If we are trying to insert an entry with duplicate key then old value will be replaced with new value.

```
1) >>> d[100]='suresh'
2) >>> d
3) {100: 'suresh', 102: 'durga', 103: 'sunny'}
```

-->We can create an empty dictionaly as follows.

```
1) >>> d={}
2) >>> type(d)
3) <class 'dict'>
```

-->We can add key-value pairs as follows.

```
1) >>> d['a']='apple'
2) >>> d['b']='banana'
3) >>> d
4) {'a': 'apple', 'b': 'banana'}
```

-->We can create an empty set by using set() function.

```
1) >>> s=set()
2) >>> type(s)
3) <class 'set'>
4) >>> s.add(10)
5) >>> s.add(30)
6) >>> s
7) {10, 30}
```

Note: dict is mutable and the order wont be preserved.

Note:

-->In general we can use bytes and bytearray data types to represent binary information like images, video files etc....

-->In python-2 long data type is available. But in python-3 it is not available and we can represent long values also by using int type only.

-->In python there is no char data type. Hence we can represent char values also by using str type

14).None Data Type:

-->None means nothing or No value aasociated.
-->If the value is not available, then to handle such type of cases None introduced.
-->It is some thing like null value in java.

```
>>> def f1():
a=10
>>> print(f1())
None
>>> def f2(): print("Hello Don't be like a None")
>>> f2()
Hello Don't be like a None
def f3():
    pass
```

Escape Characters:

-->In string literals we can use escape characters to aasociate a special meaning.

```
>>> s="Durga\nSoftware"
>>> print(s)
Durga
Software
>>> s="Durga\tSoftware"
>>> print(s)
Durga   Software
>>> s="This is " symbol"
SyntaxError: invalid syntax
>>> s="This is \" symbol"
>>> print(s)
This is " symbol
```

-->The following are various important escape characters in python.
1).\n==>New Line
2).\t==>Horizantal tab
3).\r==>Carriage Return
4).\b==>Back Space
5).\f==>Form Feed
6).\v==>Vertical tab
7).\'==>Single quote
8).\"==>Double quote
9).\\==>back slash symbol.

Constants:

-->Constants concept is not applicable in python.
-->But it is convention to use only upper case characters if we don't want to change.

```
MAX_VALUE=10
```

-->It is just convention but we can change the value.

OPERATORS

-->Any person who is doing certain activity is called as an operator.
-->Operator is a symbol that performs certain operations.
-->Python provides the following set of operators.
1).Arithmetic Operators
2).Relational Operators OR Comparision Operators
3).Logical Operators.
4).Bitwise Operators
5).Assignment Operators
6).Special Operators

1).Arithmetic Operators:

+ ==>Addition
- ==>Subtratction
* ==>MUltiplication
/ ==>Division operator
% ==>Modulo operator

// ==>Floor Division operator
** ==>Exponent operator or power operator

Ex:

```
1)  >>> a=10
2)  >>> b=2
3)  >>> print('a+b=',a+b)
4)  a+b=12
5)  >>> print('a-b=',a-b)
6)  a-b=8
7)  >>> print('a*b=',a*b)
8)  a*b= 20
9)  >>> print('a/b=',a/b)
10) a/b=5.0
11) >>> print('a//b=',a//b)
12) a//b=5
13) >>> print('a%b=',a%b)
14) a%b=0
15) >>> print('a**b=',a**b)
```


NARESH IT, 2nd Floor, Durga Bhavani Complex, Satyam Threater Rd, Ameerpet, HYD
Telangana-500016 | Tel:+91 40 2374 6666 | Ph:+91 90009 94007 | www.nareshit.in

```
16) a**b=100
```

Ex:

```
1)  a=10.5
2)  b=2
3)  a+b=12.5
4)  a-b=8.5
5)  a*b=21.0
6)  a/b=5.25
7)  a//b=5.0
8)  a%b=0.5
9)  a**b=110.25
```

Ex:

```
1)  >>> 10/2-->5.0
2)  >>> 10//2  -->5
3)  >>> 10.0/2-->5.0
4)  >>> 10.0//2-->5.0
```

Note:

-->/ operator always performs floating point arithmetic, Hence it will always returns float value.
-->But floor division(//) can perform both floating point and integral arithmetic. If arguments are int type then result is int type. If atleast one argument is float type then the result is float type.

Note:

-->We can use +, * operators for str data type also.
-->If we want to use + operator for str type then compalsary both arguments should be str type only otherwise we will get error.

```
1.  >>> "Mahesh"+3
2.  TypeError: can only concatenate str (not "int") to str
3.  >>> "Mahesh"+"3"
4.  'Mahesh3'
```

-->If we use * operator for str type then compalsary one argument should be int and other argument should be str type.

```
1)  >>> "Mahesh"*3
2)  'MaheshMaheshMahesh'
3)  >>> 3*"Mahesh"
```

```
MaheshMaheshMahesh'
>>> "Mahesh"*2.5
TypeError: can't multiply sequence by non-int of type 'float'
>>> "Mahesh"*"Mahesh"
TypeError: can't multiply sequence by non-int of type 'str'
```

Note: For any number x,
-->x/0 and x%0 always raise "ZeroDivisionError"
10/0, 10%0.

2).Relational Operators:

>, >=, <, <=

Ex:

```
a=10
b=20
a>b is False
a>=b is False
a<b is True
a<=b is True
```

-->We can apply relational operators for str type also

Ex:

```
>>> a="mahesh"
>>> b="durga"
>>> a>b is  True
>>> a<b is False
>>> a="durga"
>>> b="darga"
>>> a>b is True
>>> a="durga"
>>> b="Durga"
>>> a>b is True
```

Ex:

```
>>> True>True
False
>>> True>=True
True
>>> 10>True
```


NARESH IT, 2nd Floor, Durga Bhavani Complex, Satyam Threater Rd, Ameerpet, HYD
Telangana-500016 | Tel:+91 40 2374 6666 | Ph:+91 90009 94007 | www.nareshit.in

```
6)  True
7)  >>> False>True
8)  False
9)  >>> 10>'a'
10) TypeError: '>' not supported between instances of 'int' and 'str'
```

Ex:

```
1)  a=10
2)  b=20
3)  if(a>b):
4)      print("a is greater than b")
5)  else:
6)      print("a is not grater than b")
```

O/P: a is not grater than b

Note:

Chaining of relational operators is possible. In the chaining, if all comparisions returns True then only result is True. If atleast one comparision False then the result is False.

```
1)  >>> 10<20
2)  True
3)  >>> 10<20<30
4)  True
5)  >>> 10<20<30<40
6)  True
7)  >>> 10<20<30<40>50
8)  False
```

3).Equality Operators:

==,!=

-->We can apply these operators for any type even for incompatible types also.

```
1)  >>> 10==20
2)  False
3)  >>> 10!=20
4)  True
5)  >>> 10==True
6)  False
7)  >>> False==False
8)  True
9)  >>> 'mahesh'=='mahesh'
10) True
```

```
>>> 10=='mahesh'
False
>>> 10==10.0
True
>>> 10==10.1
False
>>> 10=='10'
False
>>> 10.10==10.1
True
>>> 1==True
True
```

Note: Chaining concept is applicable for equality operators. If atleast one comparision returns False then the result is False.Otherwise the result is True.

```
>>> 10==20==30==40
False
>>> 10==5+5==3+7==2*5
True
```

Note: Never raise an error even for complex numbers.

```
>>> (10+2j)==(10+2j)
True
>>> (10+2j)==(10+5j)
False
```

4).Logical Operators:

and,or,not

-->We can apply for all types.

For boolean types behaviour:

and==>If both arguments are True then only result is True.
or==>If one argument is True then result is True.
not==>Complement

Ex: True and False==>False
True or False==>True
not False==>True

For non-boolean types behaviours:

-->0 means false
-->non-zero means True
-->Empty string is always treated as False

x and y:

-->If x evaluates to False return x otherwise return y.

```
1) >>> 10 and 20
2) 20
3) >>> 0 and 20
4) 0
```

-->If the first argument is zero then result is zero otherwise result is y.

x or y:

-->If x evaluates to True then result is x otherwise result is y

```
1) >>> 10 or 20
2) 10
3) >>> 0 or 20
4) 20
```

not x:

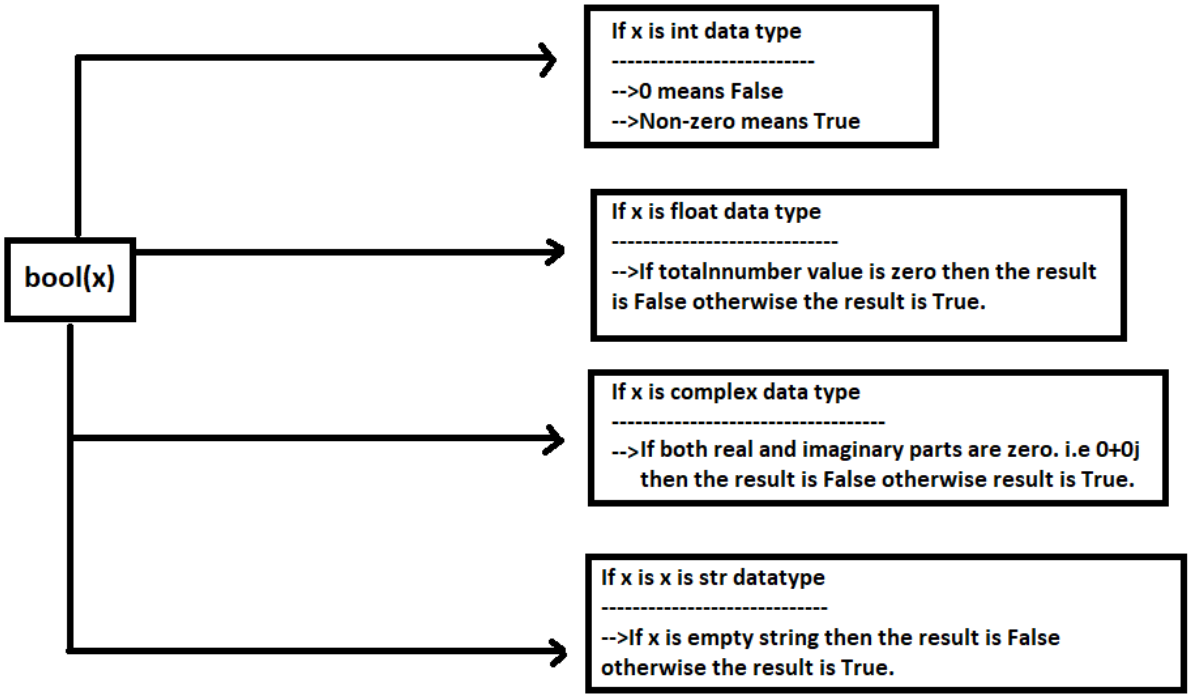
-->If x evaluates to False then the result is True otherwise False.
Ex:

```
1) >>> not 10
2) False
3) >>> not ''
4) True
```

Ex:

```
1) >>> "durga" and "durgasoft"
2) 'durgasoft'
3) >>> "" and "durgasoft"
4) ''
5) >>> "" or "durgasoft"
6) 'durgasoft'
7) >>> "durgasoft" or ""
```

```
'durgasoft'
>>> not "durgasoft"
False
>>> 10 or 10/0
10
>>> 0 or 10/0
ZeroDivisionError: division by zero
```



5).Bitwise Operators:

-->We can apply these operators bitwise.
-->These operators are applicable only for int and boolean types.
-->By mistake if we are trying to apply for any other data type then we will get an error.
&, |, ^, ~, <<, >>

Ex:

```
print(4 & 5)==>Valid
print(10.5 & 5.6)==>TypeError: unsupported operand type(s) for &: 'float' and 'float'
print(True & True)==>Valid
```

&==>If both bits are 1 then only result is 1 otherwise result is 0.
|==>If atleast one bit is 1 then the result is 1 otherwise result is 0.
^==>x-or==>If bits are different then only result is 1 otherwise result is 0.
~==>bitwise complement operator

1==>0 & 0==>1

Ex:

```
1) >>> print(4&5) 4
2) >>> print(4|5) 5
3) >>> print(4^5) 1
```

bitwise complement operator(~):

-->We have to apply complement for total bits.

```
1) >>> print(~4)  -5
2) >>> print(~True)   -2
```

Note:

-->The most significant bit acts as sign bit. 0 value represents +ve number where as 1 represent -ve value.
-->Positive numbers will be represented directly in memory where as -ve numbers will be represented indeirectly in 2's complement form.

Shift Operators:

<< Left shift operator:

-->After shifting the empty cell we have to fill with zero
Ex: >>> print(10<<2) 40

>>Right shift operator:

-->After shifting the empty cells we have to fill with sign bit.(0 for positive and 1 for -ve)
Ex:

```
1) >>> print(10>>2) 2
2) >>> print(-10>>2)   -3
```

-->We can apply bitwise operators for boolean type also

Ex:

```
1) >>> print(True & False)   False
2) >>> print(True | False)   True
3) >>> print(True ^ False)   True
4) >>> print(~True)   -2
```

```
5) >>> print(True<<2)  4
6) >>> print(True>>2)  0
```

6).Assignment Operators:

-->We can use assignment operator to assign a value to the valiable.
Ex: x=10

-->We can combine assignment operator with some other operator to form compound assignment operator.
Ex: x+=10==>x=x+10

-->The following are the possible compound assignment operators in python
+=, -=, *=, /=, %=, //=, **=, &=, |=, ^=
-->No increment operators in python

Ex:
x++==>Invalid syntax
++x==>Valid
x--==>Invalid syntax
--x==>Valid

7).Ternary Operator:

-->In java we can use
x=(condition)?First value:Secondvalue
Ex:

```
1) x=(10<20)?30:40
2) print(x)
3) SyntaxError: invalid syntax(In python)
```

-->In python we can use ternary operator
x = Firstvalue if condition else Secondvalue

Ex:

```
1) x=30 if 10<20 else 40
2) print(x)
```

o/p:30

Ex:

```
1) a,b=10,20
2) x=30 if a>b else 40
```

```
3) print(x)
```

o/p:40

-->Read two number from keyboard then find min value.

Note: Bydefault if we enter anything from the keyboard it will be treated as str type, hence we convert into int.

Ex:

```
1) a=int(input("Enter first number :"))
2) b=int(input("Enter second number :"))
3) min=a if a<b else b
4) print("Minimum value :", min)
```

-->Firstvalue if condition1 else Secondvalue if condition2 else Thirdvalue

Ex:

```
1) x=10 if 20<30 else 40 if 50<60 else 70
2) print(x)
```

o/p:10

```
1) x=10 if 20>30 else 40 if 50>60 else 70
2) print(x)
```

o/p:70

-->Find max number in three given numbers

```
1) a=int(input("Enter first value :"))
2) b=int(input("Enter second value :"))
3) c=int(input("Enter third value :"))
4) max=a if a>b and a>c else b if b>c else c
5) print("Max value :",max)
```

8).Special Operators:

-->Python defines the following 2-special operators

1).Identity operator

2).Memebership operator

1).Identity operator:

We can use identify operators for address comparision. 2-identifty operators are available.

-->is

-->is not

-->r1 is r2 returns True if both r1 and r2 are pointing to the same object.

-->r1 is not r2 returns True if both r1 and r2 are not pointing to the same object.

Ex:

```
1) a=10
2) b=10
3) print(a is b)==>True
```

Ex:

```
1) x=True
2) y=True
3) print(x is y)==>True
```

Ex:

```
1) a="mahesh"
2) b="mahesh"
3) print(id(a))
4) print(id(b))
5) print(a is b)
```

Ex:

```
1) list1=["one","two","three"]
2) list2=["one","two","three"]
3) print(id(list1))
4) print(id(list2))
5) print(list1 is list2)
6) print(list1 is not list2)
7) print(list1 == list2)
```

Note: We can use is operator for address comparision where as == operator for content comparision.

2).Memebership operator:

We can use membership operators to check whether the given object present in the given collection.(Ot may be string, list, tuple, set or Dict).

in==>Returns True if the given object present in the specified collection.
not in==>Returns True if the given object not present in the specified collection.

Ex:

```
1) >>> list=[10,20,30,40]
2) >>> print(10 in list)==>True
3) >>> print(50 in list)==>False
4) >>> print(50 not in list)==>True
```

Ex:

```
1) >>> x="python is very easy"
2) >>> print('p' in x)==>True
3) >>> print('d' not in x)==>True
4) >>> print('python' in x)==>True
```

Operator Precedence:

If multiplr operators present then which operator will be evaluated first is decided by operator precedence.
Ex:

```
1) >>> 3+10*2==>23
2) >>> (3+10)*2==>26
```

Ex:

```
1) >>> a=30
2) >>> b=20
3) >>> c=10
4) >>> d=5
5) >>> print((a+b)*c/d)==>100.0
6) >>> print((a+b)*(c/d))==>100.0
7) >>> print(a+(b*c)/d)==>70.0
```

-->The following list describes operator precedence in python.
()-->Paranthesis
****-->Exponential operator**
~,-==>Bitwise complement operator, unary minus operator

*,/,%,//==>Multiplication, division,modulo,floor division.
+-==>Addition, subtratcion
<<,>>==>Left & Right shft
&==>Bitwise and
^==>Bitwise x-or
|==>Bitwise or
>,>=,<,<=,==,!= ==>Relational operators & comparision operators
=,+=,-=,*= ==>Assignment operators
is, is not==>identity operators
in, not in==>Membership operators
not==>Logical not
and==>Logical and
or==>Logical or

Ex:

```
1) 3/2*4+3+(10/5)**3-2
2) ==>3/2*4+3+2.0**3-2
3) ==>3/2*4+3+8.0-2
4) ==>1.5*4+3+8.0-2
5) ==>6.0+3+8.0-2
6) ==>15.0
```

Mathematical Functions:(Math Module)

-->A module is a collection of functions, variables and classes etc.
-->Math is a module that contains several functions to perform mathematical operations.
-->If we want to use any module in python, first we have to import that module.
Ex: import math
-->Once we import a module then we can call any function of that module.
Ex:

```
1) import math
2) print(math.sqrt(16))
3) print(math.pi)
```

o/p: 4.0
3.14159.....

-->We can create alias name, by using "as" keyword.

Ex: import math as m

-->Once we create alias name , by using that we can access functions and variables of that module.

Ex:

```
1) import math as m
2) print(m.sqrt(16))
3) print(m.pi)
```

-->If we import a member explicitly then it is not required to use module name while accessing.

Ex:

```
1) from math import sqrt,pi
2) print(sqrt(16))
3) print(pi)
4) print(math.pi)==>NameError: name 'math' is not defined
```

-->Important functions of math module:

ceil(x), floor(y), pow(x,y),factorial(x),trunc(x),gcd(x),sin(x),cos(x),tan(x).....

-->Important variables in python:

pi==>3.14

e==>2.71

inf==>infinity

nan==>not a number

-->w.a.p to find area of circle.

```
1) from math import *
2) r=int(input("Enter Radius : "))
3) area=pi*r**2
4) print("The area of circle: ",area)
```

Input & Output streams

Reading dynamic input from the keyboard:

-->In python-2 the following 2-functions are available to read dynamic input from the keyboard.

1).raw_input()
2).input()

1).raw_input():

This function always reads the data from the keyboard in the form of string format. We have to convert that string type to our required type by using the corresponding type casting methods.

```
1) x=raw_input("Enter some value:")
2) print(type(x))==>It always print str type only for any input type
```

2).input():

input() function can be used to read data directly in our required format. We rae not required to perform type casting.
Ex:

```
1) x=input(Enter value")
2) type(x)
3) 10==>int
4) "mahesh"==>str
5) 10.5==>float
6) True==>bool
```

Note:
-->But in python-3 we have only input() method and raw_input() method is not available.

-->Python-3 input() function behaviour exactly same as raw_input() method in python-2. ie every input value is treated as str type only.
-->raw_input() function of python-2 is renamed as input() function in python-3.
Ex:

```
1) >>> x=input("Enter some value:")
2) Enter some value:10
3) <class 'str'>
4) Enter some value:True
```

```
<class 'str'>
Enter some value:1.234
<class 'str'>
```

Q: w.a.p to read 2-numbers from the keyboard and print sum.

1).

```
x=input("Enter First Number :")
y=input("Enter Second Number :")
i=int(x)
j=int(y)
print("The sum : ",i+j)
```

2).

```
x=int(input("Enter First Number :"))
y=int(input("Enter Second Number :"))
print("The sum : ",x+y)
```

3).

```
print("The sum : ",int(input("Enter First Number :"))
            +int(input("Enter Second Number :")))
```

Q: w.a.p to read employee data from the keyboard and print that data.

```
eno=int(input("Enter Employee No: "))
ename=input("Enter Employee Name: ")
esal=float(input("Enter Employee Salary: "))
eadd=input("Enter Employee Address: ")
emarried=bool(input("Enter Employee Married?[True | False]: "))
print("Confirm Information")
print("Employee No :",eno)
print("Employee Name :",ename)
print("Employee Salary :",esal)
print("Employee Address :",eadd)
print("Employee Married? :",emarried)
```

Q: w.a.p to read multiple values from the keyboard in a single line:

```
a,b=[int (x) for x in input("Enter two numbers:").split()]
print(a,b)
print("Product is :", a*b)
```

Q: w.a.p to read 3 float numbers from the keyboard with , separator and print their sum

```
1) a,b,c=[float(x) for x in input("Enter 3 float numbers :").split(",")]
2) print("The sum is :",a+b+c)
```

Note: split() function can take space as separator bydefault, But we can pass anything as separator.

eval():

-->eval function take a string and evaluate the result.

```
1) >>> x=eval("10+20+30")
2) >>>x
3) >>>60
4)
5) x=eval(input("Enter expression :"))
6) print(x)
7) Enter expression :10+2*3/4
8) 11.5
```

-->eval() can evaluate the input to list, tuple, set etc...based on provided input.
Ex: w.a.p to accept list from the keyboard and diplsy.

```
1) l=eval(input("Enter list :"))
2) print(type(l))
3) print(l)
```

o/p: Enter list :[10,20,30]
<class 'list'>
[10, 20, 30]

Q:How to read different types of data from the keyboard

```
1) a,b,c=[eval(x) for x in input("Enter 3 values :").split(",")]
2) print(type(a))
3) print(type(b))
4) print(type(c))
5) o/p:Enter 3 values :10,True,10.3
6) <class 'int'>
7) <class 'bool'>
8) <class 'float'>
```

Note:
-->split() always accept sting only as a separator.
-->If we are not using eval() then it is treated as str.

Ex:

```
1) a,b,c=[eval(x) for x in input("Enter 3 values :").split("5")]
2) print(type(a))
3) print(type(b))
4) print(type(c))
```

o/p: Enter 3 values :105True510.3
<class 'int'>
<class 'bool'>
<class 'float'>

Command Line Arguments:

-->argv is not an array it is a list. It is available in sys module.
-->The arguments which are passing at the time of execution are called as command line arguments.
ex: D:\pythonclasses>py test.py 10 20 30
-->Within the python program this command line arguments are available in argv. which is present in sys module.

Note: argv[0] represents name of the program, but not the first command line argument. argv[1] represents First command line argument.

Program: To check type of argv from sys.

```
1) from sys import argv
2) print(type(argv))
```

o/p: D:\pythonclasses>py test.py 10 20
<class 'list'>

Ex: w.a.p to display command line arguments

```
1) from sys import argv
2) print("The Number of command line arguments :",len(argv))
3) print("The list of command line arguments :",argv)
4) print("Command line arguments one by on :")
5) for x in argv:
6)     print(x)
```

Ex: Sum of the arguments.

```
1) from sys import argv
2) sum=0
3) args=argv[1:]
4) for x in args:
5)     n=int(x)
6)     sum=sum+n
7)     print("The sum is :",sum)
```

Note: Usually space is separator between command line arguments. If our command line arguments itself contains space then we should enclose within double quotes(but not in single quote).

```
1) from sys import argv
2) print(argv[1])
```

o/p: D:\pythonclasses>py test.py sunny leone
sunny

```
1) from sys import argv
2) print(argv[1])
```

o/p: D:\pythonclasses>py test.py 'sunny leone'
'sunny

```
1) from sys import argv
2) print(argv[1])
```

o/p: D:\pythonclasses>py test.py "sunny leone"
sunny leone

Note: Within the python program command line arguments are available in the string form. Based on our requirement, we can convert into corresponding type by using type casting methods.
Ex:

```
1) from sys import argv
2) print(argv[1]+argv[2])
3) print(int(argv[1])+int(argv[2]))
```

o/p: D:\pythonclasses>py test.py 10 20
1020
30

Note: If we are trying to access command line arguments with out of range index then we will get an error.
Ex:

```
1) from sys import argv
2) print(argv[100])
```

o/p: D:\pythonclasses>py test.py 10 20 30 40
IndexError: list index out of range

Output Streams:

We can use print() function to display output.

Form-1: print() without argument

just it prints new line character.

Form-2: print(String)

print("Hello world")-->We can use escape characters also

print("Hello \n world")
print("Hello \t world")
-->We can use repetetion operator(*) in the string

print(10*"hello")
print("hello"*10)

-->We can use + operator also
print("Hello" + "World")

Note:
-->If both arguments are string type then + operator acts as concatination operator.
-->If one argument is string type and second is any other type like int then we eill get an error.
-->If both arguments are number type then + operator avts as arithmetic addition operator.

Q: what is diff between
print("Hello"+"world")==>Helloworld
print("Hello","world")==>Hello word

Form-3: print() with variable number of arguments

Ex:

```
1) a,b,c=10,20,30
2) print("The values are: ",a,b,c)
```

o/p: The values are: 10 20 30
-->Bydefault output values are separated by space. If we want we can specify separator by using "sep" attribute.

Ex:

```
1) a,b,c=10,20,30
2) print(a,b,c,sep=',')
3) print(a,b,c,sep=':')
```

o/p:D:\pythonclasses>py test.py
10,20,30
10:20:30

Form-4:print() with end attribute

```
1) print("Hello")
2) print("Durga")
3) print("Soft")
```

o/p: Hello
Durga
Soft

-->If we want output in the same line with space

```
1) print("Hello",end=' ')
2) print("Durga",end=' ')
3) print("Soft")
```

o/p:D:\pythonclasses>py test.py
Hello Durga Soft

Note: The default value of end attribute is \n, which is nothing but new line character.

Form-5:print(object) statement

-->We can pass any object(like list, tuple, set etc) as argument to the print statement.

Ex:

```
1) l=[10,20,30,40]
2) t=(10,20,30,40)
3) print(l)
4) print(t)
```

Form-6:print(String, variable list):

-->We can use print() statement with string and any number of arguments.

Ex:

```
1) s="Mahesh"
2) a=50
3) s1="Selenium"
4) s2="Python"
5) print("Hello",s,"Your age is :",a)
6) print("You are teaching", s1,"and", s2)
```

o/p: Hello Mahesh Your age is : 50
You are teaching Selenium and Python

Form-7:print(formatted string):

%i====>int
%d===>int
%f===>float
%s===>string type

syn: print("formatted string" %(variable list))

Ex:

```
1) a=10
2) b=20
3) c=30
4) print("a value is %d" %a)
5) print("b value is %d and c value is %d" %(b,c))
```

o/p: a value is 10
b value is 20 and c value is 30

Ex:

```
1) s="Mahesh"
2) list[10,20,30,40]
3) print("Hello %s.....The list of values are %s" %(s,list))
```

o/p: Hello Mahesh.....The list of values are [10, 20, 30, 40]

Form-8:print() with replacement operator { }

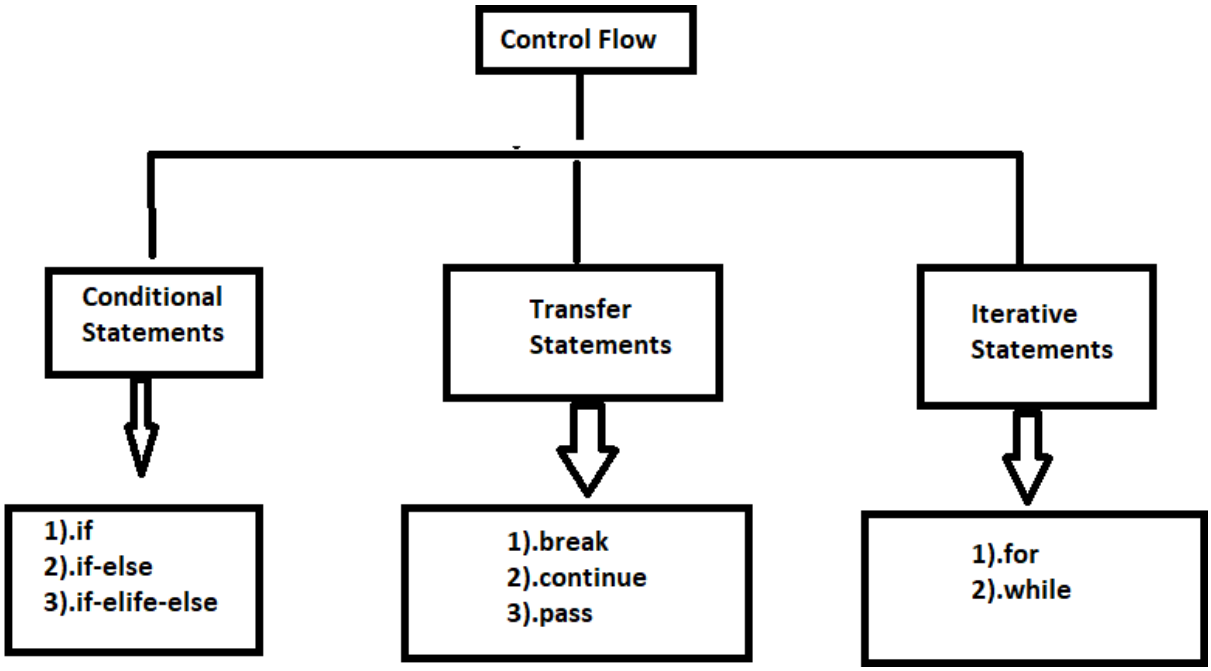
Ex:

```
1) name="Mahesh"
2) salary=10000
3) gf="sunny"
4) print("Hello {0} your salary {1} and yor girl friend {2} is waiting".format(name,salar
   y,gf))
5) print("Hello {} your salary {} and yor girl friend {} is waiting".format(name,salary,gf
   ))
6) print("Hello {x} your salary {y} and yor girl friend {z} is waiting".format(z=gf,x=nam
   e,y=salary))
```

o/p: Hello Mahesh your salary 10000 and yor girl friend sunny is waiting
Hello Mahesh your salary 10000 and yor girl friend sunny is waiting
Hello Mahesh your salary 10000 and yor girl friend sunny is waiting

FLOW CONTROL

-->Flow control describes the order in which statements will be executed at runtime.



1).Conditional Statements:

==========

1).if:

```
        if condition: statement
```

or

```
        if condition:
           statement-1
           statement-1
           statement-1
```

-->If condition is true then statements will be executed.

Ex:

```
1) name=input("Enter Name:")
2) if name=="mahesh":
3)       print("Hello mahesh good morning")
4) print("How r you!!!")
```

o/p: D:\pythonclasses>py test.py

Enter Name:mahesh
Hello mahesh good morning
How r you!!!

o/p: D:\pythonclasses>py test.py
Enter Name:durga
How r you!!!

2).if-else:

```
if condition:
   Action-1
else:
   Action-2
```

-->If condition is true Action-1 will be executed otherwise Axtion-2 will be executed.
Ex:

```
1) name=input("Enter Name:")
2) if name=="mahesh":
3)     print("Hello mahesh good morning")
4) else:
5)     print("Hello guest good morning")
6) print("How r you!!!")
```

o/p: D:\pythonclasses>py test.py
Enter Name:mahesh
Hello mahesh good morning
How r you!!!

o/p: D:\pythonclasses>py test.py
Enter Name:durga
Hello guest good morning
How r you!!!

3).if-elif-else:
syn:

```
if condition1:
   Action-1
elif condition-2:
  Action-2
elif condition-3:
  Action-3
else
  Default Action
```

-->Based on condition the corresponding action will be executed.

Ex:

```
1) brand=input("Enter your favourite Brand:")
2) if brand=="RC":
3)   print("It is a childrens brand")
4) elif brand=="KF":
5)    print("It is not much that kick")
6) elif brand=="FO":
7)    print("Buy one get one free")
8) else:
9)    print("Other brands are not recommended")
```

```
o/p:D:\pythonclasses>py test.py
Enter your favourite Brand:RC
It is a childrens brand

D:\pythonclasses>py test.py
Enter your favourite Brand:KF
It is not much that kick

D:\pythonclasses>py test.py
Enter your favourite Brand:FO
Buy one get one free

D:\pythonclasses>py test.py
Enter your favourite Brand:karjura
Other brands are not recommended
```

1).else part is always optional

Hence the following are various possible syntaxex

-->if

-->if-else

-->if-elif-else

-->if-elife

Q: w.a.p to find biggest of given 2 numbers from the command prompt?

```
1) n1=int(input("Enter First Number:"))
2) n2=int(input("Enter Second Number:"))
3) if n1>n2:
4)    print("Biggest Number is:",n1)
5) else:
```

```
6)     print("Biggest Number is:",n2)
```

Q: w.a.p to find biggest of given 3 numbers from the command prompt?

```
1)  n1=eval(input("Enter First Number:"))
2)  n2=eval(input("Enter Second Number:"))
3)  n3=eval(input("Enter Third Number:"))
4)  if n1>n2 and n1>n3:
5)     print("Biggest Number is:",n1)
6)  elif n2>n3:
7)     print("Biggest Number is:",n2)
8)  else :
9)     print("Biggest Number is:",n3)
```

```
o/p:D:\pythonclasses>py test.py
Enter First Number:10
Enter Second Number:40
Enter Third Number:20
Biggest Number is: 40

D:\pythonclasses>py test.py
Enter First Number:10.6
Enter Second Number:2.6
Enter Third Number:17.3
Biggest Number is: 17.3

D:\pythonclasses>py test.py
Enter First Number:"mahesh"
Enter Second Number:"durga"
Enter Third Number:"sunny"
Biggest Number is: sunny
```

Q: w.a.p to check whether the given number is in between 1 -10?

```
1)  n=int(input("Enter Number:"))
2)  if n>=1 and n<=10:
3)     print("The number",n,"is in between 1 to 10")
4)  else:
5)     print("The number",n,"is not in between 1 to 10")
```

Q: w.a.p to take a single digit number from the keyboard and print value in english word?

0==>ZERO
3==>THREE

```
1)  n=int(input("Enter Number:"))
2)  if n==0:
3)      print("ZERO")
4)  elif n==1:
5)      print("ONE")
6)  elif n==2:
7)      print("TWO")
8)  elif n==3:
9)      print("THREE")
10) elif n==4:
11)     print("FOUR")
12) elif n==5:
13)     print("FIVE")
14) elif n==6:
15)     print("SIX")
16) elif n==7:
17)     print("SEVEN")
18) elif n==8:
19)     print("EIGHT")
20) elif n==9:
21)     print("NINE")
22) else:
23)     print("Please enter a digit from 0-9")
```

Iterative Statements:

==========

-->If we want to execute a group of statements multiple times then we should go for iterative statements.

-->Python supports 2-types of iterative statements.

1). for loop

2).while loop

1).for loop:

-->If we want to execute some action for every element present in some sequence(IT may be string or collection) then we should go for for loop.

Syn:

```
for x in sequence:
    Body
```

Ex: To print the characters present in the given string.

```
1) s="Sunny Leone"
2) for x in s:
3)     print(x)
```

Ex:To print characters present in string index wise.

```
1) s="Sunny Leone"
2) i=0
3) for x in s:
4)     print("The character present at ",i, "index",x)
5)     i=i+1
```

o/p:

```
D:\pythonclasses>py test.py
The character present at  0 index S
The character present at  1 index u
The character present at  2 index n
The character present at  3 index n
The character present at  4 index y
The character present at  5 index
The character present at  6 index L
The character present at  7 index e
The character present at  8 index o
The character present at  9 index n
The character present at  10 index e
```

Ex: To print Hello 10 times.

```
1) for x in range(10):
2)     print("Hello")
```

Ex:To display numbers from 0-10

```
1) for x in range(10):
2)     print(x)
```

Ex:To display odd numbers from 0-20

```
1) for x in range(21):
2)     if(x%2!=0):
3)         print(x)
```

(or)

```
1) for x in range(1,21,2):
2)     print(x)
```

Ex: To display numbers from 10-1 in descending order.

```
1) for x in range(10,0,-1):
2)     print(x)
```

2).while loop:

==========

-->If we want to execute a group of statements iteratively until some condition false, then we should go for while loop.

Syn:

```
while condition:
        body
```

Ex: To print numbers from 1 to 10 by using while loop

```
1. x=1
2. while x<=10:
3.         print(x)
4.         x+=1
```

Ex: To display the sum of fisrt n numbers

```
1) n=int(input("Enter Number:"))
2) sum=0
3) i=1
4) while i<=n:
5)     sum=sum+i
6)     i+=1
7) print("The sum of the first",n,"numbers is:",sum)
```

Ex: w.a.p to prompt user to enter some name until entering mahesh.

```
1) name=""
2) while name!="mahesh":
3)     name=input("Enter Name:")
4) print("Thanks for confirmation")
```

Ex:w.a.p to prompt user to enter name & pwd until mahesh & python

```
1) username=""
2) password=""
3) while (username!="mahesh") or (password!="python"):
```

```
4)     username=input("Enter correct username :")
5)     password=input("Enter correct pwd:")
6)  print("Thanks for confirmation")
```

Infinite Loops:

==========

```
1)  i=0
2)  while True:
3)     i=i+1
4)     print("Hello",i)
```

Nested Loops:

==========

-->Sometimes we can take a loop inside anopther loop, which are also known as nested loops

Ex:

```
1)  for i in range(5):
2)    for j in range(4):
3)     print("i=",i," j=",j)
```

Alternative way:

```
1)  for i in range(5):
2)    for j in range(4):
3)      print("i={} and j={}".format(i,j))
```

-->w.a.p to display *'s in right angled triangle form

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

Ex:

```
1)  n=int(input("Enter some number:"))
2)  for i in range(1,n+1):
3)       for j in range(1,i+1):
4)            print("*",end=" ")
5)                print()
```

Alternative way:

```
1)  n=int(input("Enter some number:"))
2)      for i in range(1,n+1):
3)   print("* "*i)
```

-->w.a.p to display *'s as in below format

num:1

*

num:2

* *

* *

num:3

* * *

* * *

* * *

Ex:

```
1)  n = int(input("Enter some number:"))
2)      for i in range(n):
3)           for j in range(n):
4)              print("*", end=" ")
5)       print()
```

ALternative way:

```
1)  n = int(input("Enter some number:"))
2)  for i in range(n):
3)       print("* "*n)
```

Transfer Statements:

==========

1)break:

We can use break statement inside loops to break loop execution based on some condition.

Ex:

```
1)  for i in range(10):
2)    if  I == 3:
3)   print("processing is enough...plz break")
4)   break
5)   print(i)
```

o/p: D:\pythonclasses>py test.py

0

```
1
2
processing is enough...plz break
```

Ex:

```
1)  cart=[10,20,600,60,70]
2)  for item in cart:
3)      if item>500:
4)  print("To place this order insurance must be required")
5)  break
6)      print(item)
```

```
o/p:    D:\pythonclasses>py test.py
        10
        20
        To place this order insurance must be required
```

2).continue:

We can use continue statement to skip current iteration and continue next iteration.

Ex: To print odd numbers in the range 0 to 9.

```
1)  for i in range(10):
2)   if i%2==0:
3)       continue
4)   print(i)
```

```
o/p:    D:\pythonclasses>py test.py
        1
        3
        5
        7
        9
```

Ex:

```
1)  cart=[10,20,600,60,70,550,40]
2)   for item in cart:
3)       if item>500:
4)           print("We can't process this item: ",item)
5)           continue
6)       print(item)
```

o/p:

```
D:\pythonclasses>py test.py
10
20
We can't process this item:  600
60
70
We can't process this item:  550
40
```

Ex:

```
1)  numbers=[10,20,0,5,0,30]
2)   for n in numbers:
3)        if n==0:
4)            print("hey how we can devide with zero...just skipping")
5)            continue
6)       print("100/{}={}".format(n,100/n))
```

o/p:

```
D:\pythonclasses>py test.py
100/10=10.0
100/20=5.0
hey how we can devide with zero...just skipping
100/5=20.0
hey how we can devide with zero...just skipping
100/30=3.3333333333333335
```

loops with else block:

-->Inside loop execution, if break statement not executed, then only else part will be executed.

Ex:

```
1)  cart=[10,20,30,40,50]
2)   for item in cart:
3)       if item>500:
4)            print("We can't process this order")
5)            break
6)       print(item)
7)   else:
8)       print("Congrats.... all items processed suuccessfully")
```

o/p:

```
D:\pythonclasses>py test.py
10
20
30
40
50
Congrats.... all items processed suuccessfully
```

Ex:

```
1)  cart=[10,20,30,600,40,50]
2)   for item in cart:
3)      if item>500:
4)            print("We can't process this order")
5)            break
6)      print(item)
7)   else:
8)      print("Congrats.... all items processed suuccessfully")
```

o/p:

```
D:\pythonclasses>py test.py
10
20
30
We can't process this order
```

Q: What is difference between for loop and while loop in python?
-->We can use loops to repeat code execution.
-->Repeat code for every item in sequence==>for loop
-->Repeat code as long as condition is true==>while loop

Q:How to exit from the loop?
-->By using break statement.

Q:How to skip some iterations inside the loop?
-->BY using continue statement.

Q:When else part will be executed wrt loops?
-->If loop executed without break.

3).pass statement:

-->pass is a keyword in python.

-->In our programming syntactically if block is required which won't do anything then we can define that empty block with pass keyword.

pass:

-->It is an empty statement

-->It is null statement

-->It won't do anything

Ex:

```
if True:
```

o/P:SyntaxError: unexpected EOF while parsing

Ex: `if True:pass`

==>Valid

Ex:

```
def f1():
```

o/p:SyntaxError: unexpected EOF while parsing

Ex: `def f1():pass`

==>Valid

Use case of pass:

-->Sometimes in the parent class we have to declare a function with empty body and child class responsible to provide implementation. such type of empty body we can define by using pass keyword.(It is something like abstract method in java)

Ex:

```
1)  for i in range(100):
2)      if i%9==0:
3)          print(i)
4)      else:pass
```

o/p:

```
D:\pythonclasses>py test.py
0
9
18
27
36
45
```

54
63
72
81
90
99

del statement:

-->del is a keyword in python
-->After using a variable, it is highly recommended to delete that variable if it is no longer required, so that the corresponding object is eligible for garbage collection. We can delete variable by using del statement.

Ex:

```
1)  x=10
2)   print(x)
3)   del x
4)   print(x)
```

-->After deleting a variable we can't access that variable otherwise we will get name error
o/p:
NameError: name 'x' is not defined
Note:
We can delete variables which are pointing to immutable objects. BUt we can't delete the elements present inside the immutable object.

Ex:

```
1)  s="mahesh"
2)   del s==>Valid
3)   del s[0]==>Name Error: name 's' is not defined
```

-->But in the case of None assignment the variable won't be removed but the corresponding object is eligible for garbage collection(re bind operation). Hence after assigning with None value, we can access that variable.

Ex:

```
1)   s="mahesh"
2)   s=None
3)   print(s)==>None
```

String Data Type

The most commonly used object in any project and in any programming language is String only. Hence we should aware complete information about the string DT.

What is a String?

Any sequence of characters within either single quotes or double quotes is considered as a String.

Syn:

```
s='mahesh'
s="mahesh"
```

Note:

In most of other languages like C, C++, java a single character with in single quotes is treated as char data type value. But in python we are not having char data type. Hence it is treated as String only.

Ex:

```
1)  >>> ch='a'
2)  >>> type(ch)
3)  <class 'str'>
```

How to define multiline string literals:

We can define multi-line string literals by using triple single or double quotes.

Ex:

```
1)  >>>s='''Durga
2)   Software
3)          Solutions'''
```

-->We can also use triple quotes to use single quotes or double quotes as symbol inside the string.

Ex:

```
1)  s='This is ' a single quote symbol'==>(Invalid)
2)  s='This is \' a single quote symbol'==>(valid)
3)  s="This is ' a single quote symbol"==>(valid)
4)  s='This is " a double quote symbol'==>(valid)
5)  s='The "Python Notes" by 'mahesh' is very helpful'==>(Invalid)
6)  s="The "Python Notes" by 'mahesh' is very helpful"==>(Invalid)
7)  s='The \"Python Notes\" by \'mahesh\' is very helpful'==>(valid)
8)  s='''The "Python Notes" by 'mahesh' is very helpful'''==>(valid)
```


NARESH IT, 2nd Floor, Durga Bhavani Complex, Satyam Threater Rd, Ameerpet, HYD
Telangana-500016 | Tel:+91 40 2374 6666 | Ph:+91 90009 94007 | www.nareshit.in

How to access characters of a String:

We can access characters of a string by using the following ways.

1).By using Index

2).By using slice operator

1).By using Index:

-->Python supports both +ve and -ve index.

-->+ve index means left to right(Forward Direction)

-->-ve means right to left(Backward Direction)

Ex:

```
1)  -6- 5  -4 -3 -2-1
2)  s = 'm  a  h  e  s  h'
3)   0   1  2  3 4  5
```

Ex:

```
1)  s[0]==>m
2)  s[-1]==>h
3)  s[10]==>String index out of range.
```

Q: w.ap to accept some string from the keyboard and display its characters by index wise(both +ve and -ve)

```
1)  s=input("Enter some string:")
2)  i=0
3)  for x in s:
4)  print("The character present at Positive index {} and
5)                      Negative index {} is {}".format(i,i-len(s),x))
6)  i=i+1
```

o/p:

```
        D:\pythonclasses>py test.py
Enter some string:mahesh
The character present at Positive index 0 and Negative index -6 is m
The character present at Positive index 1 and Negative index -5 is a
The character present at Positive index 2 and Negative index -4 is h
The character present at Positive index 3 and Negative index -3 is e
The character present at Positive index 4 and Negative index -2 is s
The character present at Positive index 5 and Negative index -1 is h
```

2).Accessing characters by using slice operator:

==========

Syn:

s[Begin index:End index:step]

-->Beginindex: From where we have to consider slice(substring)

-->Endindex:We have to terminate the slice(substring) at endindex-1.

-->Step:Increment value.

Note: If we are not specifying begin index then it will consider from beginning of the string.

If we are not specifying end index then it will consider up to end of the string.

The default value for step is 1.

Ex:

```
1)  >>> s="Learning Python is very very easy"
2)  >>> s[1:7:1]    'earnin'
3)  >>> s[1:7]              'earnin'
4)  >>> s[:7]               'Learnin'
5)  >>> s[7:]               'g Python is very very easy'
6)  >>> s[:]                'Learning Python is very very easy'
7)  >>> s[::]               'Learning Python is very very easy'
8)  >>> s[::-1]             'ysae yrev yrev si nohtyP gninraeL'
```

Behaviour of slice operator:

---

s[Begin:End:step]

-->step value can be either +ve or -ve

-->If +ve then it should be forward direction(left to right) and we have to consider Begin to End-1.

-->If -ve then it should be backward direction(right to left) and we have to consider Begin to End+1.

Note:

-->In the backward direction if end value is -1 then result is always empty.

-->In the forward direction if end value is 0 then result is always empty.

In forward direction:

Default value for Begin:0

Default value for end: length of string

Default value for step:1

In backward direction:

Default value for Begin:-1

Default value for end:-(length of string+1)

Note: Either forward or backward directions, we can both +ve and -ve values for begin and end index.

Ex:

```
1)  >>> s='0123456789'
2)  >>> s[2:8:1]    '234567'
3)  >>> s[2:8:-1]   ''
4)  >>> s[-1:-6:-1] '98765'
5)  >>> s[2:-5:1]   '234'
6)  >>> s[-5:0:-9]  '5'
7)  >>> s[:0:-1]    '987654321'
```

Mathematical operators for a string:

==========

-->We can apply the following mathematical operators for a string.

1).+ operator for concatenation.

2).* operator for repetition.

```
1)  Ex:      print("Durga"+"Soft")==>DurgaSoft
2)   print("DurgaSoft"*2)==>DurgaSoftDurgaSoft
```

Note:

-->To use + operator for strings, compulsory both arguments should be str type.

-->To use * operator for strings, compulsory one argument should be str and other argument should be int.

len() in-built function:

==========

-->We can use len() function to find the number of characters present in the string.

Ex:

```
1)  s="mahesh"
2)  print(lens(s))==>6
```

Q: w.a.p to access each character of string in forward direction and backward direction by using while loop?

```
1)  s=input("Enter some string:")
2)  n=len(s)
3)  i=0
4)  print("Forward Direction:")
5)  while i<n:
6)    print(s[i],end=' ')
7)    i=i+1
8)  print()
9)  print("Backward Direction")
10) i=-1
11) while i>=-n:
12)   print(s[i],end=' ')
13)   i=i-1
14)   (OR)
15) print("Backward Direction:")
16) i=n-1
17) while i>=0:
18)   print(s[i],end=' ')
19)   i=i-1
```

Alternative ways:

```
1)  s=input("Enter some string:")
2)  print("Forward Direction:")
3)  for i in s:
4)    print(i,end=' ')
5)  print()
6)  print("Forward Direction:")
7)  for i in s[::]:
8)    print(i,end=' ')
9)  print()
10) print("Backward Direction:")
11) for i in s[::-1]:
12)   print(i,end=' ')
```

Checking Membership:

==========

-->We can check whether the character or string is the memebr of another string or not by using in and not in operators.

Ex:

```
1)  s="mahesh"
2)  print('m' in s)==>True
3)  print('z' in s)==>False
```

Program:

```
1)  s=input("Enter main string:")
2)  subs=input("Enter sub string:")
3)  if subs in s:
4)    print(subs," is found in main string")
5)  else:
6)    print(subs," is not found in main string")
```

Comparision of Strings:

==========

-->We can use comparision operators(<, <=,>,>=) and equality operators(==, !=) for strings.

-->Comparision will be performed based on alphabatical order.

Ex:

```
1)  s1=input("Enter First String:")
2)  s2=input("Enter Second String:")
3)  if s1==s2:
4)    print("Both Strings are equal")
5)  elif s1<s2:
6)    print("First string is less than second string")
7)  else:
8)    print("First string is greater than second string")
```

Ex:

```
1)  l1=["A","B","C"]
2)  l2=["A","B","C"]
3)  l3=l2
4)  print(id(l1))==>2774045844040
5)  print(id(l2))==>2774045844104
6)  print(l1 is l2)==>False
7)  print(l2 is l3)==>True
8)  print(l1 == l2)==>True
```

Removing spaces from the string:

==========

-->We can use the following 3-methods.

1).rstrip()==>To remove spaces at right hand side.

2).lstrip()==>To remove spaces at left hand side.

3).strip()==>To remove spaces both sides

Ex:

```
1)  city=input("Enter Your City:")
2)  scity=city.strip()
3)  if scity=='hyderabad':
4)    print("Hello hydrabadi...gud mng")
5)  elif scity=='chennai':
6)    print("Hello Madrasi....Vanakkam")
7)  elif scity=='bangalore':
8)    print("Hello Kannadiga...Subhodaya")
9)  else:
10)  print("Your entered city is invalid")
```

Finding Substrings:

==========

-->We can use the following 4-methods

For forward direction:

-->find()

-->index()

For backward direction:

-->rfind()

-->rindex()

1).find():

Syn:

s.find(substring)

-->Returns index of the first occurenece of the given substring. If it is not available then we will get -1.

Ex:

```
1)  s="Learning python is very easy"
2)  print(s.find("python"))
3)  print(s.find("e"))
4)  print(s.find("z"))
5)  print(s.rfind("e"))
```

```
o/p:D:\pythonclasses>py test.py
9
1
-1
24
```

Note:
Bydefault find() method can search total string. we can also specify the boundaries to search.

Syn:

```
s.find(substring,begin,end)
```

Ex:

```
1)  s="maheshdurgasunny"
2)  print(s.find("a"))
3)  print(s.find("a",5,15))
4)  print(s.find("z",5,15))
```

```
o/p:
1
10
-1
```

index():

---

index() method is exactly same as find() method except that if the specified substring is not available then we eill get ValueError.

Ex:

```
1)  s="python"
2)  s.index("z")==>ValueError
```

Ex:

```
1)  s=input("Enter some string: ")
2)  subs=input("Enter sub string: ")
```

```
3)  try:
4)          n=s.index(subs)
5)  except ValueError:
6)          print("Substring is not found in main string")
7)  else:
8)          print("Sub string is found in main string",n)
```

o/p:
D:\pythonclasses>py test.py
Enter some string: learning python is easy
Enter sub string: python
Sub string is found in main string 9

D:\pythonclasses>py test.py
Enter some string: learning python is easy
Enter sub string: java
Substring is not found in main string

Q: w.a.p to display all positions of substring in a given main string
Ex:

```
1)  s=input("Enter main string: ")
2)  subs=input("Enter sub string: ")
3)  flag=False
4)  pos=-1
5)  n=len(s)
6)  while True:
7)          pos=s.find(subs,pos+1,n)
8)          if pos==-1:
9)                  break
10)         print("Found at position", pos)
11)         flag=True
12) if flag==False:
13)         print("Not found")
```

o/p:D:\pythonclasses>py test.py
Enter main string: abbabcabbabdefa
Enter sub string: a
Found at position 0
Found at position 3
Found at position 6
Found at position 9
Found at position 14

Counting substring the given string:

==========

-->We can find the number of occurances of substring present in the given string by using count() method.

1).s.count(substring)==>It will search through out the string

2).s.count(substring,begin,end)==>It will search from begin index to end-1 index.

ex:

```
1)  s="abababababaababa"
2)  print(s.count("a"))
3)  print(s.count("ab"))
4)  print(s.count("a",3,10))
```

```
o/p:D:\pythonclasses>py test.py
8
6
4
```

Replacing a string with another string:

==========

Syn:

```
s.replace(oldstring,newstring)
```

-->Inside s, every occurance of old string will be replaced with new string.

Ex:

```
1)  s="Learning python is difficult"
2)  s1=s.replace("difficult","easy")
3)  print(s1)
```

```
o/p:D:\pythonclasses>py test.py
Learning python is easy
```

Ex: All occurances will be replaced.

```
1)  s="abababababaababa"
2)  s1=s.replace("a","b")
3)  print(s1)
```

```
o/p:D:\pythonclasses>py test.py
bbbbbbbbbbbbb
```

Q:String objects are immutable then how we can change the content by using replace() method?

Once we creates string object, we can't change the content. This non changable behaviour is nothing but immutability. If we are trying to change the content by using any

method, then with those changes a new object will be created and changes won't be happened in existing object.

Hence with replace() method also a new object got created but existing object won't be changed.

Ex:

```
1)  s="abab"
2)  s1=s.replace("a","b")
3)  print(s, " is available at: ",id(s))
4)  print(s1, " is available at: ",id(s1))
```

```
o/p:D:\pythonclasses>py test.py
abab  is available at:  2086080298880
bbbb  is available at:  2086080295296
```

-->In the above example, original object is available and we can see new object which was created because of replace() method.

Splitting of string:

==========

-->We can split the given string according to specified separator by using split() method.

Syn: l=s.split(separator)

-->The default separator is space. The return type of split() method is list.

Ex:

```
1)  s="Durga Software Solutions"
2)  l=s.split()
3)  print(l)
4)  for x in l:
5)          print(x)
```

```
o/p:D:\pythonclasses>py test.py
['Durga', 'Software', 'Solutions']
Durga
Software
Solutions
```

Ex:

```
1)  s="29-09-2018"
2)  l=s.split('-')
3)  for x in l:
4)          print(x)
```

```
o/p:D:\pythonclasses>py test.py
29
```

09
2018

Joining of Strings:
==========
-->We can join a group of strings (list or tuple) wrt the given separator.
Syn: s=separator.join(group of strings)

Ex:

```
1) t=('sunny','bunny','chinny')
2) s='-'.join(t)
3) print(s)
```

o/p:sunny-bunny-chinny
Ex:

```
1) l=['hyderabd','bangalore','chennai','dubai']
2) s=':'.join(l)
3) print(s)
```

o/p:hyderabd:bangalore:chennai:dubai

Changing Case of a string:
==========
-->We can change case of a string by using the following 5-methods.
1).upper(): To convert all characters to upper case
2).lower(): To convert all characters to lower case
3).swapcase(): Converts all lower case characters to upper case and all upper case characters to lower case.
4).title(): To convert all the characters to title case. i.e first character in every word should be upper case and all remaining characters should be lower case.
5).capitalize(): Only first character will be converted to upper case and all remaining characters can be converted to lower case.
Ex:

```
1) s='learning Python is very Easy'
2) print(s.upper())
3) print(s.lower())
4) print(s.swapcase())
5) print(s.title())
6) print(s.capitalize())
```

o/p:D:\pythonclasses>py test.py
LEARNING PYTHON IS VERY EASY

learning python is very easy
LEARNING pYTHON IS VERY eASY
Learning Python Is Very Easy
Learning python is very easy

Checking Starting and ending part of the string:

==========

-->Python contains the following methods for this purpose.

1).s.startswith(substring)

2).s.endswith(substring)

Ex:

```
1)  s='learning Python is very easy'
2)  print(s.startswith("learning"))
3)  print(s.endswith("learning"))
4)  print(s.endswith("easy"))
```

o/p:D:\pythonclasses>py test.py
True
False
True

To check type of characters present in a string:

==========

1).isalnum():Returns True if all the characters are alphanumerics(a-z, A-Z, 0-9)

2).isalpha():Returns True if all the characters are only alphabate symbols(a-z, A-Z)

3).isdigit():Returns True if all the characters are digits only(0-9)

4).islower():Returns True if all the characters are lower case alphabet symbols.

5).isupper():Returns True if all the characters are upper case alphabet symbols.

6).istitle():Returns True if the string is in title case

7).isspace():Returns True if string contains only spaces.

Ex:

```
1)  print('Mahesh3333'.isalnum())#True
2)  print('Mahesh3333'.isalpha())#False
3)  print('Mahesh'.isalpha())#True
4)  print('Mahesh'.isdigit())#False
```

```
5)  print('123456'.isdigit())#True
6)  print('abc'.islower())#True
7)  print('abc123'.islower())#True
8)  print('ABC'.isupper())#True
9)  print('Learning Python is Easy'.istitle())#False
10) print('Learning Python Is Easy'.istitle())#True
11) print(' '.isspace())#True
```

Demo Program:

==========

```
1)  s=input("Enter any character:")
2)  if s.isalnum():
3)    print("Alpha Numeric Character")
4)    if s.isalpha():
5)          print("Alphabet Character")
6)          if s.islower():
7)                print("Lower case alphabet character")
8)          else:
9)                print("Upper case alphabet character")
10) else:
11)         print("It is a digit")
12) elif s.isspace():
13)  print("It is space character")
14) else:
15)  print("It is a special character")
```

```
o/p:D:\pythonclasses>py test.py
Enter any character:9
Alpha Numeric Character
It is a digit

D:\pythonclasses>py test.py
Enter any character:a
Alpha Numeric Character
Alphabet Character
Lower case alphabet character

D:\pythonclasses>py test.py
Enter any character:A
Alpha Numeric Character
Alphabet Character
Upper case alphabet character
```

```
D:\pythonclasses>py test.py
Enter any character:
It is space character

D:\pythonclasses>py test.py
Enter any character:#
It is a special character
```

Formatting The string:

==========

-->We can format the string with variable values by using replacement operator{ } and format() method.

Ex:

```
1)  name="Mahesh"
2)  salary=10000
3)  age=50
4)  print("{}'s salary is {} and his age is {}".format(name,salary,age))
5)  print("{0}'s salary is {1} and his age is {2}".format(name,salary,age))
6)  print("{x}'s salary is {y} and his age is {z}".format(z=age,y=salary,x=name))
```

o/p:

```
Mahesh's salary is 10000 and his age is 50
Mahesh's salary is 10000 and his age is 50
Mahesh's salary is 10000 and his age is 50
```

Important programs regrading string concept:

==========

Q: w.a.p to reverse the given string.

1st way:

```
1)  s=input("Enter some string: ")
2)  print(s[::-1])
```

2nd way:

```
1)  s=input("Enter some string: ")
2)  for x in reversed(s):
3)    print(x,end='')
```

3rd way:

```
1) s=input("Enter some string: ")
2) print(''.join(reversed(s)))
```

4th way:

```
1) s=input("Enter some string: ")
2) i=len(s)-1
3) target=''
4) while i>=0:
5)   target=target+s[i]
6)   i=i-1
7) print(target)
```

Q: w.a.p to reverse order of words
I/P: Learning Python Is Very Easy!!
O/P:Easy!! Very Is Python Learning

```
1) s=input("Enter some string:")
2) l=s.split()
3) l1=[]
4) i=len(l)-1
5) while i>=0:
6)   l1.append(l[i])
7)   i=i-1
8) output=' '.join(l1)
9) print(output)
```

Q:w.a.p to reverse internal content of each word.
i/p: Durga Software Solutions
o/p: agruD erawtfoS snoituloS

```
1) s=input("Enter Some String: ")
2) l=s.split()
3) print(l)
4) l1=[]
5) i=0
6) while i<len(l):
7)   l1.append(l[i][::-1])
8)   i=i+1
9) print(' '.join(l1))
```

Q: w.a.p to print characters at odd position ans even position for the given string.

1stway:

```
1) s=input("Enter Some String: ")
2) print("Characters at Even Position: ",s[0::2])
3) print("Characters at Odd Position: ",s[1::2])
```

2nd way:

```
1) s=input("Enter Some String: ")
2) i=0
3) print("Character at Even Position:")
4) while i<len(s):
5)   print(s[i],end=',')
6)   i=i+2
7) print()
8) i=1
9) print("Character at Odd Position:")
10) while i<len(s):
11)  print(s[i],end=',')
12)  i=i+2
```

Q: w.a.p to merge characters of 2-strings into a single string by taking characters alternatively.

```
1) s1='mahesh'
2) s2='durga'
3) o/p:mdauhregsah
4)
5) s1=input("Enter First String: ")
6) s2=input("Enter Second String: ")
7) output=''
8) i,j=0,0
9) while i<len(s1) or j<len(s2):
10)  if i<len(s1):
11)          output=output+s1[i]
12)          i=i+1
13)  if j<len(s2):
14)          output=output+s2[j]
15)          j+=1
16) print(output)
```

Q: w.a.p to sort the characters of the string and first alphabet symbols followed by numeric values.
i/p: B4A1D3
o/p: ABD134

```
1)  s=input("Enter Some String: ")
2)  s1=s2=output=''
3)  for x in s:
4)    if x.isalpha():
5)              s1=s1+x
6)    else:
7)              s2=s2+x
8)  for x in sorted(s1):
9)    output=output+x
10) for x in sorted(s2):
11)  output=output+x
12) print(output)
```

Q: w.a.p for the following requirement.
i/p: a4b3c2
o/p:aaaabbbcc

```
1)  s=input("Enter Some String: ")
2)  output=''
3)  for x in s:
4)    if x.isalpha():
5)              output=output+x
6)              previous=x
7)    else:
8)              output=output+previous*(int(x)-1)
9)  print(output)
```

Q: w.a.p to perfrom the following activity
i/p:a4k3b2
o/p:aeknbd

```
1)  s=input("Enter Some String: ")
2)  output=''
3)  for x in s:
4)    if x.isalpha():
5)              output=output+x
6)              previous=x
7)    else:
8)              output=output+chr(ord(previous)+int(x))
```

```
9)  print(output)
```

Q:w.a.p to remove the duplicate characters from the given input string?
i/p:ABCDCDEADDB
o/p:ABCDE

1way:

```
1)  s=input("Enter Some String: ")
2)  l=[ ]
3)  for x in s:
4)   if x not in l:
5)              l.append(x)
6)  print(''.join(l))
```

2nd way:

```
1)  s=input("Enter Some String:")
2)  print(''.join(set(s)))
```

Q:w.a.p to find the occurences of each character present in the given string.
i/p:ABCDCBAB
o/p:A=2, B=3, C=2, D=1

```
1)  s=input("Some String:")
2)  d={}
3)  for x in s:
4)   if x in d.keys():
5)              d[x]=d[x]+1
6)   else:
7)              d[x]=1
8)  for k,v in d.items():
9)   print("{}={}".format(k,v))
```

Ex: Print the keys and corresponding values from dict

```
1)  d={100:'mahesh',200:'durga',300:'sunny'}
2)  print(d.keys())
3)  for k,v in d.items():
4)   print("{}={}".format(k,v))
```

List Data Structure

-->If we want to represent a group of individual objects as a single entity where insertion order preserved and duplicates are allowed, then we should go for list.
-->Insertion order is preserved.
-->Duplicates objects are allowed.
-->Heterogeneous objects are allowed.
-->List is dynamic because based on our requirement we can increase the size and decrease the size.
-->In the list elements will be palced with in square brackets and with comma separator.
-->We can differentiate duplicate elements by using index and we can preserve insertion order by using index.Hence index will play very important role.
-->Python supports both +ve and -ve indexes. +ve index means from left to right where as -ve index means right to left.

Ex:

```
1)  l=[10,"A","B",20,30,True]
2)
3)          -6  -5 -4 -3  -2   -1
4)          10  A  B  20  30  True
5)          0   1  2  3   4    5
```

-->List objects are mutable. i.e we can change the content.

Creation of list objects:
==========
1).We can create empty list objects as follows.....

```
1)  list=[ ]
2)  print(list)
3)  print(type(list))
```

2).If we know elements already then we can create list as follows.

```
1)  list=[10,20,30,40]
```

3).With dynamic input:

```
1)  list=eval(input("Enter List:"))
2)  print(list)
3)  print(type(list))
```


NARESH IT, 2nd Floor, Durga Bhavani Complex, Satyam Threater Rd, Ameerpet, HYD
Telangana-500016 | Tel:+91 40 2374 6666 | Ph:+91 90009 94007 | www.nareshit.in

4).with list() function:

```
1)  l=list(range(10))
2)  print(l)
3)  print(type(l))
```

Ex:

```
1)  s="mahesh"
2)  l=list(s)
3)  print(l)
```

5).with split() function:

```
1)  s="Learning Python Is Very Easy"
2)  l=s.split()
3)  print(type(l))
4)  print(l)
```

Note:

Sometimes we can create list inside another list, such type of lists are called as nested lists.

Ex: [10,20,[30,40]]

Accessing elements of List:

-->We can access elements of the list either by using index or by using slice operator(:)

1).By using Index:

-->List follows zero based index. i.e index of first element is zero.

-->List supports both +ve & -ve indexes. +ve(left to right). -ve(Right to left)

```
1)  list=[10,20,30,40]
2)
3)  print(list[0])==>10
4)  print(list[-1])==>40
5)  print(list[10])==>List index out of range
```

2).By using slice operator:

Syn:

list2=list1[start:stop:step]

start-->It indicates the index where slice has to start.

default value is 0.

stop-->It indicates the index where slice has to end.
default value is max alloed index of lisr i.e length of the list.
step-->increment value
default value is 1.
Ex:

```
1)  n=[1,2,3,4,5,6,7,8,9,10]
2)  print(n[2:7:2])
3)  print(n[4::2])
4)  print(n[3:7])
5)  print(n[8:2:-2])
6)  print(n[4:100])
```

o/p:D:\pythonclasses>py test.py
[3, 5, 7]
[5, 7, 9]
[4, 5, 6, 7]
[9, 7, 5]
[5, 6, 7, 8, 9, 10]

List Vs mutability:
==========
-->Once we create a list object, we can modify its content. Hence list objects are mutable.

Ex:

```
1)  n=[10,20,30,40]
2)  print(n)
3)  n[1]=333
4)  print(n)
```

o/p:[10,20,30,40]
[10,333,30,40]

Traversing the elements of list:

-->The sequential access of each element in the list is called as traversal.

1).By using while loop:

```
1)  n=[0,1,2,3,4,5,6,7,8,9,10]
2)  i=0
3)  while i<len(n):
```

```
4)    print(n[i])
5)    i=i+1
```

2).By using for loop:

```
1)  n=[0,1,2,3,4,5,6,7,8,9,10]
2)  for i in n:
3)    print(i)
```

3).To display only even numbers:

```
1)  n=[0,1,2,3,4,5,6,7,8,9,10]
2)  for i in n:
3)    if i%2==0:
4)              print(i)
```

4).To display elements by index wise:

```
1)  l=["A","B","C"]
2)  x=len(l)
3)  for i in range(x):
4)    print(l[i], "is available at +ve index: ",i,"and at -ve index: ",i-x)
```

```
o/p:D:\pythonclasses>py test.py
A is available at +ve index:  0 and at -ve index:  -3
B is available at +ve index:  1 and at -ve index:  -2
C is available at +ve index:  2 and at -ve index:  -1
```

Important functions of List:

==========

1).To get the information about list:

1.len():

Returns number of elements present in the list.

Ex:

```
1)  l=[10,20,30,40]
2)  len(l)==>4
```

2.count():

It returns the number of occurences of specified item in the list

Ex:

```
1)  n=[1,2,2,2,2,3,3]
2)  print(n.count(1))==>1
3)  print(n.count(2))==>4
4)  print(n.count(3))==>2
```

3.index():

It returns index of first occurance of the specified element.

Ex:

```
1)  l=[1,2,2,3,3,4]
2)  print(l.index(1))==>0
3)  print(l.index(2))==>1
4)  print(l.index(3))==>3
5)  print(l.index(4))==>5
```

Note:

If the specified element not present in the list then we will get ValueError. Hence before index() method we have to check whether item present in the list or not by using in operator.

2).Manipulating elements of list:

==========

1). append() function:

We can use append() function to add item at the end of the list.

Ex:

```
1)  >>> l=[ ]
2)  >>> l    [ ]
3)  >>> l.append("A")
4)  >>> l.append("B")
5)  >>> l.append("C")
6)  >>> l    ['A', 'B', 'C']
```

Ex: To add all elements to list upto 100 which are divisible by 10

```
1)  list=[ ]
2)  for i in range(101):
3)   if i%10==0:
4)              list.append(i)
5)  print(list)
```

o/p:D:\pythonclasses>py test.py

[0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100]

2).insert() function:

To insert an item at specified index position.

Ex:

```
1)  >>> n=[1,2,3,4]
2)  >>> n.insert(1,333)
3)  >>> n   [1, 333, 2, 3, 4]
```

Ex:

```
1)  >>> n=[1,2,3,4]
2)  >>> n.insert(10,777)
3)  >>> n.insert(-10,999)
4)  >>>print(n)     [999, 1, 2, 3, 4, 777]
```

Note:

If the specified index is greater than max index then element will be inserted at last position. If the specified index is smaller than min index then element will be inserted at first position.

Difference between append() and insert():

append():

In list when we add any element it will come in last. i.e it will be last element.

insert():

In list we can insert any element in a particular index number.

3).extend() function:

-->To add all items of one list to another list.

Syn: l1.extend(l2)

-->All items present in l2 will be added to l1.

Ex:

```
1)  order1=["Chicken","Mutton","Fish"]
2)  order2=["RC","KF","FO"]
3)  order1.extend(order2)
4)  print(order1)
```

o/p:D:\pythonclasses>py test.py
['Chicken', 'Mutton', 'Fish', 'RC', 'KF', 'FO']

NARESH IT, 2nd Floor, Durga Bhavani Complex, Satyam Threater Rd, Ameerpet, HYD
Telangana-500016 | Tel:+91 40 2374 6666 | Ph:+91 90009 94007 | www.nareshit.in

Ex:

```
1)  order=["Chicken","Mutton","Fish"]
2)  order.extend("mushroom")
3)  print(order)
```

```
o/p:D:\pythonclasses>py test.py
['Chicken', 'Mutton', 'Fish', 'm', 'u', 's', 'h', 'r', 'o', 'o', 'm']
```

4).remove() function:

We can use this function to remove specified item from the list. If the item present multiple times then only first occurence will be removed.

Ex:

```
1)  n=[10,20,30,10,30]
2)  n.remove(10)
3)  print(n)
```

```
o/p:D:\pythonclasses>py test.py
[20, 30, 10, 30]
```

-->If the specified item not present in the list we will get ValueError

Ex:

```
1)  n=[10,20,30]
2)  n.remove(50)
3)  print(n)
```

```
o/p:ValueError:list.remove(x):x not in list
```

Note:

Hence before using remove() method first we have to check specified element present in the list or not by using in operator.

Ex:

```
1)          l=[10,20,30,40,10,30]
2)          x=int(input("Enter element to be removed:"))
3)          if x in l:
4)                  l.remove(x)
5)                  print("Removed successfully...")
6)          else:
7)                  print("Specified element is not available")
```


NARESH IT, 2nd Floor, Durga Bhavani Complex, Satyam Threater Rd, Ameerpet, HYD
Telangana-500016 | Tel:+91 40 2374 6666 | Ph:+91 90009 94007 | www.nareshit.in

5).pop() function:

-->It removes and returne the last element of the list.

-->This is only the function which manipulates list and returns some element.

Ex:

```
1)  n=[10,20,30,40]
2)  print(n.pop())
3)  print(n.pop())
4)  print(n)
```

```
o/p:D:\pythonclasses>py test.py
40
30
[10, 20]
```

-->If the list is empty then pop() function raises indexerror.

Ex:

```
1)  n=[ ]
2)  print(n.pop())
```

```
o/p:IndexError: pop from empty list
```

Note:

1.pop() is the only function which manipulates the list and returns some value.

2.In general we can use append() and pop() functions to implement stack datastructure by using list, which follows LIFO(Last In First Out) order.

-->In general we can use pop() function to remove last element of the list, but we can use to remove elements based on index.

n.pop(index)==>To remove and return element present at specified index.

n.pop()==>To remove and return last element of the list

Ex:

```
1)      n=[10,20,30,40,50,60]
2)      print(n.pop())#60
3)      print(n.pop(1))#20
4)      print(n.pop(10))
```

```
o/p:D:\pythonclasses>py test.py
60
20
IndexError: pop index out of range
```

Differences between remove() and pop():

remove()	pop()
1.We can use to remove specified element from the list.	1.We can use to remove last element from the list.
2.It can't return any value.	2.It returned removed element.
3.If specified element not available then we wilget ValueError.	3.If list is empty then we will get an error.IndexError.

Note:

List objects are dynamic i.e based on our requirement we can increase and decrease the size.

append(),insert(),extend()==>For increasing size/growable nature

remove(),pop()==>For decreasing size/shrinking nature

3).Ordering elements of the List:

==========

1.reverse():

We can use reverse() order of elements of list.

Ex:

```
1)  n=[10,20,30,40]
2)  n.revrse()
3)  print(n)
```

o/p: [40,30,20,10]

2.sort() function:

In list by default insertion order is preserved. If we want to sort the elements of list according to default natural sorting order then we should go for sort() method.

For Numbers==>default sording order is ascending order.

For Strings==>default sorting order is alphabatical order.

Ex:

```
1)  n=[20,10,40,30]
2)  n.sort()
3)  print(n)
```

o/p:[10,20,30,40]

Ex:

```
1)      n=["Banana","Cat","Apple"]
2)      n.sort()
3)      print(n)
```

o/p:['Apple', 'Banana', 'Cat']

Note: To use sort() function, compulsory list should contain only homogeneous elemets otherwise we will get typeerror.

Ex:

```
1)      n=[20,10,"B","A"]
2)      n.sort()
3)      print(n)
```

o/p:TypeError: '<' not supported between instances of 'str' and 'int'

Note: In python-2 list contains both numbers and strings then sort() function first sort numbers followed by strings

Ex:

```
1)      n=[20,10,"B","A"]
2)      n.sort()
3)      print(n)
```

o/p:[10,20,"A","B"]

-->But in python-3 it is invalid.

To sort in reverse of default natural sorting order:

-->We can sort according to reverse of default natural sorting order by using reverse=True argument.

Ex:

```
1)      n=[10,30,20,40]
2)      n.sort()
3)      print(n)==>[10,20,30,40]
4)      n.sort(reverse=True)
5)      print(n)==>[40,30,2010]
6)      n.sort(reverse=False)
7)      print(n)==>[10,20,30,40]
```


NARESH IT, 2nd Floor, Durga Bhavani Complex, Satyam Threater Rd, Ameerpet, HYD
Telangana-500016 | Tel:+91 40 2374 6666 | Ph:+91 90009 94007 | www.nareshit.in

Aliasing and cloning of List objects:

==========

The process of giving another reference variable to the existing list is called as aliasing.

Ex:

```
1)      x=[10,20,30,40]
2)      y=x
3)      y[1]=333
4)      print(x)
5)      print(y)
```

```
o/p:D:\pythonclasses>py test.py
[10, 333, 30, 40]
[10, 333, 30, 40]
```

-->The problem in this approach is by using one reference variable if we are changing content, then those changes will be reflected to the other reference variable.

-->To overcome this problem we should go for cloning.

-->The process of creating exactly duplicate independent object is called as cloning.

-->We can implement cloning by using slice operator or by using copy() function.

1).By using slice operator:

Ex:

```
1)      x=[10,20,30,40]
2)      y=x[:]
3)      y[1]=333
4)      print(x)
5)      print(y)
```

```
o/p:D:\pythonclasses>py test.py
[10, 20, 30, 40]
[10, 333, 30, 40]
```

2).By using copy() function:

Ex:

```
1)      x=[10,20,30,40]
2)      y=x.copy()
3)      y[1]=333
4)      print(x)
```

```
5)      print(y)
```

```
o/p:D:\pythonclasses>py test.py
[10, 20, 30, 40]
[10, 333, 30, 40]
```

Q:Difference between = operator and copy() function?

= operator mean for aliasing.
copy() function menas for cloning.

Using Mathematical operators for list objects:

==========

-->We can use + and * operators for list objects.

1).Contactination operator(+):

We can use + to concatinate 2 lists into a single list.

Ex:

```
1)      a=[10,20,30]
2)      b=[40,50,60]
3)      c=a+b
4)      print("c:",c)
```

```
o/p:D:\pythonclasses>py test.py
c: [10, 20, 30, 40, 50, 60]
```

Ex:

```
1)      a=[10,20,30]
2)      b=[40,50,60]
3)      c=a.extend(b)
4)      print("a:",a)
5)      print("b:",b)
6)      print("c:",c)
```

```
o/p:D:\pythonclasses>py test.py
a: [10, 20, 30, 40, 50, 60]
b: [40, 50, 60]
c: None
```

Note:

To use + operator compulsory both arguments should be list objects, otherwise we will get an error.

Ex:

c=a+40==>TypeError: can only concatenate list (not "int") to list
c=a+[40]==>valid

2).Repetition operator(*):

We can use repetition operator * to repeat the elements of list specified number of times.

Ex:

```
1)  x=[10,20,30]
2)  y=x*3
3)  print(y)==>[10,20,30,10,20,30]
```

Comparing List objects:

==========

We can use comparison operators for list objects.

Ex:

```
1)  x=["Dog","Cat","Rat"]
2)  y=["Dog","Cat","Rat"]
3)  z=["DOG","CAT","RAT"]
4)  print(x==y)==>True
5)  print(x==z)==>False
6)  print(x!=z)==>True
```

Note:

Whenever we are using relational operator(<,<=,>,>=) between list objects, only first element comparision will be performed.

Ex:

```
1)  x=[50,40,20]
2)  y=[10,20,30,40,50,60]
3)  print(x>y)==>True
4)  print(x>=y)==>True
5)  print(x<y)==>False
6)  print(x<=y)==>False
```

Ex:

```
1)  x=["Dog","Cat","Rat"]
2)  y=["Rat","Cat","Dog"]
3)  print(x>y)==>False
4)  print(x>=y)==>False
5)  print(x<y)==>True
```


NARESH IT, 2nd Floor, Durga Bhavani Complex, Satyam Threater Rd, Ameerpet, HYD
Telangana-500016 | Tel:+91 40 2374 6666 | Ph:+91 90009 94007 | www.nareshit.in

```
6)      print(x<=y)==>True
```

Membership operators:

==========

We can check whether element is a memeber of the list or not by using memebership operators.

-->in operator

-->not in operator

clear() function:

We can use clear() function to remove all elements of list.

Ex:

```
1)      n=[10,20,30,40]
2)      print(n)
3)      n.clear()
4)      print(n)
```

```
o/p:D:\pythonclasses>py test.py
[10, 20, 30, 40]
[ ]
```

Nested Lists:

==========

Sometimes we can take one list inside another list. Such type of lists are called as nested lists.

Ex:

```
1)      n=[10,20,[30,40]]
2)      print(n)
3)      print(n[0])
4)      print(n[1])
5)      print(n[2][0])
6)      print(n[2][0])
```

```
o/p:D:\pythonclasses>py test.py
[10, 20, [30, 40]]
10
20
30
40
```

Note:

We can access nested list elements by using index just like accessing multi dimensional array elements.

Nested list as matrix:

==========

In python we can represent matrix by using nested lists.

Ex:

```
1)  n=[[10,20,30],[40,50,60],[70,80,90]]
2)  print(n)
3)  print("Elements by row wise:")
4)  for r in n:
5)          print(r)
6)  print("Elements by Matrix style:")
7)  for i in range(len(n)):
8)          for j in range(len(n[i])):
9)                  print(n[i][j],end=' ')
10)         print()
```

```
o/p:D:\pythonclasses>py test.py
[[10, 20, 30], [40, 50, 60], [70, 80, 90]]
Elements by row wise:
[10, 20, 30]
[40, 50, 60]
[70, 80, 90]
Elements by Matrix style:
10 20 30
40 50 60
70 80 90
```

List Comprehensions

It is very easy and compact way of creating list objects from any iterable objects (like list,tuple,dict,range etc...) based on some condition.

Syn:

```
list=[expression for items in list if condition]
```

Ex: Print first 10-numbers squares in list format.

o/p:[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]

Normal program:

```
1) l=[ ]
2) for x in range(11):
3)         l.append(x*x)
4) print(l)
```

By using comprehensions:

```
1) s=[x*x for x in range(1,11)]
2) print(s)
```

print [2 4 8 16 32]

```
1) s=[2**x for x in range(1,6)]
2) print(s)
```

Print only even numbers from list:

```
1) s=[x*x for x in range(1,11)]
2) m=[v for v in s if v%2==0]
3) print(m)
```

Ex:Print starting letter of each word.

```
1) words=["Chiranjeevi","Nag","Venkatesh","Balaiah"]
2) l=[x[0] for x in words]
3) print(l)
```


NARESH IT, 2nd Floor, Durga Bhavani Complex, Satyam Threater Rd, Ameerpet, HYD
Telangana-500016 | Tel:+91 40 2374 6666 | Ph:+91 90009 94007 | www.nareshit.in

Ex:

```
1) num1=[10,20,30,40]
2) num2=[30,40,50,60]
3) num3=[x for x in num1 if x not in num2]
4) print(num3)
5) num4=[x for x in num1 if x in num2]
6) print(num4)
```

Ex:

```
1) word="the quick brown fox jumps over the lazy dog"
2) words=word.split()
3) print(words)
4) l=[[w.upper(),len(w)] for w in words]
5) print(l)
```

Ex: w.a.p to display unique vowels present in the given string.

```
1) i/p: durgasoftwaresolutions
2) ['u','a','o','e','i']
3) vowels=['a','e','i','o','u']
4) word=input("Enter the word to search for vosels:")
5) found=[]
6) for letter in word:
7)         if letter in vowels:
8)                 if letter not in found:
9)                         found.append(letter)
10) print(found)
```

Tuple Data Structure

-->Tuple is exactly same as List except that it is immutable. i.e once we create tuple object, we can't perform any changes in that object.
-->Hence tuple is read only version of list.
-->If our data is fixed and never changes then we should go for tuple.
-->Insertion order is preserved.
-->Duplicates are allowed.
-->Heterogeneous objects are allowed.
-->We can preserve insertion order and we can differentiate duplicate objects by using index. Hence index will play very important role in Tuple also.
-->We can represent tuple elements within parenthesis and with comma separator. Parenthesis are optional but recommended to use.

Ex:

```
1. t=10,20,30,40
2. print(t)
3. print(type(t))
```

```
o/p:D:\pythonclasses>py test.py
(10, 20, 30, 40)
<class 'tuple'>
```

```
t=()
print(type(t))==>tuple
```

Note: We have to take special care about single valued tuple. COmpulsory the value should ends with comma, otherwise it is not treated as tuple.
Ex:

```
1. t=10
2. print(t)
3. print (type(t))
```

```
o/p:D:\pythonclasses>py test.py
10
<class 'int'>
```

Ex:

```
1. t=10,
2. print(t)
3. print (type(t))
```

```
o/p:D:\pythonclasses>py test.py
(10,)
<class 'tuple'>
```

Q:Which of the following are valid:

1.t=()==>V
2.t=10,20,30,40==>V
3.t=10==>Invalid
4.t=10,==>V
5.t=(10)==>Invalid
6.t=(10,)==>V
7.=t=(10,20,30)==>V

Tuple Creation:

1.t=():

Creation of empty tuple

2.t=(10,) or t=10,

Creation of single valued tuple, paranthesis are optional, should ends with comma.

3.t=10,20,30 or t=(10,20,30)

Creatin of multiple values tupl & parenthesis are optional.

4.By using tuple() function:

```
1.      l=[10,20,30,40]
2.      t=tuple(l)
3.      print(t)
4.      print(type(t))
```

```
o/p:D:\pythonclasses>py test.py
(10, 20, 30, 40)
<class 'tuple'>
```

Ex:

```
1.      t=tuple(range(10,20,2))
2.      print(t)
```

o/p:D:\pythonclasses>py test.py
(10, 12, 14, 16, 18)

Accessing elements of tuple:

==========

We can access either by index or by using slice operator.

1.By using index:

```
1.  t=(10,20,30,40)
2.  print(t[0])==>10
3.  print(t[-1])==>40
4.  print(t[100])==>IndexError
```

2.By suing slice operator:

```
1.  t=(10,20,30,40,50,60)
2.  print(t[2:5])
3.  print(t[2:100])
4.  print(t[::2])
```

o/p:
(30, 40, 50)
(30, 40, 50, 60)
(10, 30, 50)

Tuple Vs immutability:

==========

-->Once we create tuple, we can't change its content.
-->Hence tuple objects are immutable.
Ex:

```
1.  t=(10,20,30,40,50)
2.  t[1]=60==>TypeError: 'tuple' objects does not support item assignment.
```

Mathematical operators for tuple:

==========

-->We can apply + and * operators for tuple.

1.concatination operator(+):

```
1.  t1=(10,20,30,40)
2.  t2=(10,20,50,60)
3.  t3=t1+t2
```

```
4.      print(t3)
```

o/p:(10, 20, 30, 40, 10, 20, 50, 60)

2.Multiplication operator or repetition operator(*):

```
1.      t1=(10,20,30,40)
2.      t2=t1*2
3.      print(t2)
```

o/p:(10, 20, 30, 40, 10, 20, 30, 40)

Important function in tuple:

==========

1.len():

To return number of elements present in tuple

Ex:

```
1.      t=(10,20,30,40)
2.      len(t)==>4
```

2.count():

To return number of occurences of given element in the tuple.

Ex:

```
1.      t=(10,20,30,10,10)
2.      t.count(10)==>3
```

3).index():

-->Returns index of first occurence of the given element.

-->If the specified element is not available then we will get ValueError.

Ex:

```
1.      t=(10,20,30,30,10,40)
2.      t.index(30)==>2
3.      t.index(50)==>ValueError
```

4.sorted():

To sort elements based on default natural sorting order.

Ex:

```
1.      t=(40,20,50,10,30)
2.      t1=sorted(t)
3.      print(t)
```

```
4.      print(t1)
```

o/p:
(40, 20, 50, 10, 30)
[10, 20, 30, 40, 50]==>After sorting it will return as list

Ex:

```
1.      t=(40,20,50,10,30)
2.      t1=tuple(sorted(t))
3.      print(t)
4.      print(t1)
```

o/p:
(40, 20, 50, 10, 30)
(10, 20, 30, 40, 50)

Ex:

```
1.      t1=sorted(t,reverse=True)
2.      print(t1)==>[50,40,30,20,10]
```

5.min() and max() functions:

These functions returns min and max values according to default natural sorting order.

Ex:

```
1.      t=(40,20,50,10,30)
2.      print(min(t))==>10
3.      print(max(t))==>50
```

Ex:

```
1.      t="Mahesh"
2.      print(min(t))==>M
3.      print(max(t))==>s
```

6.cmp():
-->It compares the elements of both tuples.
-->If both tuples are equal then returns 0.
-->If the first tuple is less than second tuple then it returns -1
-->If the first tuple is greater than second tuple then it returns +1

Ex:

```
1.  t1=(10,20,30)
2.  t2=(40,50,50)
3.  t3=(10,20,30)
4.  print(cmp(t1,t2))#-1
5.  print(cmp(t1,t3))#0
6.  print(cmp(t2,t3))#+1
```

Note: cmp() function is available only in python-2 but not in python-3

Tuple Packing and Unpacking:

-->We can create a tuple by using packing a group of variables.

Ex:

```
1.  a=10
2.  b=20
3.  c=30
4.  d=40
5.  t=a,b,c,d
6.  print(t)
```

o/p:(10, 20, 30, 40)

-->Here a,b,c,d are packed into a tuple t. This is nothing but tuple packing.

-->Tuple unpacking is the reverse process of tuple packing.

-->We can unpack a tuple and assign its values to different variables.

Ex:

```
1.  t=(10,20,30,40)
2.  a,b,c,d=t
3.  print("a=",a,"b=",b,"c=",c,"d=",d)
```

o/p:

a= 10 b= 20 c= 30 d= 40

Note:

At the time of tuple unpacking the number if variables and number of values should be same. otherwise we will get an error.

Ex:

```
1.  t=(10,20,30,40)
2.  a,b,c=t
3.  print("a=",a,"b=",b,"c=",c)
```


NARESH IT, 2nd Floor, Durga Bhavani Complex, Satyam Threater Rd, Ameerpet, HYD
Telangana-500016 | Tel:+91 40 2374 6666 | Ph:+91 90009 94007 | www.nareshit.in

o/p:ValueError: too many values to unpack (expected 3)

Tuple Comprehension:

Tuple comprehension is not supported by python.

Ex:

```
t=(x**2 for x in range(1,6))
```

-->Here we are not getting tuple object and we are getting generator object

Ex:

```
1.  t=(x**2 for x in range(1,6))
2.  print(type(t))
3.  for i in t:
4.      print(i)
```

o/p:

```
<class 'generator'>
1
4
9
16
25
```

Q: w.a.p to take tuple of numbers from the keyboard and print sum and average?

```
1. t=eval(input("Enter Tuple Of Numbers:"))
2. l=len(t)
3. sum=0
4. for x in t:
5.     sum=sum+x
6. print("The Sum is:",sum)
7. print("The Average :",sum/l)
```

o/p:

```
Enter Tuple Of Numbers:(10,20,30)
The Sum is: 60
The Average : 20.0
```

Difference between List and Tuple

List	Tuple
1.List a group of comma separated values within square brackets and square brackets are mandatory. Ex: l=[10,20,30]	1.Tuple is a group of comma separated values within parenthesis parenthesis are optional. Ex: t=(10,20,30) or t=10,20,30
2.List objects are mutable i.e once we creates list object we can perform any changes in that object. ex:l[1]=50	2.Tuple objects are immutable i.e once we creates tuple objects we can't change it content. Ex:t[1]=50-->ValueError
3.If the content is not fixed and keep on changing then we should go for list.	3.If the content is fixed and never changes then we should go for tuple.
4.Comprehensions are available.	4.There is no comprehensions.

Set Data Structure

-->If we want to represent a group of unique values as a single entity then we should go for set.
-->Duplicates are not allowed.
-->Insertion order is not preserved. But we can sort the elements.
-->Indexing and slicing not allowed for the set.
-->Heterogeneous elements are allowed.
-->Set objects are mutable i.e once we create set object we can perform any changes in that object based on our requirement.
-->We can represent set elements with curly braces and with comma separation.
-->We can apply mathematical operations like union, intersection, difference etc on set objects.

Creation of Set Objects:

Ex:

```
1. s={10,20,30,40}
2.    print(s)
3.    print(type(s))
```

o/p:
{40,10,20,30}
<class 'set'>
-->We can create set objects by using set() function.
s=set(any sequence)
Ex:
l=[10,20,30,40,10,20]
s=set(l)
print(s)
Ex:
s=set(range(5))
print(s)
s={ }==>It treated as dictionary but not an empty set.
Ex:
s={ }
print(type(s))
o/p: <class 'dict'>

NARESH IT, 2nd Floor, Durga Bhavani Complex, Satyam Threater Rd, Ameerpet, HYD
Telangana-500016 | Tel:+91 40 2374 6666 | Ph:+91 90009 94007 | www.nareshit.in

Ex:

```
1.          s=set()
2.      print(s)
3.      print(type(s))
```

o/p:

```
set()
<class 'set'>
```

Important functions of set:

1.add(x):

Add item x to the set.

Ex:

```
1.          s={10,20,30}
2.      s.add(40)
3.      print(s)
```

2.update(x,y,z):

-->To add mutiple items to the set.

-->Arguments are not individual elements and these are iterable elements like list, range etc..

-->All elements present in the given iterable objects will be added to the set.

Ex:

```
1.          s={10,20,30}
2.      l=[40,50,60]
3.      s.update(l,range(5))
4.      print(s)
```

Q: What is difference between add() and update() functions in set?

-->We can use add() to add individual items to the set, where as we can use update() function to add multiple items to the set.

-->add() function can take only one argument where as update function can take any number of arguments but all arguments should be iterable objects.

Q: Which are valid?

s.add(10) Valid

s.add(10,20,30) Invalid

s.update(30) Invalid

s.update(range(1,10,2),range(0,10,2)) Valid

3.copy():

-->Returns copy of the set

-->It is cloned object.

Ex:

```
1.          s={10,20,30}
2.    s1=s.copy()
3.    print(s1)
```

4.pop():

It removes and returns some random element from the set.

Ex:

```
1.          s={10,20,30,40}
2.    print(s)
3.    print(s.pop())
4.    print(s)
```

o/p:

{40, 10, 20, 30}

40

{10, 20, 30}

5.remove(x):

-->It removes specified element from the set.

-->If the specified element not present in the set then we will get KeyError.

Ex:

```
1.          s={40,20,30,10}
2.    s.remove(30)
3.    print(s)==>{40,20,10}
4.    s.remove(50)==>KeyError:50
```

6.discard(x):

-->It removes specified element from the set.

-->If the specified element not present in the set then we won't get any error.

Ex:

```
1.          s={40,20,30,10}
2.    s.discard(30)
3.    print(s)==>{40,20,10}
4.    s.discard(50)
5.    print(s)==>{40,20,30,10}
```

7.clear():

-->To remove all elements from the set.

Ex:

```
1.          s={10,20,30}
2.      print(s)
3.      s.clear()
4.      print(s)
```

o/p:

{10,20,30}

set()

Mathematical operations on the set:

==========

1).union():

x.union(y)==>We can use this function to return all elements present in both sets.

Syn: x.union(y) or x|y

Ex:

```
1.          x={10,20,30,40}
2.      y={30,40,50,60}
3.      print(x.union(y))#{40, 10, 50, 20, 60, 30}
4.      print(x|y)  #{40, 10, 50, 20, 60, 30}
```

2).intersection():

x.intersection(y) or x & y

-->Returns common elements present in both x and y

Ex:

```
1.          x={10,20,30,40}
2.      y={30,40,50,60}
3.      print(x.intersection(y))#{40, 30}
4.      print(x&y)#{40, 30}
```

3).difference():

x.difference(y) or x-y

-->Returns the elements present in x but not in y.

Ex:

```
1.          x={10,20,30,40}
2.      y={30,40,50,60}
3.      print(x.difference(y))#{10, 20}
4.      print(x-y)#{10, 20}
```

```
5.      print(y-x)#{50, 60}
```

4).symmetric_difference():

x.symmetric_difference(y) or x^y

-->Returns element present in either x or y but not in both.

Ex:

```
1.          x={10,20,30,40}
2.      y={30,40,50,60}
3.      print(x.symmetric_difference(y))#{10, 50, 20, 60}
4.      print(x^y)#{10, 50, 20, 60}
```

Membership operators:(in, not in)

Ex:

```
1.          s=set("mahesh")
2.      print(s)
3.      print('m' in s)
4.      print('z' in s)
```

o/p:{'s', 'a', 'm', 'e', 'h'}
True
False

Set comprehension:

-->set comprehension is possible.

```
s={i *i for i in range(5)}
print(s)#{0, 1, 4, 9, 16}

s={2 **i for i in range(2,10,2)}
print(s)#{16, 256, 64, 4}
```

set objects won't support indexing and slicing:

Ex:

```
1.  s={10,20,30,40}
2.  print(s[0])    TypeError: 'set' object does not support indexing
3.  print(s[1:3])  TypeError: 'set' object does not support indexing
```

Q: w.a.p to eliminate duplicates present in the list.
Approach-1:

```
1. l=eval(input("Enter List Of Values:"))
2. s=set(l)
3. print(s)
```

o/p:Enter List Of Values:[10,20,10,20,30,40,30]
{40, 10, 20, 30}

Approach-2:

```
1. l=eval(input("Enter List Of Values:"))
2. l1=[ ]
3. for x in l:
4.     if x not in l1:
5.         l1.append(x)
6. print(l1)
```

o/p:
D:\pythonclasses>py test.py
Enter List Of Values:[10,20,20,10,30,40]
[10, 20, 30, 40]

Q:w.a.p to print different vowels in the given word?

```
1. w=input("Enter word to serach for vowels:")
2. s=set(w)
3. v={'a','e','i','o','u'}
4. d=s.intersection(v)
5. print("The different vowels present in",w,"are:",d)
```

o/p:D:\pythonclasses>py test.py
Enter word to serach for vowels : mahesh
The different vowels present in mahesh are: {'e', 'a'}

Dictionary Data Structure

-->We can use list, tuple and set to represent a group of individual objects as a single entity.
-->If we want to represent a group of objects key-value pairs then we should go for dictionary.

Ex:

rollno-----name
phone number---address
ip address----domain name

-->Duplicate keys are not allowed but values can be duplicated.
-->Heterogeneous objects are allowed for both key and values.
-->Insertion order is not preserved.
-->Dictionaries are mutable.
-->Dictionaries are dynamic.
-->Indexing and slicing concepts are not applicable.

Note:

In C++ and java Dictionaries are known as "Map" where as in perl and Ruby it is knonw as "Hash"

How to create dictionary?

==========

d={ } or d=dict()

-->We are creating empty dictionary. We can add entries as follows.

```
1. d={}
2. d1=dict()
3. print(type(d))
4. d[100]="Mahesh"
5. d[200]="Durga"
6. d[300]="Sunny"
7. print(d)#{100: 'Mahesh', 200: 'Durga', 300: 'Sunny'}
```

-->If we know the data in advance then we can create dictionary as follows
d={100:'mahesh',200:'durga',300:'sunny'}
d={key:value,key:value}

NARESH IT, 2nd Floor, Durga Bhavani Complex, Satyam Threater Rd, Ameerpet, HYD
Telangana-500016 | Tel:+91 40 2374 6666 | Ph:+91 90009 94007 | www.nareshit.in

How to access data from the dictionary?

-->We can access data by using index.

```
d={100:"mahesh",200:"durga",300:"sunny"}
print(d[100])==>mahesh
print(d[300])==>Sunny
```

-->If the specified is not available then we will get a KeyError

```
print(d[400])==>KeyError:400
```

-->We can prevent this by checking whether key is already available or not by using has_key() function or by using in operator.

Ex:d.has_key(400)==>Returns 1 if key is available otherwise returns 0.

-->But has_key() function is available in python-2 but not in python-3. Hence compulsory we have to use in operator.

Ex:

```
if 400 in d:
        print(d[400])
```

Q: w.a.p to enter name and percentage marks in a dictionary and display information on the screen.

```
1.  rec={ }
2.  n=int(input("Enter Number Of Students: "))#3
3.  i=1
4.  while i<=n:
5.      name=input("Enter Student Name:")
6.      marks=int(input("Enter % of Marks:"))
7.      rec[name]=marks
8.      i=i+1
9.  print("Name of the student","\t","% of marks")
10. for x in rec:
11.     print("\t",x,"\t\t",rec[x])
```

```
o/p:D:\pythonclasses>py test.py
Enter Number Of Students: 3
Enter Student Name:Mahesh
Enter % of Marks:90
Enter Student Name:Durga
Enter % of Marks:98
```

```
Enter Student Name:Sunny
Enter % of Marks:100
Name of the student        % of marks
        Mahesh                  90
        Durga                   98
        Sunny                   100
```

How to update dictionaries?

Ex:

```
1.  d={100:"mahesh",200:"durga",300:"sunny"}
2.  print(d)
3.  d[400]="bunny"
4.  print(d)
5.  d[100]="chinny"
6.  print(d)
```

```
o/p:D:\pythonclasses>py test.py
{100: 'mahesh', 200: 'durga', 300: 'sunny'}
{100: 'mahesh', 200: 'durga', 300: 'sunny', 400: 'bunny'}
{100: 'chinny', 200: 'durga', 300: 'sunny', 400: 'bunny'}
```

How to delete elements from dictionary:

---

del d[key]:

It deletes entry associated with the specified key.
If the key is not available then we will get KeyError.

Ex:

```
1.  d={100:"mahesh",200:"durga",300:"sunny"}
2.  print(d)
3.  del d[300]
4.  print(d)
5.  del d[400]
6.  print(d)
```

```
o/p:D:\pythonclasses>py test.py
{100: 'mahesh', 200: 'durga', 300: 'sunny'}
{100: 'mahesh', 200: 'durga'}
KeyError: 400
```

d.clear():

To remove all entries from the dictionary.

Ex:

```
1.          d={100:"mahesh",200:"durga",300:"sunny"}
2.      print(d)
3.      d.clear()
4.      print(d)
```

```
o/p:D:\pythonclasses>py test.py
{100: 'mahesh', 200: 'durga', 300: 'sunny'}
{}
```

del d:

To delete total dictionary. Now we can't access d.

Ex:

```
1.          d={100:"mahesh",200:"durga",300:"sunny"}
2.      print(d)
3.      del d
4.      print(d)
```

```
o/p:D:\pythonclasses>py test.py
{100: 'mahesh', 200: 'durga', 300: 'sunny'}
NameError: name 'd' is not defined
```

Important functions of dictionary:

---

1).dict():

To create a dictionary.

d=dict()==>It creates empty dictionary

```
1. d=dict({100:"mahesh",200:"durga"})==>It creates dictionary with specified elemen
   ts
2. print(d)
3. d=dict([(100,"Mahesh"),(200,"Durga")])==>It creates a dictionary with the given lis
   t of tuple lements.
4. print(d)
5. d=dict(((100,"MAHESH"),(200,"DURGA")))==>It creates a dictionary with the given
   tuple of tuple lements.
6. print(d)
```

o/p:D:\pythonclasses>py test.py
{100: 'mahesh', 200: 'durga'}
{100: 'Mahesh', 200: 'Durga'}
{100: 'MAHESH', 200: 'DURGA'}

2).len():

Returns the number of items in the dictionary.

3).clear():

To remove all the elements from the dictionary.

4.get():

To get the value associated with the key.

d.get(key):

If the key is available then returns the corresponding value otherwise returns None. It wont raise any error.

d.get(key,defaultvalue):

If the key is available then returns the corresponding value otherwise returns default value.

Ex:

```
1.  d={100:"mahesh",200:"durga",300:"sunny"}
2.  print(d[100])#mahesh
3.  print(d[400])#KeyError:400
4.  print(d.get(300))#sunny
5.  print(d.get(400))#None
6.  print(d.get(100,"Guest"))#mahesh
7.  print(d.get(400,"Guest"))#Guest
```

5).pop():

d.pop(key):

It removes the entry associated with the specified key and returns the corresponding value.

If the specified key is not available then we will get KeyError.

Ex:

```
1.      d={100:"mahesh",200:"durga",300:"sunny"}
2.  print(d)
3.  print(d.pop(300))
4.  print(d)
5.  d.pop(400)
```

```
o/p:D:\pythonclasses>py test.py
{100: 'mahesh', 200: 'durga', 300: 'sunny'}
sunny
{100: 'mahesh', 200: 'durga'}
KeyError: 400
```

6).popitem():

It removes an arbitrary item(key-value) from the dictionary and return it.

Ex:

```
1.          d={100:"mahesh",200:"durga",300:"sunny"}
2.      print(d)
3.      print(d.popitem())
4.      print(d)
```

```
o/p:D:\pythonclasses>py test.py
{100: 'mahesh', 200: 'durga', 300: 'sunny'}
{100: 'mahesh', 200: 'durga'}
```

-->If the dictionary is empty then we will get KeyError.

```
d={ }
print(d.popitem())==>KeyError:'popitem():dictionary is empty'.
```

7).keys():

It returns all keys associated with dictionary.

Ex:

```
1.          d={100:"mahesh",200:"durga",300:"sunny"}
2.      print(d.keys())
3.      for k in d.keys():
4.          print(k)
```

```
o/p:D:\pythonclasses>py test.py
dict_keys([100, 200, 300])
100
200
300
```

8).values():

It returns all values associated with dictionary.

Ex:

```
1.          d={100:"mahesh",200:"durga",300:"sunny"}
2.      print(d.values())
```

```
for v in d.values():
    print(v)
```

o/p:D:\pythonclasses>py test.py
dict_values(['mahesh', 'durga', 'sunny'])
mahesh
durga
sunny

9).items():

It returns list of tupes represented key-value pairs.

[(k,v),(k,v),(k,v)]

Ex:

```
d={100:"mahesh",200:"durga",300:"sunny"}
for k,v in d.items():
    print(k,"--",v)
```

o/p:D:\pythonclasses>py test.py
100 -- mahesh
200 -- durga
300 -- sunny

10).copy():

To create exactly duplicate dictionary (cloned copy)

Ex:

```
d={100:"mahesh",200:"durga",300:"sunny"}
d1=d.copy()
print(id(d))
print(id(d1))
```

o/p:D:\pythonclasses>py test.py
2791188556752
2791188556824

11).setdefault(): d.setdefault(k,v)

-->If the key is already available then this function returns the corresponding value.

-->If the key is not available then the specified key-value will be added as new item to the dictionary.

Ex:

```
d={100:"mahesh",200:"durga",300:"sunny"}
print(d.setdefault(400,"bunny"))
print(d)
```

```
print(d.setdefault(100,"bunny"))
print(d)
```

```
o/p:D:\pythonclasses>py test.py
bunny
{100: 'mahesh', 200: 'durga', 300: 'sunny', 400: 'bunny'}
mahesh
{100: 'mahesh', 200: 'durga', 300: 'sunny', 400: 'bunny'}
```

12).update(): d.update(x)

All items in the dictionary x will added to dictionary d.

Ex:

```
d={100:"mahesh",200:"durga",300:"sunny"}
d1={'a':"apple",'b':"banana",'c':"cat"}
d.update((d1))
print(d)
d.update([(333,"A"),(999,"B")])
print(d)
```

```
o/p:D:\pythonclasses>py test.py
{100: 'mahesh', 200: 'durga', 300: 'sunny', 'a': 'apple', 'b': 'banana', 'c': 'cat'}
{100: 'mahesh', 200: 'durga', 300: 'sunny', 'a': 'apple', 'b': 'banana', 'c': 'cat', 333: 'A', 999:
'B'}
```

Q: w.a.p to take dictionary from the keyboard and print sum of values.

sol:

```
d=eval(input("Enter Dictionary:"))
s=sum(d.values())
print("Sum:",s)
```

```
o/p:D:\pythonclasses>py test.py
Enter Dictionary:{'A':100,'B':200,'C':300}
Sum: 600
```

Q:w.a.p to find number of occurances of each letter present in the given string?

sol:

```
word=input("Enter any word:")
d={}
for x in word:
    d[x]=d.get(x,0)+1
print(d)
print(sorted(d))
```

```
    for k,v in d.items():
        #print(k,"occurs",v,"times")
        print("{} occured {}".format(k,v))
```

```
o/p:D:\pythonclasses>py test.py
Enter any word:mississippi
{'m': 1, 'i': 4, 's': 4, 'p': 2}
['i', 'm', 'p', 's']
m occured 1
i occured 4
s occured 4
p occured 2
```

Q: w.a.p to find number of occurances of each vowel present in the given string
sol:

```
word=input("Enter some string:")
vowels={'a','e','i','o','u'}
d={}
for x in word:
    if x in vowels:
        d[x]=d.get(x,0)+1
for k,v in sorted(d.items()):
    print(k,"occured",v,"times")
```

```
o/p:D:\pythonclasses>py test.py
Enter some string:abaiobeioeiaieus
a occured 3 times
e occured 3 times
i occured 4 times
o occured 2 times
u occured 1 times
```

Q:w.a.p to accept student name and marks from the keyboard and creates a dictionary, also display student marks by taking student name as input?

```
n=int(input("Enter the number of students:"))
d={}
for i in range(n):
    name=input("Enter student name:")
    marks=input("Enter marks:")
    d[name]=marks
print(d)
while True:
```

```
9)    name=input("Enter student name to get marks:")
10)   marks=d.get(name,-1)
11)   if marks==-1:
12)      print("Student not found")
13)   else:
14)      print("The marks of",name,"are",marks)
15)   option=input("Do you want to find another student marks[Yes|No]")
16)   if option=="No":
17)      break
18) print("Thanks for using our application")
```

```
o/p:D:\pythonclasses>py test.py
Enter the number of students:3
Enter student name:durga
Enter marks:90
Enter student name:mahesh
Enter marks:98
Enter student name:sunny
Enter marks:100
{'durga': '90', 'mahesh': '98', 'sunny': '100'}
Enter student name to get marks:sunny
The marks of sunny are 100
Do you want to find another student marks[Yes|No]Yes
Enter student name to get marks:bunny
Student not found
Do you want to find another student marks[Yes|No]Yes
Enter student name to get marks:mahesh
The marks of mahesh are 98
Do you want to find another student marks[Yes|No]No
Thanks for using our application
```

Dictionary comprehension:

-->Comprehension concept applicable for dictionary also.

Ex:

```
1. squares={x:x*x for x in range(1,6)}
2. print(squares)
3. doubles={x:2*x for x in range(1,6)}
4. print(doubles)
```

```
o/p:
{1: 1, 2: 4, 3: 9, 4: 16, 5: 25}
{1: 2, 2: 4, 3: 6, 4: 8, 5: 10}
```


NARESH IT, 2nd Floor, Durga Bhavani Complex, Satyam Threater Rd, Ameerpet, HYD
Telangana-500016 | Tel:+91 40 2374 6666 | Ph:+91 90009 94007 | www.nareshit.in

FUNCTIONS

If a group of statements is repeatedly required then it is not recommended to write these statements every time separately. We have to define these statements as a single unit and we can call that unit any number of times based on our requirement without re-writting. THis unit is nothing but a function.

-->The main advantage of functions is code -Re-usability.

Note:In other languages functions are known as methods, procedures,subroutines etc...

-->Python supports 2-types of functions

1).Built in functions

2).User defined functions

1).Built in Functions:

The functions which are coming along with python s/w automatically, are called as built in functions or pre-defined functions.

Ex:

id(), type(), input(), eval().....

2).User Defined Functions:

The functions which are developed by programmers explicitly according to business requirements, are called as user defined functions.

Syn:

```
def function_name(parameter):
        body
        ------
        -----
        return value
```

Note: while creating functions we can use 2 keywords

1.def(mandatory)

2.return(optional)

Ex: write a function to print Hello gud mng.....

```
1. def wish():
2.     print("Hello gud mng.....")
3. wish()
```


NARESH IT, 2nd Floor, Durga Bhavani Complex, Satyam Threater Rd, Ameerpet, HYD
Telangana-500016 | Tel:+91 40 2374 6666 | Ph:+91 90009 94007 | www.nareshit.in

```
4. wish()
5. wish()
```

o/p:D:\pythonclasses>py test.py
Hello gud mng.....
Hello gud mng.....
Hello gud mng.....

Parameters:

Parameters are inputs to the function. If a function contains parameters, then at the time of calling, compulsory we should provide values, otherwise we will get error.

Ex: write a function to take name of the student as input and print wish message by name

```
1. def wish(name):
2.     print("Hello", name," gud mng.....")
3. wish("Durga")
4. wish("Guest")
```

o/p:D:\pythonclasses>py test.py
Hello Durga gud mng.....
Hello Guest gud mng.....

Ex: write a function to take number as input and print its square value.

```
1. def squareit(number):
2.     print("the square of", number," is:",number**2)
3. squareit(4)
4. squareit(5)
```

o/p:D:\pythonclasses>py test.py
the square of 4 is: 16
the square of 5 is: 25

Return Statement:

Function can take input values as parameters and executes business logic, and returns output to the caller with the return statement.

Q:write a function to accept 2-numbers as input and return sum

```
1. def add(x,y):
2.     return x+y
3. result=add(10,20)
```

```
4. print("The sum is:",result)
5. print("The sum is:",add(10,20))
```

o/p:D:\pythonclasses>py test.py
The sum is: 30
The sum is: 30

-->If we are not writing return statement then default return value is None.
Ex:

```
1. def f1():
2.     print("hello")
3. f1()
4. print(f1())
```

o/p:D:\pythonclasses>py test.py
hello
hello
None

Q:write a function to check whether the given number is even or odd?

```
1. def even_odd(num):
2.     if num%2==0:
3.         print(num,"is even number")
4.     else:
5.         print(num,"is odd number")
6. even_odd(10)
7. even_odd(15)
```

o/p:D:\pythonclasses>py test.py
10 is even number
15 is odd number

Q:write a function to find factorial of given number.

```
1. def fact(num):
2.     result=1
3.     while num>=1:
4.         result=result*num
5.         num=num-1
6.     return result
7. print(fact(5))
8. for i in range(1,5):
9.     print("The factorial ",i,"is",fact(i))
```

```
o/p:D:\pythonclasses>py test.py
120
The factorial  1 is 1
The factorial  2 is 2
The factorial  3 is 6
The factorial  4 is 24
```

Returning multiple values from a function:

-->In other languages like C, C++ and java, function can return atmost one value. BUt in python, a function can return any number of values.

Ex:

```
1. def sum_sub(a,b):
2.     sum=a+b
3.     sub=a-b
4.     return sum,sub
5. x,y=sum_sub(100,50)
6. print("sum is: ",x)
7. print("sub is: ",y)
```

```
o/p:D:\pythonclasses>py test.py
sum is:  150
sub is:  50
```

Ex:

```
1. def  calc(a,b):
2.     sum=a+b
3.     sub=a-b
4.     mul=a*b
5.     div=a/b
6.     return sum,sub,mul,div
7. t=calc(100,50)
8. print(t)
9. print(type(t))
10. print("The Results are:")
11. for i in t:
12.     print(i)
```

```
o/p:D:\pythonclasses>py test.py
(150, 50, 5000, 2.0)
<class 'tuple'>
```

The Results are:
150
50
5000
2.0

Types of arguments:
----
def f1(a,b):

----
f1(10,20)

-->a,b are formal arguments where as 10,20 are actual arguments.

-->There are 4-types of actual arguments are allowed in python
1.positional arguments
2.keyword arguments
3.default arguments
4.variable length arguments

1.positional arguments:

These are the arguments passed to function in correct positional order
Ex:

```
1. def sub(a,b):
2.     print(a-b)
3. sub(100,200)
4. sub(200,100)
```

-->The number of arguments and position of arguments must be matched. If we change the order then the result may be changed.
o/p:D:\pythonclasses>py test.py
-100
100

-->If we change the number of arguments then we will get an error.
Ex: sub(200,100,300)
o/p:ypeError: sub() takes 2 positional arguments but 3 were given

2).keyword arguments:

We can pass argument values by keyword i.e by parameter name.

Ex:

```
1. def wish(name,msg):
2.     print("Hello",name,msg)
3. wish(name="Mahesh",msg="gud mng....")
4. wish(msg="gud mng....",name="Mahesh")
```

o/p:D:\pythonclasses>py test.py
Hello Mahesh gud mng....
Hello Mahesh gud mng....

-->Here the order of arguments is not important but number of arguments must be matched.

Note:

We can use both positional and keyword arguments simultaneously. But first we have to take positional arguments and then take keyword arguments, otherwise we will get an error.

Ex:

```
1. def wish(name,msg):
2.     print("Hello",name,msg)
3. wish("Mahesh","Gud Mng.....")==>Valid
4. wish("Mahesh",msg="Gud Mng.....")==>Valid
5. wish(name="Mahesh","Gud Mng.....")==>Invalid
```

o/p:SyntaxError: positional argument follows keyword argument

3.Default argument:

Sometimes we can provide default values for our positional arguments

Ex:

```
1. def wish(name="Mahesh"):
2.     print("Hello",name,"Gud Mng.....")
3. wish()
4. wish("Durga")
```

o/p:D:\pythonclasses>py test.py
Hello Mahesh Gud Mng.....
Hello Durga Gud Mng.....

-->If we are not passing any name then only default value will be considered.

4).Variable length arguments:

-->Sometimes we can variable number of arguments to our function, such type of arguments are called as variable length arguments.

-->We can declare a variable length argument with * symbol as follows.

Syn: def f1(*n):

-->We can call this function by passing any number of arguments including zero number, internally all these values represented in the form of tuple.

Ex:

```
1. def sum(*n):
2.     total=0
3.     for i in n:
4.         total=total+i
5.     print("Sum is:",total)
6. sum()
7. sum(10,20)
8. sum(10,20,30)
9. sum(10,20,30,40)
```

```
o/p:D:\pythonclasses>py test.py
Sum is: 0
Sum is: 30
Sum is: 60
Sum is: 100
```

Ex:

```
1. def f1(n1,*s):
2.     print(n1)
3.     for i in s:
4.         print(i)
5. f1(10)
6. f1(10,20,30,40)
7. f1(10,"A",20,"B")
```

```
o/p:D:\pythonclasses>py test.py
10
10
20
30
40
```

10
A
20
B

Note: After variable length argument, if we are taking any other arguments then we should provide values as keyword arguments.
Ex:

```
1. def f1(*s,n1):
2.     for i in s:
3.         print(i)
4.     print(n1)
5. f1(10,20,"A",n1=30)
```

o/p:D:\pythonclasses>py test.py
10
20
A
30

f1("A","B",10)==>Invalid
TypeError: f1() missing 1 required keyword-only argument: 'n1'

Note: We can declare key word variable length arguments also. For this we have to use **.
Syn: f1(**n)
-->We can call this function by passing any number of keyword arguments. Internally these keyword arguments will be stored inside a dictionary.
Ex:

```
1. def display(**kwargs):
2.     for k,v in kwargs.items():
3.         print(k,"=",v)
4. display(n1=10,n2=20,n3=30)
5. display(rno=100,name="Mahesh",marks=90,subject="selenium")
```

o/p:D:\pythonclasses>py test.py
n1 = 10
n2 = 20
n3 = 30
rno = 100
name = Mahesh
marks = 90
subject = selenium

Case Study:
==========

```
def f(arg1,arg2,arg3=4,arg4=8):
        print(arg1,arg2,arg3,arg4)

1.f(3,2)#3 2 4 8

2.f(10,20,30,40)#10 20 30 40

3.f(25,50,arg4=100)#25 50 4 100

4.f(arg4=2,arg1=3,arg2=4)#3 4 4 2

5.f()#invalid
TypeError: f() missing 2 required positional arguments: 'arg1' and 'arg2'

6.#f(arg3=10,arg4=20,30,40)#invalid
SyntaxError: positional argument follows keyword argument
[After keyword arguments we should not take positional arguments]

7.f(4,5,arg2=6)#invalid
TypeError: f() got multiple values for argument 'arg2'

8.f(4,5,arg3=5,arg5=6)#Invalid
TypeError: f() got an unexpected keyword argument 'arg5'
```

Function vs Module vs Package vs Library:
==========

-->A group of lines with some name is called as a function.
-->A group of functions saved to a file is called as module.
-->A group of modules is nothing but a package.
-->A group of packages is nothing but a library.

Types of Variables:

Python supports two types of variables.

-->Global Variable
-->Local Variable

Global variable:

-->The variables which are declared outside the function are called as global variables.
-->These variables can be accessed in all functions of that module.

Ex:

```
1. a=10#global variable
2. def f1():
3.     print('f1:',a)
4. def f2():
5.     print('f2:',a)
6. f1()
7. f2()
```

o/p:
10
10

Local Variable:

-->The variables which are declared inside a function are called as local variables.
-->Local variables are available only for the function in which we declared it. i.e from outside of function we can't access.
Ex:

```
1. def f1():
2.     a=10
3.     print('f1:',a)
4. def f2():
5.     print('f2:',a)
6. f1()
7. f2()
```

o/p:
f1: 10
NameError: name 'a' is not defined

global keyword:

We can use global keyword for the following 2 purposes
-->To delcare global variable inside a function.
-->To make global variable available to the function so that we can perform required modifications.
Ex:

```
1. a=10
2. def f1():
3.     a=333
```

```
    print('f1:',a)
def f2():
    print('f2:',a)
f1()
f2()
```

o/p:D:\pythonclasses>py test.py
f1: 333
f2: 10

Ex:

```
a=10#global
def f1():
    global a
    a=333#local
    print('f1:',a)
def f2():
    print('f2:',a)
f1()          f2()
f2()          f1()
```

o/p:D:\pythonclasses>py test.py
f1: 333
f2: 333

o/p:D:\pythonclasses>py test.py
f2: 10
f1: 333

Ex:

```
def f1():
    global a
    a=10
    print('f1:',a)
def f2():
    print('f2:',a)
f1()
f2()
```

o/p:D:\pythonclasses>py test.py
f1: 10
f2: 10

Note: If the global variable and local variable having same name then we can access global variable inside a function as follows.

Ex:

```
1. a=10#global variable
2. def f1():
3.     a=333#local variable
4.     print('f1:',a)
5.     print(globals()['a'])
6. f1()
```

o/p:D:\pythonclasses>py test.py
f1: 333
10

Recursive Functions:

A function that calls itself is known as Recursive Function.

Ex:
factorial(3):3*factorial(2)
3*2*factorial(1)
3*2*1*factorial(0)

factorial(n):n*factorial(n-1)

The main advantages of recursive functions are:
-->We can reduce length of the code and improve readability.
-->We can solve complex problems very easily.

write a function to find factorial of given number with recusrion.

```
1. def factorial(n):
2.     if n=0:
3.         result=1
4.     else:
5.         result=n*factorial(n-1)
6.     return result
7. print("Factorial of 4 is: ",factorial(4))
8. print("Factorial of 0 is: ",factorial(0))
```

o/p:D:\pythonclasses>py test.py
Factorial of 4 is: 24
Factorial of 0 is: 1

Anonymous Functions:

==========

-->Sometimes we can declare a function without name, such type of nameless functions are called as anonymous functions or lamda functions.

-->The main advantage of anonymous function is just for instant use(i.e for one time usage)

Normal Function:

-->We can define by using def keyword.

```
def squareit(n):
    return n*n
```

lambda function:

-->We can define by using lambda keyword.

```
lambda n:n*n
```

Syntax of lambda function:

```
lambda argument_list:expression
```

Note:

By using lambda functions we can write concise code so that readability of the program will be improved.

Q: w.a.p to create a lambda function to find square of given number

----

```
1. s=lambda n:n*n
2. print("The square of 3 is:",s(3))
3. print("The square of 5 is:",s(5))
```

```
o/p:D:\pythonclasses>py test.py
The square of 3 is: 9
The square of 5 is: 25
```

Q: Lambda function to find sum of 2-given numbers

```
1. s=lambda a,b:a+b
2. print("The sum of 10 and 20 is:",s(10,20))
3. print("The sum of 100 and 200 is:",s(100,200))
```

```
o/p:D:\pythonclasses>py test.py
```

The sum of 10 and 20 is: 30
The sum of 100 and 200 is: 300

Q:Lambda function to find biggest of given values:

```
1. s=lambda a,b:a if a>b else b
2. print("The biggest of 10 and 20 is:",s(10,20))
3. print("The biggest of 100 and 200 is:",s(100,200))
```

o/p:D:\pythonclasses>py test.py
The biggest of 10 and 20 is: 20
The biggest of 100 and 200 is: 200

Note:

Lambda function internally returns expression value and we are not required to wrtite return statement explicitly.

Some times we can pass function as argument to another function. In such case lambda functions are best choice.

-->We can use lambda functions very commonly with filter(), map() and reduce() functions,bcoz these functions expect function as aegument.

1).filter function:

We can use filter() function to filter values from the given sequence based on some condition.

Syn:

filter(function,sequence)

-->Where function argument is responsible to perform conditional check sequence can be list or tuple or string.

Q: w.a.p to filter even numbers from the list by using filter()
---
without lambda function:

```
1. def isEven(n):
2.     if n%2==0:
3.         return True
4.     else:
5.         return False
6. l=[0,5,10,15,20,25,30]
7. print(type(filter(isEven,l)))
```

```
8. l1=list(filter(isEven,l))
9. print(l1)
```

o/p:D:\pythonclasses>py test.py
<class 'filter'>
[0, 10, 20, 30]

with lambda function:

```
1. l=[0,5,10,15,20,25,30]
2. l1=list(filter(lambda x:x%2==0,l))
3. print(l1)
4. l2=list(filter(lambda x:x%2!=0,l))
5. print(l2)
```

o/p:D:\pythonclasses>py test.py
[0, 10, 20, 30]
[5, 15, 25]

2).map() function:

For every element present in the given sequenece, apply some functionality and generate new element with the required modifications. For this requirement we should go for map() function.

Ex: For every element present in the list perform double and generate new list of doubles

Syn:

map(function,sequenec)

-->The function can be applied on each element of sequence and generates new sequence.

without lambda:

```
1. l=[1,2,3,4,5]
2. def doubleIt(x):
3.     return 2*x
4. l1=list(map(doubleIt,l))
5. print(l1)==> [2, 4, 6, 8, 10]
```

with lambda:

```
1. l=[1,2,3,4,5]
2. l1=list(map(lambda x:2*x,l))
3. print(l1)==> [2, 4, 6, 8, 10]
```

Ex: To find square of given numbers

```
1. l=[1,2,3,4,5]
2. l1=list(map(lambda x:x*x,l))
3. print(l1)==> [1, 4, 9, 16, 25]
```

-->We can apply map() function on multiple lists also, but make sure all list should have same length.

Syn:

map(lambda x,y:x*y,l1,l2)

x is from l1 and y is from l2

Ex:

```
1. l1=[1,2,3,4]
2. l2=[2,3,4,5]
3. l3=list(map(lambda x,y:x*y,l1,l2))
4. print(l3)
```

o/p: [2, 6, 12, 20]

3).reduce() function:

---

reduce() function reduces sequence of elements into a single element by applying the specified function.

Syn:

reduce(function, sequence)

-->reduce() function present in functools module and hence we should write import statement.

Ex:

```
1. from functools import *
2. l=[10,20,30,40,50]
3. result=reduce(lambda x,y:x+y,l)
4. print(result)
```

o/p:150

Ex:

```
1. from functools import *
2. l=[10,20,30,40,50]
3. result=reduce(lambda x,y:x*y,l)
4. print(result)
```

o/p:12000000

Ex:

```
1. from functools import *
2. result=reduce(lambda x,y:x+y,range(1,101))
3. print(result)
```

o/p:5050

Nested functions:

==========

Ex:

```
1. def f1():
2.     def inner(a,b):
3.         print("The sum:",a+b)
4.         print("The average:",(a+b)/2)
5.         print()
6.     inner(10,20)
7.     inner(20,30)
8.     inner(100,200)
9. f1()
```

o/p:D:\pythonclasses>py test.py
The sum: 30
The average: 15.0

The sum: 50
The average: 25.0

The sum: 300
The average: 150.0

Ex:

```
1)  def outer():
2)      print("outer function started")
3)      def inner():
4)          print("ineer function started")
5)      print("outer function returning inner function")
6)      return inner
7)  f1=outer()
```

o/p:D:\pythonclasses>py test.py
outer function started
outer function returning inner function

f1=outer()
f1()
o/p:D:\pythonclasses>py test.py
outer function started
outer function returning inner function
ineer function started

f1=outer() #Function call
f1=outer #Function aliasing
f1()
o/p:D:\pythonclasses>py test.py
outer function started
outer function returning inner function

Function Decorators

Decorator is a function which can take a function as a argument and extend its functionality and returns modified function with extended functionality.

Input function-------->Decorator Function---------->o/p function with extended fun

The main objective decorator function is we can extend the functionality of existing functions without modifies that function.

Decorators help to make our code shorter and more pythonic. This concept is veryhelpful while developing web applications with Django.

Ex:

```
def wish(name):
        print("Hello",name,"Good morning.....")
```

-->This function can always print same output for any name.
Hello Mahesh Good morning.....
Hello Durga Good morning.....
Hello Sunny Good morning.....

-->But we want to modify this function to provide different message if name is "Sunny". We can do this with out touching wish() function by using decorator.

Ex:

```
1)  def decor(func):
2)      def inner(name):
3)          if name=="Sunny":
4)              print("Hello Sunny Bad Morning.....")
5)          else:
6)              func(name)
7)      return inner
8)  @decor
9)  def wish(name):
10)     print("Hello",name,"Good morning.....")
11) wish("Mahesh")
12) wish("Durga")
13)  wish("Sunny")
```


NARESH IT, 2nd Floor, Durga Bhavani Complex, Satyam Threater Rd, Ameerpet, HYD
Telangana-500016 | Tel:+91 40 2374 6666 | Ph:+91 90009 94007 | www.nareshit.in

o/p:D:\pythonclasses>py test.py
Hello Mahesh Good morning.....
Hello Durga Good morning.....
Hello Sunny Bad Morning.....

-->In the above program whenever we call wish() function automatically decor function will be executed.

How to call same function with decorator and with out decorator:

Ex:

```
1)  def decor(func):
2)      def inner(name):
3)          if name=="Sunny":
4)              print("Hello Sunny Bad Morning.....")
5)          else:
6)              func(name)
7)      return inner
8)
9)  def wish(name):
10)     print("Hello",name,"Good morning.....")
11)
12) decorfunction=decor(wish)
13) wish("Mahesh")     #decorator wont be executed
14) wish("Sunny")      #decorator wont be executed
```

o/p:D:\pythonclasses>py test.py
Hello Mahesh Good morning.....
Hello Sunny Good morning.....

decorfunction("Mahesh") #decorator will be executed
decorfunction("Sunny") #decorator will be executed
o/p:D:\pythonclasses>py test.py
Hello Mahesh Good morning.....
Hello Sunny Bad Morning.....

Ex:

```
1)  def smart_division(func):
2)      def inner(a,b):
3)          print("We are deviding",a,"with",b)
```

```
4)         if b==0:
5)             print("OOP's ..can't divide with zero")
6)         else:
7)             return func(a,b)
8)     return inner
9)
10) @smart_division
11) def division(a,b):
12)     return a/b
13) print(division(20,2))
14) print(division(20,0))
```

-->without decorator we will get an error. In this case output is:

```
D:\pythonclasses>py test.py
10.0
Traceback (most recent call last):
  File "test.py", line 13, in <module>
    print(division(20,0))
  File "test.py", line 11, in division
    return a/b
ZeroDivisionError: division by zero
```

-->with decorator we won't get any error. In this casse output is:

```
D:\pythonclasses>py test.py
We are dividing 20 with 2
10.0
We are dividing 20 with 0
OOP's ..can't devide with zero
None
```

Decorator Chaining

We can define multiple decorators for the same function and all these decorators will perform decorator chaining.

Ex:

```
@decor1
@decor
def num():
```

-->For num() function we are applying 2 decorator functions. First inner decorator will work and then outer decorator.

Ex:

```
1.  def decor1(func):
2.      def inner():
3.          x=func()
4.          return x*x
5.      return inner
6.
7.  def decor(func):
8.      def inner():
9.          x=func()
10.         return 2*x
11.     return inner
12.
13. @decor1
14. @decor
15. def num():
16.     return 10
17. print(num())
```

o/p:D:\pythonclasses>py test.py
400

Generators

-->Generator is a function which is responsible to generate a sequence of values.
-->We can write generator function just like ordinary functions, but it uses "yield" keyword to return values.

Yield------>Generator--------->A sequence of values

Ex:

```
1) def mygen():
2)     yield 'A'
3)     yield 'B'
4)     yield 'C'
5) g=mygen()
6) print(type(g))
7) print(next(g))
8) print(next(g))
9) print(next(g))
```

```
o/p:D:\pythonclasses>py test.py
<class 'generator'>
A
B
C
```

Ex-2:

```
1) def countdown(num):
2)         print("start countdown")
3)         while (num>0):
4)                 yield num
5)                 num=num-1
6) values=countdown(5)
7) for i in values:
8)         print(i)
```


NARESH IT, 2nd Floor, Durga Bhavani Complex, Satyam Threater Rd, Ameerpet, HYD
Telangana-500016 | Tel:+91 40 2374 6666 | Ph:+91 90009 94007 | www.nareshit.in

```
o/p:D:\pythonclasses>py test.py
start countdown
5
4
3
2
1
```

Ex-3: To generate first n numbers.

```
1) def firstn(num):
2)         i=1
3)         while i<=num:
4)                 yield i
5)                 i=i+1
6) values=firstn(5)
7) for x in values:
8)         print(x)
9) print(type(values))
```

```
o/p:D:\pythonclasses>py test.py
1
2
3
4
5
```

-->We can convert generator into list as follows:

```
values=first(5)
l=list(values)
print(l)==>[1,2,3,4,5]
<class 'generator'>
```

Ex-4: To generate fibonacci numbers.....

```
1) def fib():
2)         a,b=0,1
3)         while True:
4)                 yield a
5)                 a,b=b,a+b
6) for x in fib():
7)         if x>100:
```

```
8)                  break
9)          print(x)
```

```
o/p:D:\pythonclasses>py test.py
0 1 1 2 3 5 8 13 21 34 55 89
```

Advantages of generator functions:

---

1).When compared with class level iterations, generators are very easy to use.
2).Improves memory utilization and performance.
3).Generators are best suitable for reading data from large number of large files.
4).Generators work great for web scraping and crawling.
5).Memory utilization, bcoz we are not required to store all values at a time.

MODULES

-->A group of functions, variables and classes saved into a file, which is nothing but module.
-->Every python file(.py) acts as module.
-->Advantages of modules are: Reusability, Readability & Maintainability.

Ex: maheshmath.py

```
1)  x=333
2)  def add(a,b):
3)          return a+b
4)  def product(a,b):
5)          return(a*b)
```

-->py maheshmath.py
-->maheshmath module contains one variable and 2 functions.
-->If we want to use memebers of the module in our program then we should import that module.

```
import modulename
```

-->We can access members by using module name.

```
modulename.variable
modulename.function()
```

test.py:

```
1)  import maheshmath
2)  print(maheshmath.x)
3)  print(maheshmath.add(10,20))
4)  print(maheshmath.product(10,20))
```

o/p:
333
30
200

Note:
Whenever we are using a module in our program, for that module compiled file will be generated and stored in the hard disk permanently

__pycache__ folder

NARESH IT, 2nd Floor, Durga Bhavani Complex, Satyam Threater Rd, Ameerpet, HYD
Telangana-500016 | Tel:+91 40 2374 6666 | Ph:+91 90009 94007 | www.nareshit.in

Renaming a module at the time of import(module aliasing):

Ex:

```
import maheshmath as m
```

-->Here maheshmath is original module name and m is alias name.

-->We can access memebrs by using alias name m.

test.py

```
1) import maheshmath as m
2) print(m.x)
3) print(m.add(10,20))
4) print(m.product(10,20))
```

from.....import:

-->We can import a particular member of module by using from....import

-->The main advantage of this is we can access memebrs directly without using module name.

Ex:

```
1) from maheshmath import x,add
2) print(x)
3) print(add(10,20))
4) print(product(10,20))==>Name Error:name 'product' is not defined
```

-->We can import all the members of a module as follows

```
from maheshmath import *
```

Ex:

```
1) from maheshmath import *
2) print(x)
3) print(add(10,20))
4) print(product(10,20))
```

member aliasing:

```
1) from maheshmath import x as y,add as sum
2) print(y)
3) sum(10,20)
```

-->Once we defined as alias name, we should use alias name only and we should not use original name.

Ex:

```
1)      from maheshmath import x as y
2)      print(x)==>Name Error:name 'x' is not defined
```

Various possibilities of import:
---

1) import modulename
2) import module1,module2,module3
3) import module1 as m
4) import module1 as m1,module2 as m2,module3 as m3
5) from module import memeber
6) from module import memeber1,member2,member3
7) from module import memeber1 as x
8) from module import *

Reloading a Module:

-->Bydefault module will be loaded only once eventhough we are importing multiple times.

Ex: module1.py

```
print("This is from module1")
```

test.py

```
1) import module1
2) import module1
3) import module1
4) import module1
5) print("This is test module")
```

o/p:
This is from module1
This is from test module

-->In the above program test module will be loaded only once eventhough we are importing multiple times.

-->The problem in this approach is after loading a module if it is updated outside then updated version of module1 is not available to our program.
-->We can solve this problem by reloading module explicitly based on our requirement.
-->We can reload by using reload() function of imp module.

```
import imp
imp.reload(module1)
```

Ex:

```
1) import time
2) import module1
3) print("program entering into sleeping state")
4) time.sleep(30)
5) import module1
6) print("This is last line of the program")
```

```
o/p:D:\pythonclasses>py test.py
This is from module1
program entering into sleeping state
This is last line of the program
```

Ex:

```
1) import time
2) from imp import reload
3) import module1
4) print("program entering into sleeping state")
5) time.sleep(30)
6) reload(module1)
7) print("program entering into sleeping state")
8) time.sleep(30)
9) reload(module1)
10) print("This is last line of the program")
```

```
o/p:
This is from module1
program entering into sleeping state
This is first module1
program entering into sleeping state
This is second module1
This is last line of the program
```

-->The main advantage of explicit module reloading is we can ensure that updated version is always available to our program.

Finding members of module by using dir() function:

-->Python provides inbuilt function dir() to list out all the members of current module or a specified module.

dir()==>To list out all members of current module.
dir(module name)==>To list out all the members of specified module.

Ex:test.py

```
1)  x=333
2)  def add(a,b):
3)          return a+b
4)  print(dir())
```

o/p:D:\pythonclasses>py test.py
['__annotations__', '__builtins__', '__cached__', '__doc__', '__file__', '__loader__', '__name__', '__package__', '__spec__', 'add', 'x']

Ex-2: To display members of a particular module.

maheshmath.py

```
1)  x=333
2)  def add(a,b):
3)          return a+b
4)  def product(a,b):
5)          return(a*b)
```

test.py

```
1)  import maheshmath
2)  print(dir(maheshmath))
```

o/p:D:\pythonclasses>py test.py
['__builtins__', '__cached__', '__doc__', '__file__', '__loader__', '__name__', '__package__', '__spec__', 'add', 'product', 'x']

Note: For every module at the time of execution python interpreter will add some special properties automatically for internal use.

Ex: '__builtins__', '__cached__', '__doc__', '__file__', '__loader__', '__name__', '__package__', '__spec__',

-->Based on our requirement we can access these properties also in our program.

Ex: test.py

```
1) print(__builtins__)
2) print(__cached__)
3) print(__doc__)
4) print(__file__)
5) print(__loader__)
6) print(__name__)
7) print(__package__)
8) print(__spec__)
```

```
o/p:D:\pythonclasses>py test.py
<module 'builtins' (built-in)>
None
None
test.py
<_frozen_importlib_external.SourceFileLoader object at 0x000001F562583CF8>
__main__
None
None
```

The special variable __name__:

-->For every python program, a special variable __name__ will be added internally.This variable stores information regarding whether the program is executed as an individual program or as a module.

-->If the program executed as an individual program then the value of this variable is __main__.

-->If the program executed as a module from some other program then the value of this variable is the name of the module where it is defined.

-->Hence by using this __name__ variable we can identify whether the program executed directly or as a module.

Ex: module1.py

```
1)  def f1():
2)      if __name__=="__main__":
3)              print("The code executed directly as a program")
4)              print("The value of __name__:",__name__)
5)      else:
6)              print("The code executed indirectly as a module from some other module")
7)              print("The value of __name__:",__name__)
8)  f1()
```

o/p:D:\pythonclasses>py module1.py
The code executed directly as a program
The value of __name__: __main__

test.py

```
1)  import module1
2)  module1.f1()
```

o/p:D:\pythonclasses>py test.py
The code executed indirectly as a module from some other module
The value of __name__: module1
The code executed indirectly as a module from some other module
The value of __name__: module1

working with math module:

-->Python provides inbuilt module math
-->This module defines several functions which can be used for mathematical operations.
-->The main important functions are:
sqrt(x), ceil(x), floor(x), log(x), sin(x), tan(x), fabs(x)

Ex:

```
1)  from math import *
2)  print(sqrt(4))          #2.0
3)  print(ceil(10.1))       #11
4)  print(floor(10.1))      #10
5)  print(fabs(-10.6))      #10.6
6)  print(fabs(10.6))       #10.6
```

Note: We can find help of any module by using help() function

Ex:

```
1) import math
2) help(math)
```

Working with random module:

-->This module defines several functions to generate random numbers.

-->We can use these functions while developing games, in cryptography and to generate numbers on fly for authentication.

1).random():

This function always generates some float value between 0 and 1(not inclusive).

0<x<1

Ex:

```
1) from random import *
2) for i in range(10):
3)         print(random())
```

```
o/p:D:\pythonclasses>py test.py
0.7690917847505055
0.12958902402812178
0.4865336117685294
0.21174369485166067
0.22533721686631736
0.24270745076560807
0.6156350105877338
0.3588237251403473
0.5609192891678808
0.46565274922504374
```

2).randint():

To generate random integer between two given numbers(inclusive)

Ex:

```
1) from random import *
2) for i in range(10):
```

```
3)          print(randint(1,100))#Generates random int values between 1 and
    100(inclusive)
```

```
o/p:D:\pythonclasses>py test.py
70
88
20
73
16
8
27
72
80
71
```

-->Generate 5 6-digit random numbers
Ex:

```
1)  from random import *
2)  for i in range(5):
3)          print(randint(0,9),randint(0,9),randint(0,9),randint(0,9),randint(0,9),randint
    (0,9),sep='')
```

```
o/p:D:\pythonclasses>py test.py
420863
352322
626559
689042
735553
```

3).uniform():

-->It returns random float values between 2-given numbers(not inclusive)

Ex:

```
1)  from random import *
2)  for i in range(10):
3)          print(uniform(1,10))
```

```
o/p:D:\pythonclasses>py test.py
1.456481671599132
8.262379808015648
```

8.294591177579873
2.4766196390802415
3.9929683121049644
6.908124318470733
5.113015507787097
1.9677692845675518
8.48400436311528
7.760067312991328

random()==>in between 0 and 1(not inclusive)
randint()==>in between z and y(incluseve)
uniform()==>in between x and y(not inclusive)

4).randrange([start],stop,[step]):

-->Returns a random number from the range.
start<=x<stop
-->start argument is optional and default value is 0.
-->step argument is optional and default value is 1.

randrange(10)==>generates a number from 0 to 9
randrange(1,11(==>generates a number from 1 to 10
randrange(1,11,2)==>generates a number from 1,3,5,7,9

Ex 1:

```
1) from random import *
2) for i in range(10):
3)         print(randrange(10))
```

Ex 2:

```
1) from random import *
2) for i in range(10):
3)         print(randrange(1,11))
```

Ex 3:

```
1) from random import *
2) for i in range(10):
3)         print(randrange(1,11,2))
```

5).choice():

-->It wont return random number.

-->It will return a random object from the given list or tuple,str.

Ex 1:

```
1) from random import *
2) list=["sunny","bunny","chinny","vinny","pinny"]
3) for i in range(10):
4)         print(choice(list))
```

Ex 2:

```
1) from random import *
2) list=("sunny","bunny","chinny","vinny","pinny")
3) for i in range(10):
4)         print(choice(list))
```

Ex 3:

```
1) from random import *
2) list={"sunny","bunny","chinny","vinny","pinny"}
3) for i in range(10):
4)         print(choice(list))
```

o/p:D:\pythonclasses>py test.py
TypeError: 'set' object does not support indexing

Ex 4:

```
1) from random import *
2) for i in range(10):
3)         print(choice("mahesh"))
```

Ex: To generate random pwd of 6-length where as 1,3,5 are aplhabate symbols and 2,4,6 are digits.

```
1) from random import *
2) for i in range(5):
```

```
3)          print(chr(randint(65,65+25)),randint(0,9),chr(randint(65,65+25)),randint(0,9
    ),chr(randint(65,65+25)),randint(0,9),sep='')
```

```
o/p:D:\pythonclasses>py test.py
L0Z4J6
X1M8N6
A8L2U1
V7O2Z5
S9U2S1
```

PACKAGES

-->It is an encapsulation mechanism to group related modules into a single unit.
-->Package is nothing but folder or directory which represents collection of python modules.
-->Any folder or directory contains __init__.py file, is considered as a python package. This can be empty.
-->A package can contains sub packages also.
-->Python 3.3 has implicitly namespace packages that allows to create a package without __init__.py file.

-->The main advantages of package statements are:
1).We can resolve naming conflicts.
2).We can identify our components uniquely
3).It improves modularity of the application.

Ex1:

D:\pythonclasses
|--test.py
|--pack1
|-module1.py
|-__init__.py

__init__.py:
---
empty file.

module1.py:

```
def f1():
    print("Hello this is from module1 present in pack1")
```

test.py(version-1):

```
import pack1.module1
pack1.modul1.f1()
```

o/p: Hello this is from module1 present in pack1

NARESH IT, 2nd Floor, Durga Bhavani Complex, Satyam Threater Rd, Ameerpet, HYD
Telangana-500016 | Tel:+91 40 2374 6666 | Ph:+91 90009 94007 | www.nareshit.in

test.py(version-2):

```
-------------------------
from pack1.module1 import f1
f1()
```

Ex 2:

```
-------
D:\pythonclasses
        |--test.py
        |--com
                |--module1.py
                |--__init__.py
                        |--durgasoft
                                |--module2.py
                                |--__init__.py
```

__init__.py:

```
----------------
        empty file
```

module1.py:

```
-----------------
def f1():
        print("Hello this is from module1 present in com")
```

module2.py:

```
-----------------
def f2():
        print("Hello this is from module 2 present in com.durgasoft")
```

test.py:

```
----------
```

```
1) from com.module1 import f1
2) from com.durgasoft.module2 import f2
3) f1()
4) f2()
```

o/p:

```
Hello this is from module1 present in com
Hello this is from module 2 present in com.durgasoft
```