==

Real Time Applications of Python

===

=>With Programming, we can develop 22+ Real Time Applications

1) Web Applications Development.---->
    - a) Java------>Servlets , JSP
    - b) C#.net---->ASP.net
    - c) Python---->Django,Falsk, Bottle...etc
2) Gaming Application Development.
3) Artificial Intelligence-----Machine Learning and Deep Learning
4) Desk top GUI Applications
5) Image Processing applications.
6) Text Processing Applications
7) Business Applications.
8) Audio and Video Based Applications
9) Web Scrapping Applications / Web Harvesting Applications
10) Data Visulization.
11) Complex Math Calculations.
12) Scientific Applications
13) Software Development
14) Operating System
15) CAD and CAM based Applications
16) Embedded Applications
17) IOT Based Applications
18) Language Applications
19) Automation of Testing
20) Animation Applications
21) Data Analysis and Data Analystics
22) Education Sector
23) Computer Vision

===

Getting started with Python

===

=>History of Python
=>Versions of Python
=>Downloading Process of Python

===

=>History of Python

=>Python Programming language foundation stone laid in the year 1980.
=>Python Programming language implemetation started in the year 1989.
=>Python Programming language officially released in the year 1991 Feb.
=>Python Programming language developed By GUIDO VAN ROSSUM.
=>Python Programming language developed at CWI Institute in Nether lands.
=>ABC programming language is the Predecessor of Python Programming language.

---x----------------------

=>Versions of Python

====================

=>Python Programming Contains two Versions. They are

1) Python 2.x----- Here x ---> 1 2 3 4 5 6 7 -----outdated---
2) Python 3.x----> here x 1 2 3 4 5 4   6   7   8   9   10

=>Python 3.x does not contain backward compatability with Python 2.x
=>To down load Python 3.x software , we use www.python.org
=>Python Software and its updations are maintained by a Non-Commerical Organization called " Python Software Foundation(PSF) "

Python Programming Inspired from

=>Python Programming Inspired from 4 programming language

1) Functional Programming from C
2) Object Oriented Programming from CPP
3) Scripting Programming from PERL
4) Modular Programming from Modulo3

Features of Python Programming

=>Features of a language are nothing but services / Facilities provided language developers and they are used by language programmers for developing real time applications.
=>Python Programming Provides 11 features.

1. Simple
2. Freeware and Open Source
3. Platform Independent
4) Dynamically Typed
5) Portable
6) Interpreted
7) High Level
8) Robust(Strong)
9) Extensible
10) Embedded
11) Extensive Third Party Library / API support
(Numpy,Pandas, Matplotlib,scikit, scipy...etc)
12) Both Procedure oriented(Core Python) and Object Oriented (Adv Python)

1. Simple

=>Python is one of the SIMPLE programming, bcoz of 3 Important Tech Factors.

a) Python Programming Provides "Rich Set of APIs". So that Python Programmer can Re-Use the pre-defined Libraries / API for solving real time requirements.

Definition of API (Application Programming Interface):

=>An API is a collection Modules.
=>A Module is a collection of Functions, Variables and Classes
Examples:- math, cmath, random,calendar, re, cx_Oracle, mysql-connector, threading, gc....etc

b) Python Programming Provides Inbuilt "Garbage Collection " Facility. So that It collects un-used memory space and improves performance of Python Based Applications.

Def of Garbage Collector:

Garbage Collector is one of the In-built Program in Python Software, which is running behind of every Regular Python Program and whose purpose is that to Collect Un-Unsed / Un-referenced Memory space and Improves the Performnace of Python Based Applications.

----

c) Python Programming Privdes User Friendly Syntaxes. So that Python Programmer can develop Error-Free Program in a limited span of time.

============================X==

==

Freeware and Open Source

==

=>Freeware:

=> If any software is available Freely Downlodable then it called FreeWare.

Examples:- PYTHON and JAVA

=>The Python which we down load from www.python.org is called Standard Python and Whose name Is "CPYTHON"

=>Open Source:

=>Some of the Companies Came forward and customized CPYTHON for Their In-House Requirments and those Open Source Software of python are called "Python Distributions".

=>Some of the Python Distributions are :

1) JPYTHON (or) JYTHON---->Used To Run Java Based Applications.
2) Iron Python---------------------->Used To run C#.net Based Application
3) Micro Python----------->Used To develop Micro Controller Applications
4)Ruby Python------------>Used to run RUBY ON RAIL based Applications
5) Anakonda Python--->Used deal with BIGDATA / Haddop Based Appls.

============================

Freeware and Open Source

============================

=>Freeware:

=> If any software is available Freely Downlodable then it called FreeWare.

Examples:- PYTHON and JAVA

=>The Python which we down load from www.python.org is called Standard Python and Whose name Is "CPYTHON"

=>Open Source:

=>Some of the Companies Came forward and customized CPYTHON for Their In-House Requirments and those Open Source Software of python are called "Python Distributions".
=>Some of the Python Distributions are :

1) JPYTHON (or) JYTHON---->Used To Run Java Based Applications.
2) Iron Python--------------------->Used To run C#.net Based Application
3) Micro Python----------->Used To develop Micro Controller Applications
4)Ruby Python------------>Used to run RUBY ON RAIL based Applications
5) Anakonda Python--->Used deal with BIGDATA / Haddop Based Appls.

3. Platform Independent

Concept / Definition:

=>A language is said to be Platform Independent iff whose applications / Programs runs on every OS

Property :

=>The property of Platform Independent in Python is that "All the Values in Python Stored in the form Objects and Objects conatins unlimitedf amount of data storage" . So that run on any OS.

=>In Python Programming all values are stored in the fom Objects.

Portable

=>A Portable Project is one which can run on all types of OSes with Considering vendors and their Architectures.
Examples:--- PYTHON , JAVA

Example for NON-portable: C,CPP...etc

7) High Level

=>Even though we represent the data in the form Binary , Octal and Hexa Decimal Format and at output stage we are getting the output in high level Understandable Format.
=>Understanding python statements is Simple.

6) Interpreted

=>When we run the python program, Two internal steps are taking place. They are

1) Compilation Process:

The Python Compiler Converts .py (Source Code) into .pyc Code(Byte Code) in the form Line by Line.

Example: sum.py-------->sum.pyc----during Compile Time

2) Execution Phase:

=>The PVM reads Line by Line of Byte Code and converted into Machine Understandable Code(Binary Code) and It is read By OS and Processer and Gives Result.
=>Hence In Pyhon Execution Environment, Compilation Process and Execution is Performing Line by Line anf Python is One of the Interpreted Programming.

==

Extensible and Embedded

===

Extensible:

=>Since Python Programming Provides its services (Programming Segments / snippets) to other languages for fullfillung its requiements easily.
Examples:-C Programs-can call The coding segments of PYTHON.

Embedded:

=>Since Python programming cal also the call / utilize the services of C, Other Languages as part of its development and Hence Python is onbe of the Embedded Programming Languages.
Examples:--Numpy, Scikit,Pandas,Scipy, matplot lib etc these developed in Python and Uses C language.

===

11) Extensive Third Party Library (or) API support

===

=>With Traditional Python Programming APIs, we may not be able to perform complex operations. To do these complex Operations , we use Third party Libraries and Some of the Third party Libraries are
Examples:- numpy,Pandas,scipy,scikit,matplot lib...etc

===

4) Dynamically Typed

===

=>We have two types of Programming Languages. They are
1. Static Typed Programming Languages
2. Dynamically Typed Programming Languages

1. Static Typed Programming Languages:

=>In This Programming Languages, Data type of values must specified by programmer explicitly. Otherwise we get Errors
Examples:

C,CPP, JAVA, .NET...etc

```
Examples:     int a=10;
               int b=20;
              int c=a+b;
```

----

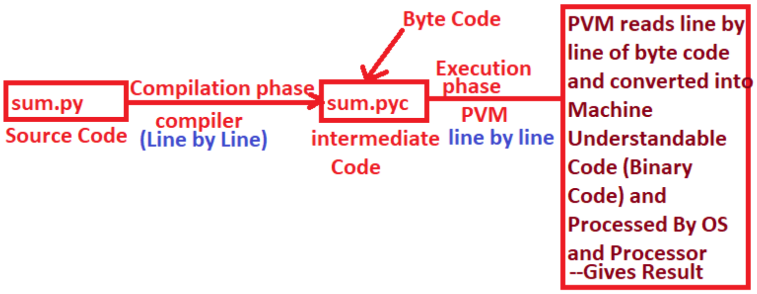
2. Dynamically Typed Programming Languages:

=>In This Programming Languages, Data type of the values need not specify by the programmer and more over data type of the value is implicitly decided by Python Execution Environment.

=>In Python Programming , all values are stored in the form of Objects and to cerate objects we need classes.

Examples: PYTHON

```
Examples:
--------------------
>>> a=100
>>> b=200
>>> c=a+b
>>> print(a,b,c)-------------------------100 200 300
>>> print(type(a), type(b),type(c))-------<class 'int'> <class 'int'> <class
'int'>
>>> print(a,type(a))--------------100 <class 'int'>
>>> print(b,type(b))------------200 <class 'int'>
>>> print(c,type(c))-------------300 <class 'int'>
```



Literals in PythonVariables (or) Identifiers in Python

Rules for Using Variables in Python

Data Types in Python

```
                    =============================================
                            Data Representation in Python
                                         (or)
                                  Literals in Python
                     =============================================
```

=>Literals are nothing but values used for giveing inputs to the program.
=>Basically we have 4 types of Literals. They are

a) Integer Literals
b) Float Literals
c) String Literals
d) Boolean Literals.

=>In general to represent / store any type of Literals / Data in main memory of computer, we need objects.

===

Rules for Using Variables in Python

===

=>To use the Variables in Python Programming, we must follow the rules. They are

1) The Variable Name is a comibination of Alphabets (Lower and upper Case), Digits and Special Symbol Under Score (_)
2) The Variable Name must starts with Either with an alphabet or Under Score (_)

```
Examples:
----------------
               12abc=10-----invalid
               -abc=20-------invalid
               abc=123-----valid
               a123=34----valid
               _abc=34---valid
               _sal_=2.3--valid
               _123=2.3---valid
               _=23----valid
```

3) Within in the Variable Name , special symbols are not allowed except Under Score (_)

```
Examples:
----------------
               tot  sal=2.3---invalid
               tot$sal=2.3--invalid
               tot_sal=2.3--valid
```

4) All the Variables in Python are Case Sensitive.

```
Examples:
----------------
                      age=99---valid
                      AGE=89---valid
                      Age=79---valid
```

5) Keywords can't be used as Variables Names bcoz all the Key words are Reserved Words they have some specfic meaning to the language Compilers.

```
Examples:
-----------------
                      if=12---invalid
                      while=23---invalid
                      else=45---invalid
                      if123=56---valid
                      _while=34----valid
                      IF=45----valid
                      int=12.34---valid
                      float=45----valid
```

6) All the Variable Names are recommended to Take User-Friendly Names.

Examples:-

```
>>> sal_of_an_employee=1.2--Valid--Not Recommended
>>> emp_sal=1.2--Valid--Recommended
```

=========================X==================================

Variables (or) Identifiers in Python

=>All types of Literals are stored in Main memory in the created memory space. To process the values stored in main memory, as programmer, we must give distinct names to the cerated memory space. So that distinct names makes us to identify the values and hence they are called Identifiers.
=>Identifier values are changing / Varying during the program execution ands hence Identifier are called Variables.
=>In Python all types of Literals / Values are stored in Main Memory in the form Variables / Identifiers and all types of Variables / Identifiers are called objects.

=>Def. of Variable:-

=>A Variable is an Identifier whose values are changing during execution of the program.

---X--------------------

Data Types in Python

=>The purpose of Data types in Python is that to allocate sufficient memory space for input values and performs Various Operations on the data
=>In Python Programming, we have 14 Data Types. They are

```
I) Fundamental Catagery Data Types
        i) int
        ii) float
        iii) bool
        iv) complex
II) Sequence Catagery Data Types
        i) str
        ii) bytes
        iii)bytearray
        iv) range
III) List Catagery Data Types (Collection Data Types)
        i) list
        ii) tuple
IV) Set Catagery Data Types  (Collection Data Types)
        i) set
        ii) frozenset
V) Dict Catagery Data Types  (Collection Data Types)
        i) dict
VI) None Catagery Data Type:
        i)   None
```

I) Fundamental Catagery Data Types

=>The purpose of Fundamental Catagery Data Types is that to store Single Value but they never allows us to store Multiple Values of same type or different type.

=>In Python Programming, we have 4 data Types Fundamental Catagery. They are

i) int
ii) float
iii) bool
iv) complex

Base Conversion Functions

=>The purpose of Base Conversion Functions is that to Convert One Base value into another base value.

=>In Python , we have 3 Base Conversion Functions. They are

a) bin()
b) oct()
c) hex()

a) bin():

=>This Function is used for converting any type of base value into binary number system value.

```
=>Syntax:-      varname=bin(decimal / octal / hexa decimal value)
```

Examples:

```
>> a=15
>>> print(a,type(a))---------15 <class 'int'>
>>> b=bin(a)
>>> print(b,type(b))---------0b1111 <class 'str'>
>>> a=0o14
>>> print(a,type(a))---------------12 <class 'int'>
>>> b=bin(a)
>>> print(b,type(b))-------------0b1100 <class 'str'>
>>> a=0xA
>>> print(a,type(a))----------------10 <class 'int'>
>>> b=bin(a)
>>> print(b,type(b))--------------  0b1010 <class 'str'>
```

b) oct():

=>This Function is used for converting any type of base value into octal number system value.

```
=>Syntax:-      varname=oct(decimal / binary / hexa decimal value)
```

Examples:

```
>>> a=12
>>> print(a,type(a))----------------12 <class 'int'>
>>> b=oct(12)
>>> print(b,type(b))--------------0o14 <class 'str'>
>>> a=0b1111
>>> print(a,type(a))-------------15 <class 'int'>
>>> b=oct(a)
>>> print(b,type(b))------------0o17 <class 'str'>
>>> a=0XACC
>>> print(a,type(a))-----------2764 <class 'int'>
>>> b=oct(a)
>>> print(b,type(b))------------0o5314 <class 'str'>
```

----

c) hex():

=>This Function is used for converting any type of base value into hexa Decimal number system value.

=>Syntax:-   varname=hex(decimal / binary / octal value)

Examples:

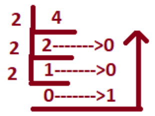
```
>>> a=2764
>>> print(a,type(a))---------------2764 <class 'int'>
>>> b=hex(a)
>>> print(b,type(b))--------------0xacc <class 'str'>
>>> b=hex(15)
>>> print(b,type(b))-----------0xf <class 'str'>
>>> a=0o15
>>> print(a,type(a))----------13 <class 'int'>
>>> b=hex(a)
>>> print(b,type(b))------------0xd <class 'str'>
>>> a=0b1010
>>> print(a,type(a))------------10 <class 'int'>
>>> b=hex(a)
>>> print(b,type(b))----------- 0xa <class 'str'>
```

==========================X=====================================

Convert Decimal data into Binary Data

$(4)_{10}$----------------->$(x)_2$ find x=0100

Sol:



2 | 4

2 | 2------->0

2 | 1------->0

0------->1

hence $(4)_{10}$------------>$(0100)_2$

Convert Binary Data into Decimal Data

$(0100)_2$--------------$(x)_{10}$ find x here x=4

Sol:-

0 1 0 0

3 2 1 0

2 2 2 2

$= 0 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 0 \times 2^0$

= 0 + 4+ 0+0

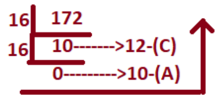
= 4

hence $(0100)_2$------->$(4)_{10}$

Convert Deciaml Data into Hexa decimal Data

Q1) $(172)_{10}$---------------$(x)_{16}$ find x x=0AC

Sol:



16 | 172

16 | 10------->12-(C)

0---------->10-(A)

hence $(172)_{10}$---------->$(AC)_{16}$

Convert Hexa Decimal data into Decimal data

Q1) $(AC)_{16}$----------------->$(x)_{10}$ find x here x=172

Sol:-

A C

1 0

16 16

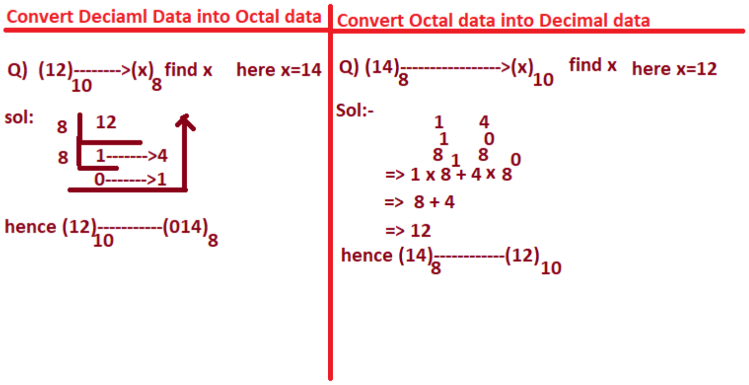
$= A \times 16^1 + C \times 16^0$

= 10x16 + 12x1

=160+12

=172

hence $(AC)_{16}$--------->$(172)_{10}$


Convert Deciaml Data into Octal data
Q) (12)10-------->(x)8 find x here x=14
sol:
8 | 12
8 | 1------->4
0------->1
hence (12)10-----------(014)8
Convert Octal data into Decimal data
Q) (14)8----------------->(x)10 find x here x=12
Sol:-
1 4
1 0
8 1 8 0
=> 1 x 8 + 4 x 8
=> 8 + 4
=> 12
hence (14)8------------(12)10

===

Operations on Strings

===

=>On the String data, we can two types of Operations. They are

a) Indexing

b) Slicing

----

a) Indexing

=>The Process of obtaining one value at a time from given string object is called Indexing.

=>In Python Programming , we have two types of Indices (or Indexes) . They are

a) Forward Indexing and starts from Left to Right (0,1,2.......)

b) Backward Indexing and starts from Right to Left (-1, -2 -3.......)

=>Syntax:

```
                     strobj [ Index ]
```

=>index represents either Possitive and Negative Index.

=>f we enter Invalid Index then we get "IndexError".

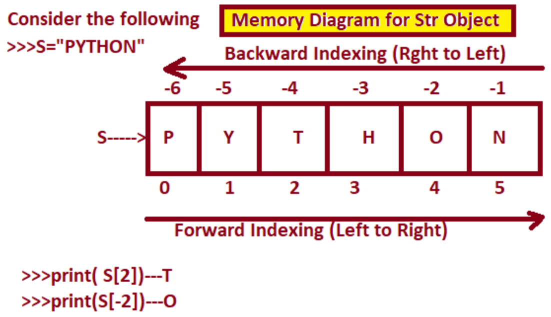
Examples:

```
>>> s="PYTHON"
>>> print(s[0])----------P
>>> print(s[-6])---------P
>>> print(s[-1])----------N
>>> print(s[5])-----------N
>>> print(s[3])----------H
>>> print(s[-4])----------T
>>> print(s[10])---------IndexError: string index out of range
>>> print(s[-10])----IndexError: string index out of range
===============================================================
b) Slicing:
--------------------------------
=>The process of obtaining range of characters (or) sub string from given
string object is called  String Slicing.

=>Syntax1:-    strobj [  Begin : End ]

=>This Syntax obtaing the data from Begin Index Value to End Index-1 Value
provided  Begin Index<End Index otherwise we never get Output (Empty).

Examples:
-----------------
>>> s="PYTHON"
>>> print(s[3:6])------------HON
>>> print(s[6:3]))----------- empty output
>>> print(s[-6:-3])----------PYT
>>> print(s[2:5])---------THO
>>> print(s[-4:-1])-------THO
--------------------------------------------------------------
              ========================================
                      Sequence Catagery Data Types
               ========================================
=>Sequence Catagery Data Types are used for storing Sequence of Values /
Multiple values of same type.
=>We have 4 types Sequence Catagery. They are

                      1) str
                      2) bytes
                      3) bytearray
                      4) range
```

```
                        ==============
                              str
                         ==============
Index:
-----------
=>Purpose of str
=>Types of Strings
=>Types String Organization and Notations
=>Operations on Strings
              a) Indexing
              b) Slicing
=============================================================
=>The collection or sequence of characters enclosed within single / double
Quotes  is called String (Python)          :
```

```
Examples:       "Python Proghramming"   "Guido Van Rossum"
                             "A"      'A'      'Java Programming'
=>'str' is one of the pre-defined class and treated as Sequence Data Type
=>The Purpose of str data type  is that "To store Sequence of values
within Single / Double  Quotes or tripple  single / double Quotes.
=>We have two types of String data. They are
                      a) Single Line String Data
                      b) Multi Line String data
----------------------------------
a) Single Line String Data
----------------------------------
=>Single Line String Data must be enclosed within Single or Double Quotes
or tripple single / double Quotes.
--------------------
Examples:-
--------------------
>>> a="Python Programming"
>>> print(a,type(a))----------Python Programming <class 'str'>
>>> b='A'
>>> print(b,type(b))--------------A <class 'str'>
>>> c="A"
>>> print(c,type(c))------------A <class 'str'>
>>> d='Java Programming'
>>> print(d,type(d))-----------Java Programming <class 'str'>
>>> crs1="Python Programming"
>>> print(crs1,type(crs1))------------Python Programming <class 'str'>
>>> crs2='Python Programming'
>>> print(crs2,type(crs2))-----------Python Programming <class 'str'>
>>> crs3="1234567"
>>> print(crs3,type(crs3))-------------1234567 <class 'str'>
>>> crs3="Python3.10"
>>> print(crs3,type(crs3))------------Python3.10 <class 'str'>
>>> x="$%#@&abc&*()"
>>> print(x,type(x))------------$%#@&abc&*() <class 'str'>

>>> x='''A'''
>>> y="""A"""
>>> a="""JAVA"""
>>> b='''PYTHON'''
>>> print(x,type(x))--------------A <class 'str'>
>>> print(y,type(y))------------A <class 'str'>
>>> print(a,type(a))-----------JAVA <class 'str'>
>>> print(b,type(b))-------PYTHON <class 'str'>

=>Hence  With Single and double Quotes we can organize / store single line
String data only but organize / store multi line String data.
Examples:
       >>> addr1="Guido van Rossum
                                    SyntaxError: unterminated string
literal
       >>> addr1=' Guido van Rossum
                                 SyntaxError: unterminated string literal
=>To organize multi line string data we must use Tripple Single or tripple
double Quotes.
```

```
--------------------------------------------------------------------------
b) Multi Line String Data
----------------------------------------
=>Multi Line String Data must be enclosed within  tripple single (or
tripple double Quotes.
-----------------
Examples:
----------------
>>> addr1="""Guido van Rossum
... HNO:3-4 Hill side
... CWI ,Python Soft Fund.
... Nether Lands--34567"""

>>> print(addr1,type(addr1))------
                                                   Guido van Rossum
                                                   HNO:3-4 Hill side
                                                   CWI ,Python Soft Fund.
                                                   Nether Lands--34567
<class 'str'>
>>> addr2='''James Gosling
... FNO: 45-56 River Side
... Sun Micro Sys,
... USA-12345678'''
>>> print(addr2,type(addr2))------------
                                                   James Gosling
                                                   FNO: 45-56 River Side
                                                   Sun Micro Sys,
                                                   USA-12345678 <class
'str'>

>>> x='''A'''
>>> y="""A"""
>>> a="""JAVA"""
>>> b='''PYTHON'''
>>> print(x,type(x))--------------A <class 'str'>
>>> print(y,type(y))------------A <class 'str'>
>>> print(a,type(a))-----------JAVA <class 'str'>
>>> print(b,type(b))-------PYTHON <class 'str'>
==================================X============================
```


Consider the following
Memory Diagram for Str Object
Backward Indexing (Rght to Left)
-6 -5 -4 -3 -2 -1
S----->
P Y T H O N
0 1 2 3 4 5
Forward Indexing (Left to Right)

```
>>>S="PYTHON"
>>>print( S[2])---T
>>>print(S[-2])---O
```

Type Casting techniques in Python
(or)
Type Conversion techniques in Python

=>The purpose of Type Casting techniques in Python is that "To Convert one data type value into another data type value".

=>In Python Programming, Fundamentally, we have 5 Type Casting techniques in Python. They are

1) int ()
2) float()
3) bool()
4) complex()
5) str()

3) bool()

=>bool() is used for converting "one possible type of value into bool type value."

=>Syntax:

```
                    varname=bool( int / floay / complex / str value )
```

=>ALL NON-ZERO Values are TRUE

=>ALL ZERO Values are FALSE

```
Examples: int value into bool--->Possible
--------------------------------------------------------------------------
-----------------------------
>>> a=120
>>> print(a,type(a))------------120 <class 'int'>
>>> b=bool(a)
>>> print(b, type(b))------------True <class 'bool'>
>>> a=-123
>>> print(a,type(a))------------123 <class 'int'>
>>> b=bool(a)
>>> print(b, type(b))-----------True <class 'bool'>
>>> a=0
>>> print(a,type(a))-------------0 <class 'int'>
>>> b=bool(a)
>>> print(b, type(b))-----------False <class 'bool'>
--------------------------------------------------------------------------
-----------------------------
Examples: float value into bool--->Possible
--------------------------------------------------------------------------
-----------------------------
>>> a=1.2
>>> print(a,type(a))---------1.2 <class 'float'>
>>> b=bool(a)
>>> print(b, type(b))--------True <class 'bool'>
>>> a=0.00000000000000000000000000000000000000001
>>> print(a,type(a))-----1e-41 <class 'float'>
>>> b=bool(a)
>>> print(b, type(b))-------True <class 'bool'>
>>> b=0.0
>>> print(b, type(b))--------0.0 <class 'float'>
>>> a=bool(b)
>>> print(a,type(a))---------False <class 'bool'>
--------------------------------------------------------------------------
--------------------------
Examples: comnplex value into bool--->Possible
--------------------------------------------------------------------------
-----------------------------
>>> a=2+3.5j
>>> print(a,type(a))-------------(2+3.5j) <class 'complex'>
>>> b=bool(a)
>>> print(b, type(b))------------True <class 'bool'>
>>> a=0+0j
>>> print(a,type(a))------------0j <class 'complex'>
>>> b=bool(a)
>>> print(b, type(b))----------False <class 'bool'>
--------------------------------------------------------------------------
----------------------------
Examples:str values into bool------Possible
--------------------------------------------------------------------------
-----------------------------
>>> a="123"  #int str
>>> print(a,type(a))
123 <class 'str'>
>>> b=bool(a)
>>> print(b, type(b))
True <class 'bool'>
>>> a="0" # int str
>>> print(a,type(a))
0 <class 'str'>
>>> b=bool(a)
>>> print(b, type(b))
True <class 'bool'>
>>> a="0000"
>>> print(a,type(a))
0000 <class 'str'>
>>> b=bool(a)
>>> print(b, type(b))
True <class 'bool'>
>>> a="KVR"
>>> print(a,type(a))
KVR <class 'str'>
>>> b=bool(a)
>>> print(b, type(b))
True <class 'bool'>
>>> a=" "
>>> print(a,type(a))
  <class 'str'>
>>> b=bool(a)
>>> print(b, type(b))
True <class 'bool'>
>>> len(a)
1
>>> a=""
>>> print(a,type(a))
 <class 'str'>
>>> b=bool(a)
>>> print(b, type(b))
False <class 'bool'>
>>> len(a)
0
>>> a="False"
>>> print(a,type(a))
False <class 'str'>
>>> b=bool(a)
>>> print(b, type(b))
True <class 'bool'>
>>> a=False
>>> b=bool(a)
>>> print(b, type(b))
False <class 'bool'>
>>> a=0e0
>>> print(a,type(a))
0.0 <class 'float'>
>>> b=bool(a)
>>> print(b, type(b))
False <class 'bool'>
==============================X===============================
                          ==========================
                                4) complex()
                          ===========================
=>This function is used for converting one possible type of value into
complex type value.
-----------------
=>Syntax: varname=complex(int / float / bool / str value)
----------------
----------------------------------------------------------
Examples:      int value-->complex--->Possible
----------------------------------------------------------
>>> a=10
>>> print(a,type(a))----------------10 <class 'int'>
>>> b=complex(a)
>>> print(b, type(b))-------------(10+0j) <class 'complex'>
----------------------------------------------------------
Examples:      float value-->complex--->Possible
----------------------------------------------------------
>>> a=12.3
>>> print(a,type(a))------------12.3 <class 'float'>
>>> b=complex(a)
>>> print(b, type(b))------------(12.3+0j) <class 'complex'>
-------------------------------------------------------------------------
Examples:      bool value-->complex--->Possible
-------------------------------------------------------------------------
>>> a=True
>>> print(a,type(a))-------------True <class 'bool'>
>>> b=complex(a)
>>> print(b, type(b))---------------(1+0j) <class 'complex'>
--------------------------------------------------------------------------
Examples:      Str value-->complex
--------------------------------------------------------------------------
-----------
>>> a="12"  # int str---->complex-->Possible
>>> print(a,type(a))---12 <class 'str'>
>>> b=complex(a)
>>> print(b, type(b))---(12+0j) <class 'complex'>
>>> a="2.3"   # float str---->complex-->Possible
>>> print(a,type(a))---------2.3 <class 'str'>
>>> b=complex(a)
>>> print(b, type(b))--------(2.3+0j) <class 'complex'>
>>> a="True"  # bool str---->complex-->Not Possible
>>> print(a,type(a))---------True <class 'str'>
>>> b=complex(a)--------ValueError: complex() arg is a malformed string
>>> a="Python"   # Pures Str--->complex--Not Possible.
>>> print(a,type(a))------Python <class 'str'>
>>> b=complex(a)------ValueError: complex() arg is a malformed string
```

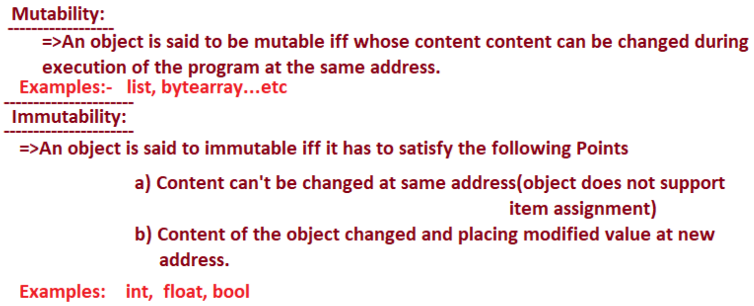
```
                                 =================
                                      float()
                                 ==================
=>float() is used for converting "one possible type of value into  float
type value."
=>Syntax:
------------------
                      varname=float( int / bool / complex / str value )
---------------------------------------------------------------------------
Example:     int value into float--->Possible
---------------------------------------------------------------------------
>>> a=12
>>> print(a,type(a))-----------------12 <class 'int'>
>>> b=float(a)
>>> print(b, type(b))----------------12.0 <class 'float'>
---------------------------------------------------------------------------
Example:     bool value into float--->Possible
--------------------------------------------------------------------------
>>> a=True
>>> print(a,type(a))----------True <class 'bool'>
>>> b=float(a)
>>> print(b, type(b))----------1.0 <class 'float'>
>>> a=False
>>> print(a,type(a))-----------False <class 'bool'>
>>> b=float(a)
>>> print(b, type(b))-----------0.0 <class 'float'>
---------------------------------------------------------------------------
Example:    complex value into float--->Not Possible
--------------------------------------------------------------------------
>>> a=2+3j
>>> print(a,type(a))-----------(2+3j) <class 'complex'>
>>> b=float(a)-------TypeError: float() argument must be a string or a
real number, not 'complex'
>>> b=float(a.real)
>>> print(b, type(b))----2.0 <class 'float'>
>>> b=float(a.imag)
>>> print(b, type(b))---------3.0 <class 'float'>
-------------------------------------------------------------------------
Example:  Attempting  to Convert Str value into  float type
--------------------------------------------------------------------------
>>> a="12"   # int str into float--Possible
>>> print(a,type(a))--------------12 <class 'str'>
>>> b=float(a)
>>> print(b, type(b))---------12.0 <class 'float'>
>>> a="12.34"  # float str into float--Possible
>>> print(a,type(a))-------12.34 <class 'str'>
>>> b=float(a)
>>> print(b, type(b))----------12.34 <class 'float'>
>>> a="True"   # bool str into   float--Not Possible
>>> print(a,type(a))--------True <class 'str'>
>>> b=float(a)----ValueError: could not convert string to float: 'True'
>>> a="2.3+3.4j"----#complex str into  float----Not Possible
>>> print(a,type(a))-----2.3+3.4j <class 'str'>
>>> b=float(a)--------ValueError: could not convert string to float:
'2.3+3.4j'
>>> a="PYTHON.JAVA"   # Pure Str into float--Not Possible
>>> print(a,type(a))---------PYTHON.JAVA <class 'str'>
>>> b=float(a)----------ValueError: could not convert string to float:
'PYTHON.JAVA'
==========================X================================
                                ================
                                    1) int()
                                 ================
=>int() is used for converting "one possible type of value into  int type
value."
=>Syntax:
------------------
                      varname=int( float / bool / complex / str value )

--------------------------------------------------------------------------
Examples: --Converting float value into int value--->Possible
--------------------------------------------------------------------------
>>> a=12.34
>>> print(a,type(a))----------------12.34 <class 'float'>
>>> b=int(a)
>>> print(b, type(b))------------12 <class 'int'>
--------------------------------------------------------------------------
Examples: --Converting bool value into int value--->Possible
-------------------------------------------------------------------------
>>> a=True
>>> print(a,type(a))------------True <class 'bool'>
>>> b=int(a)
>>> print(b, type(b))---------1 <class 'int'>
>>> a=False
>>> print(a,type(a))---------False <class 'bool'>
>>> b=int(a)
>>> print(b, type(b))---------0 <class 'int'>
-------------------------------------------------------------------------
Examples: --Converting compex value into int value--->Not Possible
--------------------------------------------------------------------------
>>> a=2+3j
>>> print(a,type(a))-------------(2+3j) <class 'complex'>
>>> b=int(a)----TypeError: int() argument not 'complex'--Invalid
--------------------------------------------------------------------------
Examples: -- Attempting  to Convert Str value into  int type
-------------------------------------------------------------------------
>>> a="123"   # int string  into   int Possible
>>> print(a,type(a))-------------123 <class 'str'>
>>> b=int(a)
>>> print(b, type(b))-----------123 <class 'int'>
>>> a="12.23" # float String into int Not Possible
>>> print(a,type(a))-----12.23 <class 'str'>
>>> b=int(a)----ValueError: invalid literal for int() with base 10:
'12.23'
>>> a="True"   # bool string into int Not Possible
>>> print(a,type(a))--------True <class 'str'>
>>> b=int(a)-----ValueError: invalid literal for int() with base 10:
'True'
>>> a="2+3.5j"  # complex String  into  int Not Possible
>>> print(a,type(a))------------2+3.5j <class 'str'>
>>> b=int(a)-----------ValueError: invalid literal for int() with base 10:
'2+3.5j'
>>> a="PYTHON" # pure string  into int Not Possible
>>> print(a,type(a))-----------PYTHON <class 'str'>
>>> b=int(a)---------ValueError: invalid literal for int() with base 10:
'PYTHON'
```

```
                                    =============
                                        str()
                                    =============
```

=>This Function is used for converting all types of values into str type.

Syntax: varname=str(int/ float/ bool / complex)

```
--------------------------------------------------------------------------
Examples:
---------------------
>>> a=100
>>> print(a,type(a))-----------100 <class 'int'>
>>> b=str(a)
>>> print(b, type(b))----------100 <class 'str'>
>>> b-------------'100'
>>> a=12.34
>>> print(a,type(a))-----------12.34 <class 'float'>
>>> b=str(a)
>>> print(b, type(b))---------12.34 <class 'str'>
>>> b---------------'12.34'
>>> a=True
>>> print(a,type(a))-----------True <class 'bool'>
>>> b=str(a)
>>> print(b, type(b))-----------True <class 'str'>
>>> b--------------'True'
>>> a=2+3.5j
>>> print(a,type(a))-----------(2+3.5j) <class 'complex'>
>>> b=str(a)
>>> print(b, type(b))------------(2+3.5j) <class 'str'>
>>> b-------------'(2+3.5j)'
=========================X===================================
```



Mutability:

=>An object is said to be mutable iff whose content content can be changed during execution of the program at the same address.

Examples:- list, bytearray...etc

Immutability:

=>An object is said to immutable iff it has to satisfy the following Points

a) Content can't be changed at same address(object does not support item assignment)

b) Content of the object changed and placing modified value at new address.

Examples: int, float, bool

```
                    ============
                        bytes
                    =============
```

=>'bytes' is one of the pre-defined class and treated as a sequential data type.
=>The purpose of this data type is that " To Store Sequence of Posstive Integer values within the range of (0,256). ie. It stores (0,255 only)
=>To convert one type of value into bytes type, we use bytes()

Syntax:-   varname=bytes(list / tuple / set /frozenset/ bytearray)

=>An object of bytes maintains insertion order (Which ever order we insert the data in the same order elements will be displyed)
=>On the object of bytes, we can perform Indexing and Slicing Operations
=>an object Bytes data types belongs to immutable

Examples:

```
>>> l1=[10,20,30,255]
>>> b=bytes(l1)
>>> type(b)
<class 'bytes'>
>>> for x in b:
...     print(x)
...
10
20
```

```
30
255
>>> print(id(b))----------------2225802328416
>>> b[0]=100-----TypeError: 'bytes' object does not support item
assignment

>>> print(b[0])-------------10
>>> print(b[1])----------20
>>> print(b[2])------------30
>>> print(b[3])----------255
>>> print(b[4])--------IndexError: index out of range

>>> for x in b[0:3]:
...     print(x)
                      ...
                      10
                      20
                      30
```

===

bytearray
===

=>'bytearray' is one of the pre-defiend data type and treated as Sequence data type.

=>The purpose of bytearray data type is that "To organize sequece of Possitive Numerical Integer values ranges from (0,256). It Stores the values from 0 to 255(256-1) only ".

=>To store the values in the object of bytearray data type, we don't have any Symbolic Notation but we can convert Other type of values into bytearray type by using bytearray()

=>The object of bytearray belongs to mutable bcoz bytearray allows us to perform updations.

=>On the object of bytearray , we can perform Both Indexing and Slicing Operations.

=>An object of bytearray maintains Insertion Order.

NOTE:- The Functionality of bytearray is exactly similar to bytes data type but the object of bytes belongs to immutable where an object bytearray is mutable.

Examples:

```
>>> lst=[10,20,30,40,-2]
>>> print(lst,type(lst))------------[10, 20, 30, 40, -2] <class 'list'>
>>> b=bytearray(lst)------------ValueError: byte must be in range(0, 256)
>>> lst=[10,20,30,40,256]
>>> b=bytearray(lst)---------ValueError: byte must be in range(0, 256)
>>> lst=[10,20,30,40,255]
>>> b=bytearray(lst)
>>> print(b, id(b),type(b))---bytearray(b'\n\x14\x1e(\xff') 1723585740720

       <class 'bytearray'>

>>> for v in b:
```

```
...     print(v)
                      ...
                      10
                      20
                      30
                      40
                      255
>>> b[0]=100     # updations
>>> for v in b:
...     print(v)
                      ...
                      100
                      20
                      30
                      40
                      255
>>> print(id(b),type(b))----1723585740720 <class 'bytearray'>
>>> print(b[-1])------255
>>> print(b[2])-----30
>>> print(b[::-1])----bytearray(b'\xff(\x1e\x14d')
>>> for v in b[::-1]:
        ...     print(v)
                      ...
                      255
                      40
                      30
                      20
                      100
============================X===============================
```

```
                ========================
                         range
                ========================
```

=>'range' is one pre-defined class and treated as sequence data type.
=>The purpose of range data type is that "To store sequence of Numerical Integer values by maintaining equal Interval of value ".
=>An object of range is immutable bcoz range object does not allow Item assignment.
=>On the object of range , we can perform Indexing and slicing Operations.
=>To cerate an object of range , we use range()
=>range() contains 3 syntaxes. They are

```
----------------------------------------------------------------------------
=>Syntax1:     varname= range(value)
=>This syntax creates an object of range from 0 to value-1

Examples:
--------------------
>>> r=range(6)
>>> print(r,type(r))-----------range(0, 6) <class 'range'>
>>> for v in r:
...     print(v)
                      ...
                      0
                      1
                      2
                      3
```

```
                       4
                       5
>>> for v in range(6):
...     print(v)
                       ...
                       0
                       1
                       2
                       3
                       4
                       5
-----------------------------------------------------------------------------
=>Syntax2:    varname= range(start,stop)
=>This syntax creates an object of range from start value to stop value-1 .
----------------------
=>Examples:
---------------------
>>> r=range(10,16)
>>> print(r,type(r))-----------range(10, 16) <class 'range'>
>>> for v in r:
...     print(v)
                       ...
                       10
                       11
                       12
                       13
                       14
                       15
>>> for v in range(20,26):
        ...     print(v)
                       ...
                       20
                       21
                       22
                       23
                       24
                       25
=>In Syntax1 and Syntax2, the default interval value is 1
-----------------------------------------------------------------------------
Syntax3:      varname=range(start,stop,step)
=>This syntax creates an object of range from start value to stop value-1  by
maintaining specified  step value( Step value  is nothing equal interval of
value)
-----------------------------------------------------------------------------
Examples:
---------------------------
Q1)  Generate 1   2    3     4     5    6     7    8   9   10-------
range(1,11)
>>> for v in range(1,11,1):
...     print(v)
...
1
2
3
4
5
6
```

```
7
8
9
10
---------------------------------------------------
Q2) generate 10    20   30  40 50   60  70   80   90  100----range(10,101,10)
>>> for v in range(10,101,10):
...     print(v)
...
10
20
30
40
50
60
70
80
90
100
-----------------------------------------------------------------------------
Q) Generate 100  105   110   115   120-----range(100,121,5)
>>> for v in range(100,121,5):
...     print(v)
...
100
105
110
115
120
----------------------------------------------------------------------------
Q) Generate -1   -2   -3   -4   -5  -6  -7  -8  -9  -10  ---range(-1,-11,-1)
>>> for v in range(-1,-11,-1):
...     print(v)
...
-1
-2
-3
-4
-5
-6
-7
-8
-9
-10
-----------------------------------------------------------
Q) generate -100   -110    -120   -130 -140  -150--range(-100,-151,-10)
>>> for v in range(-100,-151,-10):
...     print(v)
...
-100
-110
-120
-130
-140
-150
----------------------------------------------------------------------------
Q) generate 10   9    8     7   6   5   4    3   2   1----range(10,0,-1)
```

```
>>> for k in range(10,0,-1):
...     print(k)
...
10
9
8
7
6
5
4
3
2
1
-----------------------------------------------------------------------------
Q) Generate 100  90   80  70  60  50 ----range(100,49,-10)
>>> for k in range(100,49,-10):
...     print(k)
...
100
90
80
70
60
50
-----------------------------------------------------------------------------
Q) Generate  -10  - 9   -8   -7 -6 -5 -4 -3 -2 -1----range(-10,0,1)
>>> for v in range(-10,0,1):
...     print(v)
...
-10
-9
-8
-7
-6
-5
-4
-3
-2
-1
-----------------------------------------------------------------------------
Q) Generate -5   -4   -3   -2  -1   0   1  2   3   4   5---range(-5,6,1)
       >>> for v in range(-5,6,1):
...     print(v)
...
-5
-4
-3
-2
-1
0
1
2
3
4
5
----------------------------------------------------------------------------
Q) generate a multiplication table for number 9
```

```
>>> n=9
>>> for i in range(1,11):
        ...     print(n,"x",i,"=",n*i)
                               ...
                               9 x 1 = 9
                               9 x 2 = 18
                               9 x 3 = 27
                               9 x 4 = 36
                               9 x 5 = 45
                               9 x 6 = 54
                               9 x 7 = 63
                               9 x 8 = 72
                               9 x 9 = 81
                               9 x 10 = 90
==============================X==============================
```

```
Examples----range
-----------------
1   2    3     4     5    6     7    8   9   10-------range(1,11)

10    20   30  40 50   60  70   80   90  100----range(10,101,10)

100  105   110   115   120-----range(100,121,5)
----------------------------------------------------------
-1   -2   -3   -4   -5  -6  -7  -8  -9  -10  ---range(-1,-11,-1)
-100   -110    -120   -130 -140  -150--range(-100,-151,-10)
----------------------------------------------------------
10   9    8     7   6   5   4    3   2   1----range(10,0,-1)

100  90   80  70  60  50 ----range(100,49,-10)
-----------------------------------------------------------
-10  - 9   -8   -7 -6 -5 -4 -3 -2 -1----range(-10,0,1)

-5   -4   -3   -2  -1   0   1  2   3   4   5---range(-5,6,1)
```

List Catagery Data Types (Collection Data Types)

=>List Catagery Data Types are used for storing Multiple Values either of same type or different type or both types with Unique and Duplicate Values in a single variable.

=>List Catagery Data Types are classified into 2 types. They are

a) list
b) tuple

list

Index:

=>Purpose
=>Organization of elements
=>Operations on List
=>Pre-defined Functions in list
=>Inner / Nested List
=>Pre-defined Functions in inner / nested list

Properties of list:

=>'list' is one of the pre-defined class and treated as List catagery data type.

=>The purpose of list data type is that "To store Multiple Values either of same type or different type or both types with Unique and Duplicate Values in a single variable"

=>The elements of list must written within Square Brackets [ ] and elements must separated by comma.

=>An object of list maintains Insertion Order (In which ever order we insert the data in the object of list, in the same order elements will be displayed")

=>On the object of list , we can perform both indexing and slicing Operations.

=>An object of list is mutable

=>We create two types of lists. They are

```
               a) Empty List
               b) Non-empty list
=> An Empty List  is one, whose length=0  (no elements  presents)
               Syntax:-       listobj=[]      (OR)   listobj=list()
=> An Non Empty List  is one, whose length>0 (elements presents)
               Syntax:-    listobj=[val1,val2,...val-n]
```

=>To convert one type elements into list values, we use list(object)

Examples:

```
>>> l=[10,12,-4,25,67]
>>> print(l,type(l))-----------------[10, 12, -4, 25, 67] <class 'list'>
>>> len(l)-------5
>>> l1=[10,"Rossum",11.11,"CWI","NL",2+3j,True]
>>> print(l1,type(l1))--[10, 'Rossum', 11.11, 'CWI', 'NL', (2+3j), True]
<class ,'list'>
>>> len(l1)
7
>>> l2=[]
>>> print(l2,type(l2))---------[] <class 'list'>
>>> len(l2)----------0
>>> l3=list()
>>> print(l3,type(l3))-----------[] <class 'list'>
>>> len(l3)-----------0
>>> l1=[10,"Rossum",11.11,"CWI","NL",2+3j,True]
>>> print(l1,type(l1))---[10, 'Rossum', 11.11, 'CWI', 'NL', (2+3j), True]
<class 'list'>
>>> print(l1[0])----10
>>> print(l1[-1])----True
>>> print(l1[0:4])--------[10, 'Rossum', 11.11, 'CWI']
>>> print(l1[::2])--------[10, 11.11, 'NL', True]
>>> print(l1[::-1])------[True, (2+3j), 'NL', 'CWI', 11.11, 'Rossum', 10]
>>> print(l1,type(l1))----[10, 'Rossum', 11.11, 'CWI', 'NL', (2+3j), True]
<class 'list'>
>>> print(id(l1))--------2261150743872
>>> l1[-1]=False
>>> print(l1,type(l1),id(l1))
        [10, 'Rossum', 11.11, 'CWI', 'NL', (2+3j), False] <class 'list'>
2261150743872
```

VV.IMP

```
>>> a=10
>>> l1=list([a])
>>> print(l1,type(l1))-------[10] <class 'list'>
               (OR)
>>> a=100.2
>>> l1=[a]
>>> print(l1,type(l1))-------------[100.2] <class 'list'>
===============================X===========================================
               =================================================
                       Pre-defined Functions in list
               =================================================
=>In addition to the indexing and slicing Operation on list, we can also
perform Various additional operations by using Pre-defined Functions present
in list.
=>The pre-defined functions in list are
------------------------------------------------------------------------------
1) append():
---------------------
=>This Function is used for adding the values to the list at end of existing
elements of list.
=>Syntax:-     listobj.append(element)
------------------
Examples:
-----------------
>>> l1=[]
>>> print(l1,type(l1))------------[] <class 'list'>
>>> len(l1)----------0
>>> l1.append(10)
>>> print(l1,type(l1))----------[10] <class 'list'>
>>> l1.append("ROSUUM")
>>> print(l1,type(l1))--------[10, 'ROSUUM'] <class 'list'>
>>> l1.append(10.22)
>>> print(l1,type(l1))---------[10, 'ROSUUM', 10.22] <class 'list'>
>>> l2=[10,20,30,40,-45]
>>> l2.append("Hyd")
>>> print(l2,type(l1))------------[10, 20, 30, 40, -45, 'Hyd'] <class 'list'>
------------------------------------------------------------------------------
2)  insert():
-------------------
=>This Function is used for inserting a Value at a perticyulat exiting index
by passing Index and Element.
----------------
=>Syntax: listobj.insert(index,element)
-----------------
Examples:
-------------------
>>> l1=[10,20,30,40,-45]
>>> print(l1)-------------[10, 20, 30, 40, -45]
>>> l1.insert(2,"PYTHON")
>>> print(l1)------------[10, 20, 'PYTHON', 30, 40, -45]
>>> l1.insert(1,"Rossum")
>>> print(l1)-------------[10, 'Rossum', 20, 'PYTHON', 30, 40, -45]
>>> l1.insert(-3,44.44)
>>> print(l1)-----------[10, 'Rossum', 20, 'PYTHON', 44.44, 30, 40, -45]
-----------------------------------------------------------------------------
3) clear():
----------------
```

```
=>This function is used for removing / deleting all the elements of list
object
=>Syntax:-   listobj.clear()

Examples:
----------------------
>>> l1=[10,20,30,40,-45]
>>> print(l1)-------------[10, 20, 30, 40, -45]
>>> len(l1)------------5
>>> l1.clear()
>>> print(l1)-----------[]
>>> len(l1)------------0
-----------------------------------------------------------------------------
4) remove():
---------------------
=>This Function is used removing / deleting  First Occurence of the specified
element
=>If the element is not present in list then we get ValueError
Syntax:-      listobj.remove(element)
Examples:
---------------------
>>> l1=[10,"Python","Java",10,23.45,"PYTHON"]
>>> print(l1)--------[10, 'Python', 'Java', 10, 23.45, 'PYTHON']
>>> l1.remove(10)
>>> print(l1)---['Python', 'Java', 10, 23.45, 'PYTHON']
>>> l1.remove("PYTHON")
>>> print(l1)------['Python', 'Java', 10, 23.45]
>>> l1.remove(100)----------ValueError: list.remove(x): x not in list
----------------------------------------------------------------------------
5) pop(Index)
--------------------
=> This function is used for deleting the element of list based on Valid
Exiting index otherwise we get IndexError.
=>Syntax:-    listobj.pop(index)
-------------------
Examples:
-------------------
>>> l1=[10,"Python","Java",10,23.45,"PYTHON"]
>>> print(l1)-----------[10, 'Python', 'Java', 10, 23.45, 'PYTHON']
>>> l1.pop(3)--------10
>>> print(l1)----------[10, 'Python', 'Java', 23.45, 'PYTHON']
>>> l1.pop(-2)---------23.45
>>> print(l1)-----------[10, 'Python', 'Java', 'PYTHON']
>>> l1.pop(13)----------IndexError: pop index out of range
>>> list().pop(1)-------IndexError: pop from empty list
>>> [].pop(-1)---IndexError: pop from empty list
--------------------------------------------------------------------------
6) pop():
-------------------------
=>This function is used for removing last element of list object (last
indexed element)
=>when we call pop() on empty list object then we get IndexError.
Syntax:-
-------------         listobj.pop()
```

```
Examples:
-------------------
>>> lst=[10,"Python","Rossum",34.56,True]
>>> print(lst)---------------[10, 'Python', 'Rossum', 34.56, True]
>>> lst.pop()----------True
>>> print(lst)------------[10, 'Python', 'Rossum', 34.56]
>>> lst.pop()-----------34.56
>>> print(lst)-----------[10, 'Python', 'Rossum']
>>> lst.pop()----------'Rossum'
>>> print(lst)-----------[10, 'Python']
>>> lst.pop()------------'Python'
>>> print(lst)-----------[10]
>>> lst.pop()------------10
>>> print(lst)-----------[]
>>> lst.pop()------IndexError: pop from empty list
>>> lst=[10,"Python","Rossum",34.56,True]
>>> print(lst)-----------[10, 'Python', 'Rossum', 34.56, True]
>>> lst.insert(3,"Java")
>>> print(lst)-----[10, 'Python', 'Rossum', 'Java', 34.56, True]
>>> lst.pop()-----True
>>> print(lst)-----[10, 'Python', 'Rossum', 'Java', 34.56]
-----------------------------------------------------------------------------
7) copy():
----------------
=>This Function is used copying the content of one list object into another
list object ( implementing shallow copy)
-----------------
Syntax:-   listobj2=listobj1.copy()
-------------------
Examples:
-------------------
>> lst1=[10,"Python","Rossum",34.56]
>>> print(lst1,id(lst1))----[10, 'Python', 'Rossum', 34.56] 2955419270720
>>> lst2=lst1.copy()
>>> print(lst2,id(lst2))----[10, 'Python', 'Rossum', 34.56] 2955419255872
>>> lst1.append(True)
>>> print(lst1,id(lst1))----[10, 'Python', 'Rossum', 34.56, True]
2955419270720
>>> print(lst2,id(lst2))----[10, 'Python', 'Rossum', 34.56] 2955419255872
>>> lst2.insert(2,"Java")
>>> print(lst1,id(lst1))---[10, 'Python', 'Rossum', 34.56, True]
2955419270720
>>> print(lst2,id(lst2))---[10, 'Python', 'Java', 'Rossum', 34.56]
2955419255872
-------------------------
Deep Copy:
--------------------
>> lst1=[10,"Python","Rossum",34.56]
>>> lst1=[10,"Python","Rossum",34.56]
>>> lst2=lst1    # Implementing Deep Copy Process
>>> print(lst1,id(lst1))----------[10, 'Python', 'Rossum', 34.56]
2955419266624
>>> print(lst2,id(lst2))------[10, 'Python', 'Rossum', 34.56] 2955419266624
>>> lst1.append(True)
>>> print(lst1,id(lst1))----[10, 'Python', 'Rossum', 34.56, True]
2955419266624
```

```
>>> print(lst2,id(lst2))---[10, 'Python', 'Rossum', 34.56, True]
2955419266624
>>> lst2.insert(2,"DS")
>>> print(lst1,id(lst1))--[10, 'Python','DS','Rossum',34.56,True]
2955419266624
>>> print(lst2,id(lst2))--[10,'Python','DS','Rossum', 34.56, True]
2955419266624
```

Slicing Based Copy:

=>The this copy process is also Shallow Copy implementation only.

Examples:

```
>>> lst1=[10,"Python","Rossum",34.56]
>>> print(lst1,id(lst1))----[10, 'Python', 'Rossum', 34.56] 2955419255872
>>> lst2=lst1[::]  # slice based copy
>>> print(lst2,id(lst2))----[10, 'Python', 'Rossum', 34.56] 2955419270720
>>> lst1.remove(34.56)
>>> print(lst1,id(lst1))----[10, 'Python', 'Rossum'] 2955419255872
>>> print(lst2,id(lst2))----[10, 'Python', 'Rossum', 34.56] 2955419270720
>>> lst3=lst1[0:3]   # slice based copy
>>> print(lst3,id(lst3))---[10, 'Python', 'Rossum'] 2955419266624
>>> lst4=lst1[::-1]  # slice based copy
>>> print(lst4,id(lst4))---['Rossum', 'Python', 10] 2955419517312
```

8)count():

=>This function is used for counting / finding number of occurences of the specified element .

=>If the specified element does not exists in list object then we get 0.

Syntax:-   listobj.count(element)

Examples:

```
>>> lst=[10,20,"python",10,"python",10,30,20,10]
>>> lst.count(10)--------4
>>> lst.count("python")-----------2
>>> lst.count(20)----------2
>>> lst.count(30)---------1
>>> lst.count(300)--------0
```

9) index()

=>This function is used for obtaining an index of the First occurence of specified eleement

=>If element does not exists in list object then we get ValueError.

Syntax:- listobj.index(element)

Examples:

```
>>> lst=[10,20,"python",10,"python",10,30,20,10]
>>> print(lst.index(10))---------0
>>> print(lst.index(20))--------1
>>> print(lst.index("python"))-----2
>>> print(lst.index("python3.10"))-------ValueError: 'python3.10' is not in
list
---------------------------------------------------------------------------
10)reverse():
----------------------
=>This function is used for obtaining reverse of elements of list object
=>Syntax:-    listobj.reverse()
-----------------------------
Examples:
-----------------
>>> lst1=[10,"Python","Rossum",34.56]
>>> print(lst1)----------------[10, 'Python', 'Rossum', 34.56]
>>> print(lst1.reverse())-------------None
>>> print(lst1)-----[34.56, 'Rossum', 'Python', 10]
>>> lst1=[10,"Python","Rossum",34.56]
>>> print(lst1)-----------[10, 'Python', 'Rossum', 34.56]
>>> lst1.reverse()
>>> print(lst1)-------------[34.56, 'Rossum', 'Python', 10]
>>> lst2=[10,20,30,-23,45,2,67,34]
>>> print(lst2)--------[10, 20, 30, -23, 45, 2, 67, 34]
>>> lst2.reverse()
>>> print(lst2)-----[34, 67, 2, 45, -23, 30, 20, 10]
-----------------------------------------------------------------------------
11) sort():
----------------
=>This function is used for sorting the given homogeneous data of list object
either Ascending Order or in decending order.

=>Syntax:       listobj.sort(reverse=False / True )
------------------
=>If reverse=False then sort() sorts the data in Ascending order
=>If reverse=True then sort() sorts the data in Decending order
=>If we don't write reverse=False then ity similar to sort() and sorts the
data in Ascending order
------------------
Examples:
-----------------
>>> lst2=[10,20,30,-23,45,2,67,34]
>>> print(lst2)---------[10, 20, 30, -23, 45, 2, 67, 34]
>>> lst2.sort()
>>> print(lst2)----------[-23, 2, 10, 20, 30, 34, 45, 67]
>>> lst2.reverse()
>>> print(lst2)------[67, 45, 34, 30, 20, 10, 2, -23]

>>> lst3=["apple","sberry","guava","mango","abc"]
>>> print(lst3)---------['apple', 'sberry', 'guava', 'mango', 'abc']
>>> lst3.sort()
>>> print(lst3)----------['abc', 'apple', 'guava', 'mango', 'sberry']
>>> lst3.reverse()
>>> print(lst3)-------------['sberry', 'mango', 'guava', 'apple', 'abc']
----------------------------------------
>>> lst2=[10,20,30,-23,45,2,67,34]
>>> print(lst2)----------------[10, 20, 30, -23, 45, 2, 67, 34]
>>> lst2.sort(reverse=True)
>>> print(lst2)-------------[67, 45, 34, 30, 20, 10, 2, -23]
>>> lst2=[10,20,30,-23,45,2,67,34]
>>> print(lst2)--------[10, 20, 30, -23, 45, 2, 67, 34]
>>> lst2.sort(reverse=False)
>>> print(lst2)------------[-23, 2, 10, 20, 30, 34, 45, 67]
-----------------------------------------------------------------------------
12) extend():
-------------------
=>This function is used for extending functionality of source list object
with destination list object
=>Syntax:     sourcelistobject.extend(destination list obj)

Examples:
-----------------
>>> lst1=[10,20,30]
>>> lst2=["Java","python","DS","AI"]
>>> lst1.extend(lst2)
>>> print(lst1)------------[10, 20, 30, 'Java', 'python', 'DS', 'AI']
------------------------------------------------------
>>> lst1=[10,20,30]
>>> lst2=["Java","python","DS","AI"]
>>> lst3=["Oracle","MYSQL"]
>>> lst4=["Tomcat Ser","WebLogic","Web Sphere"]
>>> lst1.extend(lst2,lst3,lst4)----TypeError: list.extend() takes exactly one

        argument (3 given)
#we can achieve extend() task with + operator

>>> lst1=lst1+lst2+lst3+lst4
>>> print(lst1)---- [10, 20, 30, 'Java', 'python', 'DS', 'AI', 'Oracle',
'MYSQL',
                                  'Tomcat Ser', 'WebLogic', 'Web Sphere']
===========================X=========================================
                       Types of Copy Mechanisms
               ==========================================
=>Copy Process is nothing but copying  the content of one object into another
object.
=>WE have two types of Copy Process. They are
               a) Shallow Copy
               b) Deep Copy
----------------------------
a) Shallow Copy:
----------------------------
=>In shallow Copy
        i) Initial Content of both the objects are same
        ii) Both the objects contains different address
        iii) The Modifications on the objects are Independent.
                                             (Modifications are not
recfected)
=>To implement Shallow Copy, we use copy()

=>Syntax:-      objname1=objname2.copy()
-----------------------------------------------------------------------------
b) Deep Copy:
-----------------------------------------------------------------------------
=>In Deep Copy
        i) Initial Content of both the objects are same
        ii) Both the objects contains Same Address
        iii) The Modifications on the objects are dependent.
                 (Modifications are  recflected to each other)
```

```
=>To implement  Deep Copy, we use Assignment Operator
=>Syntax:-      objname1=objname2
```

Slicing Based Copy:

=>The this copy process is also Shallow Copy implementation only.

Examples:

```
>>> lst1=[10,"Python","Rossum",34.56]
>>> print(lst1,id(lst1))----[10, 'Python', 'Rossum', 34.56] 2955419255872
>>> lst2=lst1[::]  # slice based copy
>>> print(lst2,id(lst2))----[10, 'Python', 'Rossum', 34.56] 2955419270720
>>> lst1.remove(34.56)
>>> print(lst1,id(lst1))----[10, 'Python', 'Rossum'] 2955419255872
>>> print(lst2,id(lst2))----[10, 'Python', 'Rossum', 34.56] 2955419270720
>>> lst3=lst1[0:3]   # slice based copy
>>> print(lst3,id(lst3))---[10, 'Python', 'Rossum'] 2955419266624
>>> lst4=lst1[::-1]  # slice based copy
>>> print(lst4,id(lst4))---['Rossum', 'Python', 10] 2955419517312
```

X

Inner or Nested List

=>The Process of defining one list inside of another list is called Inner / nested list.

=>Syntax:

```
listobj=[ val1,val2....[ val11,val12,...] , [val22,val23....] ....val-n ]
```

=>[val11,val12,...] is called one inner list
=>[val22,val23....] is called another inner list
=>[val1,val2.... ....val-n] is called outer list
=>On the list and inner list we can apply all operation regarding Indexing, Slicing and all pre-defined functions.

Examples:

```
My Requirement -->To store
               stno----------10
               name -----Mahesh
               Internal Marks of three subs---18,19,17
               External Marks of three subs---73 71,65
               College name---OUCET
```

```
   stlist=[10,"Mahesh",18,19,17,73,71,65,"OUCET"]
                       (OR)
stlst1=[ 10,"Mahesh", [18,19,17],[73,71,65], "OUCET" ]
```

Examples:

```
>>> l1=[10,"Rossum",[16,19,15],[78,65,79],"NLU"]
>>> print(l1,type(l1))
[10, 'Rossum', [16, 19, 15], [78, 65, 79], 'NLU'] <class 'list'>
>>> print(l1[0])
10
```

```
>>> print(l1[1])
Rossum
>>> print(l1[2])
[16, 19, 15]
>>> print(l1[2][1])
19
>>> print(l1[2][-1])
15
>>> print(l1[3])
[78, 65, 79]
>>> print(l1[3][-3])
78
>>> print(l1[3][0])
78
>>> print(l1[4])
NLU
>>> print(l1[3][0:3]
... )
[78, 65, 79]
>>> print(l1[3][::-1])
[79, 65, 78]
>>> l1[3].append(80)
>>> print(l1[3])
[78, 65, 79, 80]
>>> l1[-3].insert(-2,16)
>>> l1[-3]
[16, 16, 19, 15]
>>> l1[-3].sort()
>>> l1[-3]
[15, 16, 16, 19]
>>> l1[-2].sort(reverse=True)
>>> l1[-2]-----------------[80, 79, 78, 65]
```

Pre-defined Functions in tuple

```
=>Tuple object contains two Functions. They are
               a) count()
               b) index()


=>Tuple does not contain the following Functions
-----------------------------------------------------------------------------
append()     insert()    clear()  copy()   pop()
pop(index)  remove()    sort()   reverse()  extend()
===============================X===============================
```

tuple (Collection type)

=>'tuple' is one of the pre-defined class and terated list type data type.
=>The purpose of tuple data type is that "To store Multiple Values either of same type or different type or both types with Unique and Duplicate Values in a single variable"
=>The elements of tuple must written within braces ( ) and elements must separated by comma.

=>An object of tuple maintains Insertion Order (In which ever order we insert the data in the object of tuple, in the same order elements will be displayed")

=>On the object of tuple , we can perform both indexing and slicing Operations.

=>An object of tuple is immutable

=>We create two types of tuples. They are

```
               a) Empty tuple
               b) Non-empty tuple
=> An Empty tuple  is one, whose length=0  (no elements  presents)
               Syntax:-       tupleobj=()    (OR)   tupleobj=tuple()
=> An Non Empty tuple is one, whose length>0 (elements presents)
               Syntax:-    tupleobj=(val1,val2,...val-n)
```

=>To convert one type elements into tuple values, we use tuple(object)

Note:- The Functionality of tuple is exactly similar to List but an object of list belongs to mutable and an object of tuple belongs to immutable.

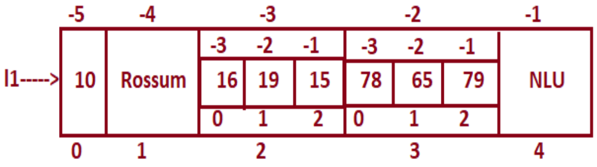
Examples:

```
-----------------
>>> t1=(10,20,-3,45,123,67,20)
>>> print(t1,type(t1))---------(10, 20, -3, 45, 123, 67, 20) <class 'tuple'>
>>> t2=(10,"Rossum",45.67,"Python",True)
>>> print(t2,type(t2))------(10, 'Rossum', 45.67, 'Python', True) <class
'tuple'>
>>> t3=()
>>> print(t3,type(t3), len(t3))--------() <class 'tuple'> 0
>>> t4=tuple()
>>> print(t4,type(t4), len(t4))---------() <class 'tuple'> 0
>>> t2=(10,"Rossum",45.67,"Python",True)
>>> print(t2[0])----10
>>> print(t2[-1])----True
>>> print(t2[0:3])----(10, 'Rossum', 45.67)
>>> print(t2[::2])----(10, 45.67, True)
>>> t2=(10,"Rossum",45.67,"Python",True)
>>> print(t2,type(t2), id(t2))--(10, 'Rossum', 45.67, 'Python', True) <class
'tuple'>
                       2399350751152
>>> t2[2]=55.66 ---TypeError: 'tuple' object does not support item assignment
==========================================================
>>> t1=(12,3,-4,45,23,78,4,1,12)
>>> print(t1,type(t1))---(12, 3, -4, 45, 23, 78, 4, 1, 12) <class 'tuple'>
>>> t1.sort()--------AttributeError: 'tuple' object has no attribute 'sort'
>>> l1=list(t1)
>>> print(l1,type(l1))---[12, 3, -4, 45, 23, 78, 4, 1, 12] <class 'list'>
>>> print(l1,type(l1),id(l1))--[12, 3, -4, 45, 23, 78, 4, 1, 12] <class
'list'>

2399351145088
>>> l1.sort()
>>> print(l1,type(l1),id(l1))--[-4, 1, 3, 4, 12, 12, 23, 45, 78] <class
'list'>

2399351145088
>>> t1=tuple(l1)
>>> print(t1,type(t1))---(-4, 1, 3, 4, 12, 12, 23, 45, 78) <class 'tuple'>
-----------------------------------------------------------------------------
```

```
>>> x=10,20,"KVR","OUCET",True
>>> print(x, type(x))---(10, 20, 'KVR', 'OUCET', True) <class 'tuple'>
--------------------------------------------------------------------------
>>> t1=(10,"Rossum",(12,16,11),"NLU")
>>> print(t1,type(t1))
(10, 'Rossum', (12, 16, 11), 'NLU') <class 'tuple'>
>>> print(t1[2])
(12, 16, 11)
>>> t1=(10,"Rossum",[12,16,11],"NLU")
>>> print(t1,type(t1))
(10, 'Rossum', [12, 16, 11], 'NLU') <class 'tuple'>
>>> print(t1[2],type(t1[2]))
[12, 16, 11] <class 'list'>
>>> t1[2].sort()
>>> print(t1,type(t1))
(10, 'Rossum', [11, 12, 16], 'NLU') <class 'tuple'>
>>> l1=[10,"KVR",(10,20,12),"OUCET"]
>>> print(l1,type(l1))
[10, 'KVR', (10, 20, 12), 'OUCET'] <class 'list'>
```

Inner / nested list memory Management

l1=[10,"Rossum",[16,19,15],[78,65,79],"NLU"]



Set Category Data Types (Collection Data Types)

>Set Category Data Types are used for storing Multiple Values either of same type or different type or both types with Unique Values in a single variable.

=>Set Category Data Types are 2 types. They are

i) set (mutable and immutable)
ii) frozenset (immutable)

set

=>'set' of one of the pre-defined class treated as Set category data type.
=>The purpose of set data type is that "To Store Multiple Values either of same type or different type or both types with Unique Values in a single variable".
=>The elments of set must organized with curly braces { } and elements bmust be separated by comma.
=>The elements of set never maintains insertion Order bcoz it displays its elements in any of the possibilities.
=>On the object of set, we can't perform indexing and Slicing Operations bcoz it can't maintain insertion order.
=>An object of set belongs to both mutable (in the case of add()) and immutable in the case item assignment (set' object does not support item assignment).
=>To convert one type value into set type values , we use set().
=>We have two types of set objects.

a) empty set
b) non-empty set

a) empty set:

=>An empty set is one, whose length is 0

```
Syntax:    setobj=set()
```

b) non-empty set:

=>An non-empty set is one, whose length is >0

```
Syntax:    setobj={val1,val2....val-n}
```

Examples:

```
>>> s1={10,20,10,20,30,123,-56}
>>> print(s1,type(s1))
{20, -56, 10, 123, 30} <class 'set'>
>>> s1={10,"KVR",33.33,"OUCET","HYD",True}
>>> print(s1,type(s1))
{'KVR', 33.33, True, 'OUCET', 10, 'HYD'} <class 'set'>
>>> s1[0]=100---->TypeError: 'set' object does not support item assignment
>>> print(s1,type(s1),id(s1))
{'KVR', 33.33, True, 'OUCET', 10, 'HYD'} <class 'set'> 1844977298208
>>> s1.add("PYTHON")
>>> print(s1,type(s1),id(s1))
{'KVR', 33.33, True, 'PYTHON', 'OUCET', 10, 'HYD'} <class 'set'>
1844977298208
-------------------------------------------------------------
>>> s1=set()
>>> print(s1,type(s1),id(s1))
set() <class 'set'> 1844977297088
>>> len(s1)
0
>>> s1.add(10)
>>> s1.add("RS")
>>> print(s1,type(s1),id(s1))
{'RS', 10} <class 'set'> 1844977297088
==============================X==========================
               =====================================
                       Pre-defined Functions in set
               =====================================
1) add():
-----------------
=>This function is used for adding an element to the set object

=>Syntax:-      setobj.add(element)

Examples:
------------------
>>> s1={10,"Rossum"}
>>> print(s1,type(s1),id(s1))
{'Rossum', 10} <class 'set'> 1844977298208
>>> s1.add("PYTHON")
>>> s1.add(11.11)
>>> print(s1,type(s1),id(s1))
{'Rossum', 10, 11.11, 'PYTHON'} <class 'set'> 1844977298208
-----------------------------------------------------------------------------
------------------------
2) remove():
-------------------
=>This function is used for removing the specified element from set object.
=>If the specified element does not exists in set object we get KeyError.
=>Syntax:-    setobj.remove(element)

Examples:
---------------------
>>> s1={'Rossum', 10, 11.11, 'PYTHON'}
>>> print(s1)----{'Rossum', 10, 11.11, 'PYTHON'}
>>> s1.remove(10)
>>> print(s1)--------------{'Rossum', 11.11, 'PYTHON'}
>>> s1.remove("Rossum")
>>> print(s1)-----{11.11, 'PYTHON'}
>>> s1.remove(101)-----------KeyError: 101
-----------------------------------------------------------------------------
--------------------------------
3) discard()
-------------------
=>This function is used for removing the  specified element from set object.
=>If the specified element does not exists in set object we nerver get any
error.
=>Syntax:-   setobj.discard(element)
------------------
Examples:
-----------------
>>> s1={'Rossum', 10, 11.11, 'PYTHON'}
>>> print(s1)-----{'Rossum', 10, 11.11, 'PYTHON'}
>>> s1.discard(10)
>>> print(s1)---------{'Rossum', 11.11, 'PYTHON'}
>>> s1.discard(100)  # here 100 does not exist and no error
>>> print(s1)-------{'Rossum', 11.11, 'PYTHON'}
-----------------------------------------------------------------------------
4)  pop()
-----------------
=>This function is used for removing an arbitrary element from set object.
=>Syntax:      setobj.pop()
Examples:
----------------------
>>> s1={'Rossum', 10, 11.11, 'PYTHON'}
>>> print(s1)-----------{'Rossum', 10, 11.11, 'PYTHON'}
>>> s1.pop()----------'Rossum'
>>> print(s1)----{10, 11.11, 'PYTHON'}
>>> s1.pop()----10
>>> print(s1)----{11.11, 'PYTHON'}
>>> s1.pop()------11.11
>>> print(s1)----{'PYTHON'}
>>> s1.pop()-------'PYTHON'
>>> s1={10,20,30,40,50,60,70,-123,3456}
>>> s1.pop()--------3456
>>> s1={10,20,30,40,50,60,70,-123,3456}
>>> print(s1)------{3456, -123, 70, 40, 10, 50, 20, 60, 30}
>>> s1.pop()------3456
>>> print(s1)----{-123, 70, 40, 10, 50, 20, 60, 30}
>>> s1.pop()--------123
>>> s1.pop()-----70
>>> s1.pop()-------40
>>> s1.pop()--------10
>>> s1.pop()--------50
>>> s1={"apple","Mango","kiwi","abc",23.45,67,2+3j}
>>> s1.pop()------'apple'
>>> s1.pop()-------'kiwi'
>>> print(s1)-------{67, 'abc', 23.45, (2+3j), 'Mango'}
>>> s1.pop()--------67
>>> s1.pop()---------'abc'
-----------------------------------
>>> set().pop()----------KeyError: 'pop from an empty set'
-----------------------------------------------------------------------------
   5) isdisjoint():
   ---------------------
=>Syntax:-      setobj1.isdisjoint(setobj2)
=>This Function returns True provided  setob1 and setobj2 does contains
common elements
=>This Function returns False provided  setob1 and setobj2  contains at least
one common element.
----------------
Examples:
------------------
>>> s1={10,20,30,40}
>>> s2={15,25,35,10}
>>> s3={12,24,36,48}
>>> s1.isdisjoint(s2)---------False
>>> s1.isdisjoint(s3)---------True
>>> s1.isdisjoint(s1)------False
>>> s1.isdisjoint(set())----True
>>> set().isdisjoint(set())---True
```

6) issuperset()

Syntax:- setob1j.issuperset(setobj2)

=>This Function returns True provided all the elemenets of setobj2 must present in setobj1. Otherwise we get False.

Examples:

```
>>> s1={10,20,30,40}
>>> s2={15,25,35,10}
>>> s3={12,24,36,48}
>>> s1.issuperset(s2)
False
>>> s1.issuperset(s3)
False
>>> s4={10,20}
>>> s1.issuperset(s4)
True
>>> s1.issuperset(s1)
True
>>> s1.issuperset(set())
True
>>> set().issuperset(set())
True
>>> set().issubset(set())
True
>>> {10,20}.issuperset({20,10})
True
>>> {10,20,25}.issuperset({20,10})
True
>>> {10,20}.issuperset({20,10,"pyt"})
False
```

7) issubset()

Syntax:- setobj1.issubset(setobj2)

=>This Function returns True provided all the elements of setobj1 are present in setobj2. otherwise we get False

Examples:

```
>>> s1={10,20,30,40}
>>> s2={10,20}
>>> s3={15,20}
>>> s2.issubset(s1)--------True
>>> s3.issubset(s1)-------False
>>> set().issubset(set())----True
```

8) Union()

=>Syntax:- setobj3=setobj1.union(setobj2)

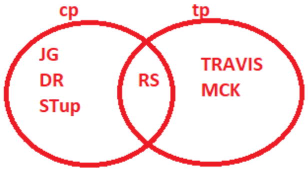
=>This takes all the elements of setobj1 and setobj2 , combine them and place them in setobj3 uniquely.

Examples:

```
>>> s1={"RS","JG","DR","STup"}
>>> s2={"TRAVIS","MCK","RS"}
>>> print(s1)----------------{'RS', 'JG', 'DR', 'STup'}
>>> print(s2)------------{'RS', 'TRAVIS', 'MCK'}
>>> allcptp=s1.union(s2)
>>> print(allcptp)------------{'RS', 'DR', 'STup', 'JG', 'TRAVIS', 'MCK'}
------------------------------------------------------------------------------
9)difference()
------------------------
Syntax:-    setobj3=setobj1.difference(setobj2)
=>This function removes the common elements from setobj1 and setobj2 and
takes remaining elements from setobj1 and place them in setobj3.
-------------------
Examples:
------------------
>>> s1={"RS","JG","DR","STup"}
>>> s2={"TRAVIS","MCK","RS"}
>>> print(s1)----------{'RS', 'DR', 'STup', 'JG'}
>>> print(s2)--------{'RS', 'TRAVIS', 'MCK'}
>>> onlycp=s1-s2
>>> print(onlycp)---------{'JG', 'DR', 'STup'}
>>> onlytp=s2-s1
>>> print(onlytp)--------{'TRAVIS', 'MCK'}
>>> onlycp=s1.difference(s2)
>>> print(onlycp)--------{'JG', 'DR', 'STup'}
>>> onlytp=s2.difference(s1)
>>> print(onlytp)-----{'TRAVIS', 'MCK'}
------------------------------------------------------------------------------
10 intersection():
-----------------------------------------------------------------------------
Syntax:-
---------------
               setobj3=setobj1.intersection(setobj2)
=>This obtains common elements from setobj1 and setobj2 and place tthem
setobj3.
-------------------
Examples:
-------------------
>>> s1={"RS","JG","DR","STup"}
>>> s2={"TRAVIS","MCK","RS"}
>>> s3=s1.intersection(s2)
>>> print(s3)--------{'RS'}
>>> s3=s2.intersection(s1)
>>> print(s3)----{'RS'}
------------------------------------------------------------------------------
11) symmetric_difference():
--------------------------------------------------------
Syntax:-       setobj3=setobj1.symmetric_difference(setobj2)
-------------
=>This function removes common elements from setobj1 and setobj2 and takes
remaining elements from both setobj1 and setobj2 and place them in setobj3.
--------------------
Examples:
--------------------
>>> s1={"RS","JG","DR","STup"}
>>> s2={"TRAVIS","MCK","RS"}
>>> print(s1)---------------{'RS', 'DR', 'STup', 'JG'}
>>> print(s2)------------{'RS', 'TRAVIS', 'MCK'}
>>> excptp=s1.symmetric_difference(s2)
>>> print(excptp)--------{'DR', 'TRAVIS', 'STup', 'JG', 'MCK'}
-----------------------------------------------------------------------------
Special Cases:
-------------------------------
>>> s1={"RS","JG","DR","STup"}
>>> s2={"TRAVIS","MCK","RS"}
>>> s3=s1.union(s2)
>>> print(s3)----------{'RS', 'DR', 'STup', 'JG', 'TRAVIS', 'MCK'}
>>> s4=s1|s2   # Bitwise OR ( | )
>>> print(s4)-----------{'RS', 'DR', 'STup', 'JG', 'TRAVIS', 'MCK'}
>>> s1={"RS","JG","DR","STup"}
>>> s2={"TRAVIS","MCK","RS"}
>>> s3=s1.intersection(s2)
>>> print(s3)----------{'RS'}
>>> s4=s1&s2   # Bitwise AND ( & )
>>> print(s4)----------{'RS'}
>>> s1={"RS","JG","DR","STup"}
>>> s2={"TRAVIS","MCK","RS"}
>>> s3=s1.symmetric_difference(s2)
>>> print(s3)-------{'DR', 'TRAVIS', 'STup', 'JG', 'MCK'}
>>> s4=s1^s2  # Bitwise   XOR (^)
>>> print(s4)---------{'DR', 'TRAVIS', 'STup', 'JG', 'MCK'}
>>> s1={"RS","JG","DR","STup"}
>>> s2={"TRAVIS","MCK","RS"}
>>> s3=s1.difference(s2)
>>> print(s3)----{'JG', 'DR', 'STup'}
>>> s4=s1-s2
>>> print(s4)----------{'JG', 'DR', 'STup'}
-----------------------------------------------------------------------------
12) update():
-----------------------
Syntax:-     setobj1.update(setobj2)
=>This Function updates / adds the elements of setobj2 to setobj1.
Examples:
------------------
>>> s1={"C","CPP"}
>>> s2={"PYTHON","DS"}
>>> s1.update(s2)
>>> print(s1)
{'C', 'CPP', 'PYTHON', 'DS'}
>>> print(s2)
{'PYTHON', 'DS'}
================================X==========================
```

Use Case : cp={"RS","JG","DR","STup"}
tp={"TRAVIS","MCK","RS"}

Q1) Find all the payer names who is all the games.
Q2) Find all the payers who are plying only "cp "
Q3) Find all the payers who are plying only "tp"
Q4) Find all the payers who are plyaing both Games
Q5) Find all the players who are plyaing exclusively cp and tp



==

frozenset

==

=>'frozenset' of one of the pre-defined class treated as Set category data type.

=>The purpose of frozenset data type is that "To Store Multiple Values either of same type or different type or both types with Unique Values in a single variable".

=>The elements of frozenset organized within curly braces { } after converting from tuple, list,set ..etc by using frozenset() and elements separated by comma.

=>The elements of frozenset never maintains insertion Order bcoz it displays its elements in any of the possibilities.

=>On the object of frozenset, we can't perform indexing and Slicing Operations bcoz it can't maintain insertion order.

=>An object of frozenset belongs to immutable (never allows add() ,item assignment)

=>To convert one type value into frozenset type values , we use frozenset().

=>We have two types of frozenset objects.

a) empty frozenset
b) non-empty frozenset

----

a) empty frozenset:

=>An empty frozenset is one, whose length is 0

```
               Syntax:    frozensetobj=frozenset()
```

b) non-empty frozenset:

```
=>An non-empty frozenset is one, whose length is >0
        Syntax:    frozensetobj=frozenset( {val1,val2....val-n} )
        Syntax:    frozensetobj=frozenset( [val1,val2....val-n] )
        Syntax:    frozensetobj=frozenset( (val1,val2....val-n) )......etc
-----------------------------------------------------------------------------
```

Note:- The functionality of frozenset is exactly similar to set but an object set belongs to both mutable (add()) and immutable (item assignment) where an object frozenset is immutable (not possible to add() and item assignment)

```
-----------------------------------------------------------------------------
Examples:
-------------------------
>>> s1={10,20,30,40,30}
>>> print(s1,type(s1))-----------{40, 10, 20, 30} <class 'set'>
>>> fs=frozenset(s1)
>>> print(fs,type(fs))----frozenset({40, 10, 20, 30}) <class 'frozenset'>
>>> tp=(10,"RS","PYTHON")
>>> fs=frozenset(tp)
>>> print(fs,type(fs))-----frozenset({'RS', 10, 'PYTHON'}) <class
'frozenset'>
>>> lst=[10,12.34,"Python","Java",2+3j]
>>> fs=frozenset(lst)
>>> print(fs,type(fs))---frozenset({'Python', 10, (2+3j), 12.34, 'Java'})
<class
               'frozenset'>
>>> print(fs[0])----TypeError: 'frozenset' object is not subscriptable
>>> print(fs[0:3])---TypeError: 'frozenset' object is not subscriptable
>>> fs[0]="Data Sci"---TypeError: 'frozenset' object does not support item

        assignment
>>> fs.add("Data Sci")---AttributeError: 'frozenset' object has no attribute
'add'
>>> fs=frozenset()
>>> print(fs,type(fs))----frozenset() <class 'frozenset'>
>>> len(fs)---------0
>>> fs=frozenset([10,20,20,30,30,10])
>>> print(fs,type(fs))----frozenset({10, 20, 30}) <class 'frozenset'>
>>> len(fs)----------3
============================X=================================
```

Pre-defined Functions in Frozenset

```
-------------------------------------------------------------------------
isdisjoint(), issuperset()  issubset()
union()  intersection()   differnce()   symmetric_difference()
-----------------------------------------------------------------------------
```

Pre-defined Functions does not contain in Frozenset

```
-----------------------------------------------------------------------------
add() remove() discard()   pop()    update()
----------------------------------------------------------------------------
=============================================================================
```

Dict Category Data Type(Collection data type)

```
               ==============================================
```

=>'dict' is one of the pre-defined class and treated as Dict Category Data Type

=>The purpose of dict data type is that " To Organize / store the data in the form of (Key,Value)

=>In (Key,Value), The values of Key represents Unique and values of Value may or may not be unique.

```
=>In organize / store the data in the object of dict, those (Key,Value) must
     written with curly braces { }
=>An object of dict maintains Insertion Order .
=>On the object of dict, we can't perform Indexing and slcing Operation bcoz
values of Key itself acts index.
=>We have two types of dict objects. they are
               a) Empty Dict
               b) Non-Empty Dict
---------------------------
a) Empty Dict:
--------------------------
=>An empty dict does not contain any elements and whose length is 0
=>Syntax:       dictobj={}
                         (or)
                         dictobj=dict()
=>Syntax for adding (Key,Value) to dict object
                       dictobj[Key1]=Value1
                       dictobj[Key2]=Value2
                       --------------------------------
                       dictobj[Key-n]=Value-n
Examples:
------------------
>>> d1={}
>>> print(d1,type(d1),id(d1))------{} <class 'dict'> 2186120856832
>>> d1[10]="RS"
>>> d1[20]="DR"
>>> d1[30]="TR"
>>> d1[40]="MCK"
>>> print(d1,type(d1),id(d1))
        {10: 'RS', 20: 'DR', 30: 'TR', 40: 'MCK'} <class 'dict'> 2186120856832
                       (OR)
>>> d1=dict()
>>> print(d1,type(d1),id(d1))----{} <class 'dict'> 2186121117760
>>> d1[10]="RS"
>>> d1[20]="DR"
>>> d1[30]="TR"
>>> d1[40]="MCK"
>>> print(d1,type(d1),id(d1))
        {10: 'RS', 20: 'DR', 30: 'TR', 40: 'MCK'} <class 'dict'> 2186121117760
-----------------------------------------------------------------------------
b) Non-Empty Dict:
-------------------------------
=>An non-empty dict  contains any elements and whose length is >0
Syntax:
------------
        dictobj={Key1:Value1,Key2:Value2.......Key-n:Value-n}
-------------------
Examples:
-------------------
>>> d1={10:"Rossum",20:"Ritche",30:"Gosling",40:"Travis"}
>>> print(d1,type(d1))
{10: 'Rossum', 20: 'Ritche', 30: 'Gosling', 40: 'Travis'} <class 'dict'>
>>> d1[10]="MCKinney"
>>> print(d1,type(d1))
{10: 'MCKinney', 20: 'Ritche', 30: 'Gosling', 40: 'Travis'} <class 'dict'>
>>> len(d1)---4
>>> d1={10:"Rossum",20:"Ritche",30:"Gosling",40:"Travis"}
>>> print(d1,id(d1))
{10: 'Rossum', 20: 'Ritche', 30: 'Gosling', 40: 'Travis'} 2186120856832
>>> d1[50]="Tim"
>>> print(d1,id(d1))
{10: 'Rossum', 20: 'Ritche', 30: 'Gosling', 40: 'Travis', 50: 'Tim'}
2186120856832
-----------------------------------------------------------------------------
>>> d1=dict()
>>> d1["Apple"]=25.67
>>> d1["Kiwi"]=30
>>> d1["Sberry"]=100.34
>>> d1["Mango"]=80
>>> print(d1)
{'Apple': 25.67, 'Kiwi': 30, 'Sberry': 100.34, 'Mango': 80}
--------------------------------------------------------------------------
>>> d1={}
>>> d1[10]="Praveen"
>>> d1["RS"]=20
>>> print(d1)
{10: 'Praveen', 'RS': 20}
---------------------------------------------------------------------------
Special Cases:
--------------------------------------
>>> d1={10:[100,200,300,400],20:(500,600,700)}
>>> print(d1)---{10: [100, 200, 300, 400], 20: (500, 600, 700)}
>>> for k,v in d1.items():
...     print(k,"--->",v)
                       ...
                       10 ---> [100, 200, 300, 400]
                       20 ---> (500, 600, 700)
============================X=================================
               =============================================
                       pre-defined functions in dict
               =============================================
1) clear():
-----------------
=>This is used for removing all the entires of dict objct.
=>Syntax:-     dictobj.clear()
=>Examples:
-----------------------
>>> d1={'Apple': 25.67, 'Kiwi': 30, 'Sberry': 100.34, 'Mango': 80}
>>> print(d1)
{'Apple': 25.67, 'Kiwi': 30, 'Sberry': 100.34, 'Mango': 80}
>>> d1.clear()
>>> print(d1)-------{}
-----------------------------------------------------------------------------
2) pop()
-----------------
=>This function is used for removing (Key,Value) from dict object by passing
Value of Key.
=>If the Value of Key does not exists in dict object then we get KeyError.
--------------------
=>Syntax:-             dictobj.pop(key)
--------------------
Examples:
--------------------
```

```
>>> d1={'Apple': 25.67, 'Kiwi': 30, 'Sberry': 100.34, 'Mango': 80}
>>> print(d1)
{'Apple': 25.67, 'Kiwi': 30, 'Sberry': 100.34, 'Mango': 80}
>>> d1.pop("Sberry")-----100.34
>>> print(d1)---{'Apple': 25.67, 'Kiwi': 30, 'Mango': 80}
>>> d1.pop("Mango")-------80
>>> print(d1)-------{'Apple': 25.67, 'Kiwi': 30}
>>> d1.pop("Mangoes")------KeyError: 'Mangoes'
-----------------------------------------------------------------------------
3)  popitem():
-----------------------
=>This Function is used for removing the last (Key,Value ) from dict object
=>When we call popitem() upon empty dict object then we get KeyError
=>Syntax:    dictobj.popitem()
---------------------
Examples:
---------------------
>>> d1={'Apple': 25.67, 'Kiwi': 30, 'Sberry': 100.34, 'Mango': 80}
>>> print(d1)---{'Apple': 25.67, 'Kiwi': 30, 'Sberry': 100.34, 'Mango': 80}
>>> d1.popitem()---('Mango', 80)
>>> print(d1)---{'Apple': 25.67, 'Kiwi': 30, 'Sberry': 100.34}
>>> d1.popitem()----('Sberry', 100.34)
>>> print(d1)----{'Apple': 25.67, 'Kiwi': 30}
>>> d1.popitem()---('Kiwi', 30)
>>> print(d1)---{'Apple': 25.67}
>>> d1.popitem()---('Apple', 25.67)
>>> print(d1)---{}
>>> d1.popitem()---KeyError: 'popitem(): dictionary is empty'
-----------------------------------------------------------------------------
4) get():
----------------------
=>This function is used for obtaining  value of Value by passing value of
Key.
=>If the value of Key does not exists then we get None
=>Syntax:-  varname=dictobj.get(Key)

Examples:
----------------------
>>> d1={'Apple': 25.67, 'Kiwi': 30, 'Sberry': 100.34, 'Mango': 80}
>>> print(d1)--{'Apple': 25.67, 'Kiwi': 30, 'Sberry': 100.34, 'Mango': 80}
>>> v1=d1.get("Apple")
>>> print(v1)---25.67
>>> v1=d1.get("Guava")
>>> print(v1)---None
-----------------------------------------------------------------------------
5) keys()
------------------
=>This Function obtains list of keys from non-empty dict object.
=> when we call keys() upon empty dict then we get empty list
=>Syntax:       keys=dictobj.keys()
Examples:
-----------------
>>> d1={'Apple': 25.67, 'Kiwi': 30, 'Sberry': 100.34, 'Mango': 80}
>>> print(d1)---{'Apple': 25.67, 'Kiwi': 30, 'Sberry': 100.34, 'Mango': 80}
>>> d1.keys()---dict_keys(['Apple', 'Kiwi', 'Sberry', 'Mango'])
>>> ks=d1.keys()
>>> print(ks)----dict_keys(['Apple', 'Kiwi', 'Sberry', 'Mango'])
>>> for k in d1.keys():
...     print(k)
                               ...
                               Apple
                               Kiwi
                               Sberry
                               Mango
>>> dict().keys()---dict_keys([])
>>> {}.keys()---dict_keys([])
>>> {'Apple': 25.67, 'Kiwi': 30, 'Sberry': 100.34, 'Mango': 80}.keys()
                       dict_keys(['Apple', 'Kiwi', 'Sberry', 'Mango'])
-----------------------------------------------------------------------------
6) values()
------------------
=>This Function obtains list of values from non-empty dict object.
=> when we call values() upon empty dict then we get empty list
=>Syntax:      values=dictobj.values()
Examples:
-----------------
>>> d1={'Apple': 25.67, 'Kiwi': 30, 'Sberry': 100.34, 'Mango': 80}
>>> print(d1)---{'Apple': 25.67, 'Kiwi': 30, 'Sberry': 100.34, 'Mango': 80}
>>> vs=d1.values()
>>> print(vs)----dict_values([25.67, 30, 100.34, 80])
>>> d1.values()----dict_values([25.67, 30, 100.34, 80])
>>> for val in d1.values():
...     print(val)
...
                               25.67
                               30
                               100.34
                               80
>>> {'Apple': 25.67, 'Kiwi': 30, 'Sberry': 100.34,}.values()
                                              dict_values([25.67, 30, 100.34])
>>> {}.values()----dict_values([])
>>> dict().values()---dict_values([])
Special Case:
------------------------
>>> d1={'Apple': 25.67, 'Kiwi': 30, 'Sberry': 100.34, 'Mango': 80}
>>> print(d1)---{'Apple': 25.67, 'Kiwi': 30, 'Sberry': 100.34, 'Mango': 80}
>>> for x in d1:
...     print(x)
                       ...
                       Apple
                       Kiwi
                       Sberry
                       Mango

----------------------------------------------------------------------------
7)  items():
----------------
=>This function obtains all (Key,value) from from dict object in the form
tuple.
=>when we call items() upon empty dict object then we get empty list.
Syntax:-     keyvalue=dictobj.items()
-----------------
Examples:
-------------------
```

```
>>> d1={'Apple': 25.67, 'Kiwi': 30, 'Sberry': 100.34, 'Mango': 80}
>>> d1.items()
dict_items([('Apple', 25.67), ('Kiwi', 30), ('Sberry', 100.34), ('Mango',
80)])
>>> kv=d1.items()
>>> print(kv)
dict_items([('Apple', 25.67), ('Kiwi', 30), ('Sberry', 100.34), ('Mango',
80)])
>>> for kv in d1.items():
...     print(kv)
...
('Apple', 25.67)
('Kiwi', 30)
('Sberry', 100.34)
('Mango', 80)
>>> for k,v in d1.items():
...     print(k,"--->",v)
...
Apple ---> 25.67
Kiwi ---> 30
Sberry ---> 100.34
Mango ---> 80

>>> dict().items()-------dict_items([])
----------------------------------------------------------------------------
8) copy()  :
------------------
```

=>This function is used for copying the content of one dict object into another dict object (shallow Copy)

```
=>Syntax:-    dictobj2=dictobj1.copy()

Examples:
-------------------------
>>> d1={'Apple': 25.67, 'Kiwi': 30}
>>> print(d1,id(d1))----------{'Apple': 25.67, 'Kiwi': 30} 2186121118976
>>> d2=d1.copy()
>>> print(d2,id(d2))----{'Apple': 25.67, 'Kiwi': 30} 2186121118656
>>> d2["Sberry"]=23.45
>>> d1["Guava"]=60
>>> print(d1,id(d1))---{'Apple': 25.67, 'Kiwi': 30, 'Guava': 60}
2186121118976
>>> print(d2,id(d2))--{'Apple': 25.67, 'Kiwi': 30, 'Sberry': 23.45}
2186121118656
-----------------------------------------------------------------------------
9)update():
----------------------

Examples:
-------------------
>>> d1={"Praveen":"Python","Kiran":"Java"}
>>> d2={"RS":"Django","DR":"C"}
>>> d3=d1.update(d2)
>>> print(d1)---{'Praveen': 'Python', 'Kiran': 'Java', 'RS': 'Django', 'DR':
'C'}
>>> print(d2)---{'RS': 'Django', 'DR': 'C'}
>>> print(d3)---None
>>> d1={"Praveen":"Python","Kiran":"Java"}
>>> d2={"Praveen":"Django","DR":"C"}
>>> d1.update(d2)
>>> print(d1)--{'Praveen': 'Django', 'Kiran': 'Java', 'DR': 'C'}
>>> print(d2)---{'Praveen': 'Django', 'DR': 'C'}
```

```
#Program for computing sum of two numbers
a=10
b=20
c=a+b
print(a,b,c)
#program sum of two numbers
a=float(input("Enter First Value:"))
b=float(input("Enter Second Value:"))
c=a+b
print("------------------------------")
print("\t\tSumming")
print("------------------------------")
print("Val of a=",a)
print("Val of b=",b)
print("Sum=",c)
print("------------------------------")
---------------------------------------------------------------------------
#Program for sum of two numbers
#sumex2.py
a=float(input("Enter First Value:"))
b=float(input("Enter Second Value:"))
c=a+b
print("------------------------------")
print("\t\tSum")
print("------------------------------")
print("Val of a=",a)
print("Val of b=",b)
print("Sum=",c)
print("------------------------------")
```

```
               =======================================
                       none type   data type
               =======================================
```

=>'NoneType' is one the pre-defined class and treated as None type Data type

=> "None" is keyword acts as value for <class,'NoneType'>

=>The value of 'None' is not False, Space , empty , 0

=>An object of NoneType class can't be created explicitly.

=>If the function is not returning any value and if we print by using print() then we get None as result.

Examples:

```
>>> a=None
>>> print(a,type(a))------------None <class 'NoneType'>
>>> a=NoneType()---------NameError: name 'NoneType' is not defined
>>> d1={10:"ABC",20:"PQR"}
>>> print(d1.get(10))--------ABC
>>> print(d1.get(100))-----None
```

Number of approaches to develop python programs

=>In Python Programming Environment, we have 2 approaches to develop a python Program. They are

a) Interactive Mode
b) Batch Mode

a) Interactive Mode:

=>This Mode of Development, The Python Programmer Issued One statement and got its result immediately and such statements can't be saved . So that we can't re-use in further applications development.

Example Software:-   Python Interactive Command Prompt
                                  Python IDLE Shell

Example   Code:

```
>>> a=100
>>> b=200
>>> c=a+b
>>> print(a,b,c)----100 200 300
```

=>This mode develop is useful for Testing one instruction at time and not recommended to develop bunch of instruction for big problem solving.
=>Industry Recommeded to Batch Mode for developing Batch of instruction for big problem solving..

2) Batch Mode:

=>In Batch Mode Programming, we develop batch of optimized instructions for solving any problem statement and it saved on filename with an extension .py (Source Code).
Examples: sum.py   mul.py   simpleint.py....etc

Example Software:-   Python IDLE Shell
                                  Edit Plus
                                  PyCharm
                                  Jupiter Note Book
                                  Spider
                                  VS CODE
                                  atom-------etc

X

Displaying the Result (or) Data on the Console

=>To Display the Result of Python Program , we use pre-defined Function called

```
    print()
```

=>In otherwords, print() is used for Displaying the Result of python on the console.

=>Syntax-1:   Displaying only Data / values

```
                       print(val1,val2....val-n)
```

```
Examples:
-----------------
>>> a=10
>>> print(a)----------10
>>> s="Python"
>>> print(s)--------Python
>>> print(a,s)-----10 Python
--------------------------------------------------------------------------
=>Syntax-2---------->Displaying Messages with Values
                              print(Message cum Values)
------------------
Examples:
-----------------
>>> a=100
>>> print("Val of a=",a)----Value of a=100
>>> a=100
>>> print(a," is the value of a")----100  is the value of a
>>> a=10
>>> b=20
>>> c=a+b
>>> print("sum=",c)-----sum= 30
>>> print(c," is the sum")---30  is the sum
>>> print("sum of ",a," and ",b,"=",c)---sum of  10  and  20 = 30
>>> a=10
>>> b=20
>>> c=30
>>> d=a+b+c
>>> print("sum of ",a,",",b," and ",c,"=",d)---sum of  10 , 20  and  30 = 60
----------------------------------------------------------------------------
=>Syntax-3:      Displaying Messages with Values by using format()
-----------------
>>> a=10
>>> b=20
>>> c=a+b
>>> print("Sum of {} and {}={}".format(a,b,c) )---Sum of 10 and 20=30
>>> sno=10
>>> name="Rossum"
>>> print("My Roll no:{} and Name:{}".format(sno,name))
                                              My Roll no:10 and Name:Rossum
-----------------------------------------------------------------------------
=>Syntax-4:----Displaying Messages with Values by using format specifiers
------------------
               print("Messages with Format specifiers " %(var1,var2...var-n))
-----------------
Examples:
-----------------
>> a=10
>>> b=20
>>> c=a+b
>>> print("Sum of %d and %d=%d" %(a,b,c))---Sum of 10 and 20=30
>>> print("Sum of %f and %f=%f" %(a,b,c))
                       Sum of 10.000000 and 20.000000=30.000000
>>> print("Sum of %0.2f and %0.2f=%0.3f" %(a,b,c))--Sum of 10.00 and
20.00=30.000
------------------------
>>> a=10
>>> b=20
>>> c=a+b
>>> print("Sum of %d and %d=%d" %(a,b,c))---Sum of 10 and 20=30
>>> print("Sum of %f and %f=%f" %(a,b,c))--
                Sum of 10.000000 and 20.000000=30.000000
>>> print("Sum of %0.2f and %0.2f=%0.3f" %(a,b,c))--Sum of 10.00 and
20.00=30.000
>>> a=2.3
>>> b=3.4
>>> c=a+b
>>> print("sum(%f,%f)=%f" %(a,b,c))---sum(2.300000,3.400000)=5.700000
>>> print("sum(%0.1f,%0.1f)=%0.2f" %(a,b,c))--sum(2.3,3.4)=5.70
>>> print("sum(%d,%d)=%0.2f" %(a,b,c))---sum(2,3)=5.70
>>> print("sum(%d,%d)=%d" %(a,b,c))---sum(2,3)=5
```

==

Reading the Data (or) Input Values from Key Board

==

=>To read the data Dynamically from Keyboard, we have 2 pre-defined functions. They are

a) input()
b) input(Message)

----

a) input()

=>input() is used reading any type of data / value dynamically from keyboard in the form of str always.

=>Syntax:

```
                    varname=input()
```

=>here 'varname' is an object of <class,'str'>. To convert str values into other data type values, we Type casting Functions (int(), float(), bool(), str(), complex()....etc)

=>input() is a pre-defined function and it reads at a time only one value in the form of str.

----

b) input(Message)

```
#Program for accepting two values and find their sum
#sum1.py
print("Enter Value for a:")
a=input()
print("Enter Value for b:")
b=input()
#convert  a and b values into int type
n=int(a)
m=int(b)
res=n+m
print("sum of {} and {}={}".format(n,m,res))
```

```
#Program for accepting two values and find their sum
#sum2.py
print("Enter Two Values for a and b:")
a=input()
b=input()
#convert  a and b values into int type
n=float(a)
m=float(b)
res=n+m
print("sum of {} and {}={}".format(n,m,res))
```

```
#Program for accepting two values and find their sum
#sum3.py
print("Enter Two Values for a and b:")
#convert  a and b values into int type
n=float( input() )
m=float( input() )
res=n+m
print("sum of {} and {}={}".format(n,m,res))
```

```
#Program for accepting two values and find their sum
#sum4.py
print("Enter Two Values for a and b:")
#convert  a and b values into int type
n=float( input() )
m=float( input() )
print("sum of {} and {}={}".format(n,m,n+m))
```

```
#Program for accepting two values and find their sum
#sum5.py
print("Enter Two Values for a and b:")
n,m=float( input() ),float(input())
print("sum of {} and {}={}".format(n,m,n+m))
```

Reading the Data (or) Input Values from Key Board

=>To read the data Dynamically from Keyboard, we have 2 pre-defined functions. They are

a) input()
b) input(Message)

a) input()

=>input() is used reading any type of data / value dynamically from keyboard in the form of str always.

=>Syntax:

```
                   varname=input()
```

=>here 'varname' is an object of <class,'str'>. To convert str values into other data type values, we Type casting Functions (int(), float(), bool(), str(), complex()....etc)

=>input() is a pre-defined function and it reads at a time only one value in the form of str.

b) input(Message)

=>This function is used for reading any type of data / value dynamically from keyboard in the form of str always and additionally it gives User-Prompting Message.

Syntax:-

```
                             varname=input(Message)
```

=>here 'varname' is an object of <class,'str'>. To convert str values into other data type values, we Type casting Functions (int(), float(), bool(), str(), complex()....etc)

=>input(Message) is a pre-defined function and it reads at a time only one value in the form of str and Message represents "User-Prompting Message".

```
#Program for accepting two values and find their mul--input(message)
#mul.py
s1=input("Enter First Value:")
```

```
s2=input("Enter Second Value:")
#convert s1 and s2  and into float type values
v1=float(s1)
v2=float(s2)
v3=v1*v2
print("------------------------------------")
print("Val of v1={}".format(v1))
print("Val of v2={}".format(v2))
print("Mul={}".format(v3))
print("------------------------------------")
```

```
#Program for accepting two values and find their mul--input(message)
#mul1.py
v1=float(input("Enter First Value:"))
v2=float(input("Enter Second Value:"))
v3=v1*v2
print("------------------------------------")
print("Val of v1={}".format(v1))
print("Val of v2={}".format(v2))
print("Mul={}".format(v3))
print("------------------------------------")
```

==

Operators in Python

==

=>An Operator is a symbol, which is used to perform certain operation.

=>Any two or more object / variables connected with an operator is called Expression.

=>In Python Programming, we have 7 types of Operators. They are

```
                1) Arithmetic Operators
                2) Assignment Operator
                3) Relational Operators
                4) Logical Operators
                5) Bitwise Operators (Most Imp)
                6) Membership Operators
                                       a) in
                                       b) not in
                7) Identity Operators
                                       a) is
                                       b) is not
```

1) Arithmetic Operators

=>Arithmetic Operators are used for performing all types of Arithmetic Operations such as addition , substraction, multiplication..etc
=>If two or more valriables / objects are connected with Arithmetic Operators then it is called Arithmetic Expression.
=>The following table gives list of Arithmetic Operators.

Slno	Symbol	Meaning	Examples a=10 b=3
1.	+	Addition	print(a+b)----13
2.	-	Substraction	print(a-b)-----7
3.	*	Multiplication	print(a*b)-----30
4.	/	Division	print(a/b)----3.3333333 (Float Quotient) print(10.0/3.0)--3.333333
5.	//	Floor Divison	print(a//b)--->3 (Integer Quotient) print(10.0//3.0)--->3.0
6	%	Modulo Division	print (a%b)----1
7.	**	Exponentiation	print(a**b)

2) Assignment Operator

=>The purpose of Assigment operator is that to transfer Right hand Side (RHS) value to Left Hand Side (LHS) Variable .
=>We can use Assignment Operator in two ways. They are

a) Single Line Assigment
b) Multi Line Assigment

a) Single Line Assigment:

Syntax: LSHVarname=RHS Var name/Value / Expression

Examples:

```
>>> a=10
>>> b=20
>>> c=a+b
>>> print(a,b,c)
```

b) Multi Line Assigment:

Syntax:

```
LHS var1, LHS var2...LHS var-n=RHS var1 , RHS var2, ...RHS var-n
              (OR)
LHS var1, LHS var2...LHS var-n=RHS Expr1 , RHS Expr2, ...RHS Expr-n
```

```
Examples:
--------------------
>>> a,b=10,20
>>> c,d,e=a+b,a-b,a*b
>>> print("c=",c)--------c= 30
>>> print("d=",d)-------d= -10
>>> print("e=",e)-------e= 200
```

```
#aop.py
a=float(input("Enter Value of a:"))
b=float(input("Enter Value of b:"))
print("="*50)
print("\tArithmetic Operations")
print("="*50)
print("\tSum({},{})={}".format(a,b,a+b))
print("\tSub({},{})={}".format(a,b,a-b))
print("\tMul({},{})={}".format(a,b,a*b))
print("\tDiv({},{})={}".format(a,b,a/b))
print("\tInetger Div({},{})={}".format(a,b,a//b))
print("\tMod({},{})={}".format(a,b,a%b))
print("\texpo({},{})={}".format(a,b,a**b))
print("="*50)
```

```
#sqrt.py
n=float(input("Ente a number:"))
res=n**(1/2)
print("sqrt({})={}".format(n,res))
```

```
#swap.py
a,b=input("Enter Value of a:"),input("Enter Value of b:")
print("="*50)
print("Original value of a={}".format(a))
print("Original value of b={}".format(b))
print("="*50)
#swapping logic
a,b=b,a   # multi line assigment
print("Swapped value of a={}".format(a))
print("Swapped value of b={}".format(b))
print("="*50)
```

```
#program cal simple interest
#simpleint.py
p=float(input("Enter Principle Amount:"))
t=float(input("Enter Time:"))
r=float(input("Enter Rate of Interest:"))
#cal si
si=(p*t*r)/100
totamt=p+si
#display the results
print("="*60)
print("\tResult of Simple Interest")
print("#"*60)
print("\tPrinciple Amount:{}".format(p))
print("\tTime:{}".format(t))
print("\tRate of Interest:{}".format(r))
print("\tSimple Rate of Interest on principle:{}".format(si))
print("\tTotal Amount to Pay:{}".format(totamt))
print("*"*60)
```

Logical Operators in Python

=>The purpose of Logical Operators is that " To combine two or more number of Relational Expressions / Conditions".
=>If two or more Relational Expressions / Conditions connected with Logical Operators then it is called Logical Expression / Compound Condition.
=>The reuslt of Logical Expression / Compound Condition is True or False
=>The Logical Operators are given in the following table

slno	symbol	meaning
1.	or	Physical ORing
2.	and	Physical ANDing
3.	not	-------------------------

1) or (Physical ORing)

=>The Functionality of 'or' operator is shown in the following truth table

=>Syntax:- RelExpr1 or RelExpr2

RelExpr1	RelExpr2	RelExpr1 or Relexpr2
True	False	True
False	True	True
False	False	False
True	True	True

Examples:

```
>>> 10>20 or 20!=30------------True
>>> 10!=20 or 10>20---------True
>>> 10!=100 or 10>20 or 20<10---True
>>> 10>20 or 10!=20  or -10!=-20---True
>>> 10>20 or 10==20  or -10!=-20----True
>>> 10>20 or 10==20  or -10<=-20----False
```

2) and (Physical ANDing)

=>The Functionality of 'and' operator is shown in the following truth table

=>Syntax:- RelExpr1 and RelExpr2

RelExpr1	RelExpr2	RelExpr1 and Relexpr2
True	False	False
False	True	False
False	False	False
True	True	True

Examples:

```
>>> 100>=200  and 10<5-------------False
>>> 100>=20  and 10!=5-----------True
>>> 100>=20  and 10!=5 or 10!=20---True
>>> 100<=20  and 10<=5 or 10!=20----True
>>> 100!=-100 and 100==20 and 20!=4 or 4!=5----True
```

---

3. not :

=>The Functionality of 'not' operator is shown in the following truth table

```
=>Syntax:-       not  repexr1
                      not(RelExpr1 or RelExpr2)
                      not (RelExpr1 and RelExpr2)
```

RelExpr1	not RelExpr1
True	False
False	True

```
>>> 10>5 and 10!=2-----------True
>>> not (10>5 and 10!=2)---------False
>>> 10<5 or 10!=10------False
>>> not (10<5 or 10!=10)--------True
>>> not 10>20-------True
>>> not 10==10---------False
>>> not True----False
>>> not False--------True
>>> not 100------------False
>>> not 0---------True
>>> not not 0-----------False
>>> not not---------SyntaxError: invalid syntax
>>> not not ""------------False
>>> not "Python"-----False
>>> "True"=="False"----------False
>>> True==False------False
```

Relational Operators in Python

=>The purpose of Relational Operators is that "To compare two or more number of values".

=> If two or more number of values / variables connected with Relational Operators then it is called Relational Expression.

=>The result of relational expression is either True or False

=>Always relational Expressions called Conditions.

=>The Relational Operators are given following table

slno	symbol	meaning	Examples a=10 b=20 c=10
1	>	Greater Than	print(a>b)------False print(b>c)------True
2.	<	Less Than	print(a<b)------True print(a<c)------False
3.	==	Equality	print(a==b)----False print(a==c)----True
4.	!=	not equal to	print(a!=c)-----False print(a!=b)-----True
5.	>=	greater than or equal to	print(a>=c)-----True print(a>=b)-----False
6.	<=	Less Than or equal to	print(a<=b)------True print(100<=b)---False

```
=================================================================
#rop.py
#program for demonstrating Relation Operators.
a=int(input("Enter Value of a:"))
b=int(input("Enter Value of b:"))
print("-"*50)
print("Results of Relational Operators:")
print("-"*50)
print("\t{}>{}={}".format(a,b,a>b))
print("\t{}<{}={}".format(a,b,a<b))
print("\t{}=={}={}".format(a,b,a==b))
print("\t{}!={}={}".format(a,b,a!=b))
print("\t{}>={}={}".format(a,b,a>=b))
print("\t{}<={}={}".format(a,b,a<=b))
print("-"*50)
```

```
                ========================================
                     Bitwise Operators (Most Imp)
                ========================================
```

=>Bitwise Operators applied only on Integer Values but not on float values.

=>Bitwise Operators converts given Integer data into Binary format and performs operations on binary data in the form of Bit by Bit and hence they named as Bitwise Operators.

=>In Python Programming, we have 6 Bitwise Operators. They are

```
               1) Bitwise Left shift Operator (  << )
               2) Bitwise Right shift Operator ( >> )
               3) Bitwise OR Operator (  |  )
               4) Bitwise AND Operator (  &  )
               5) Bitwise complement Operator (  ~  )
               6) Bitwise XOR Operator (  ^  )
-----------------------------------------------------------------------------
1) Bitwise Left shift Operator (  << ):
------------------------------------------------------
Syntax:-            resultantvar = GivenData <<  no.of bits
```

=>This operators shifts Specfied No. of Bits toward left side by adding no. Zeros which are equal to no.of bits at right side.

```
-------------------
Examples:
-------------------
>>> print(10<<3)-----------80
>>> a=10
>>> b=3
>>> c=a<<b
>>> print(c)--------80
>>> print(12<<2)-----48
--------------------------------------------------------------------------
2) Bitwise Right shift Operator (  >> ):
------------------------------------------------------
Syntax:-            resultantvar = GivenData >>  no.of bits
```

=>This operators shifts Specfied No. of Bits toward right side by adding no. Zeros which are equal to no.of bits at Left side.

```
Examples:
------------------
>>> a=10
>>> b=3
>>> c=a>>b
>>> print(c)----------1
>>> print(12>>2)----3
>>> print(16>>3)----2
```

3) Bitwise OR Operator (|):

=>Syntax: resultantvar= value1 | value2

=>The Functionality of Bitwise OR Operator (|) is shown in the following table.

Value1	Value2	Value1 | Value2
0	0	0
0	1	1
1	0	1
1	1	1

Examples:

```
>>>a=4--------------- 0100
>>>b=5----------------0101
-------------------------------------
>>>c=a|b------------>0101--------------Result  is  5
-----------------------------------------------------------------
>>>print(10|15)--------------->   1010
                                   1111
                               ------------
                                   1111----Result--15
```

Special Case of Bitwise OR (|)

```
>>>s1={10,20,30}
>>>s2={30,40,50}
>>>s3=s1.union(s2)
>>> print(s3)-----------{50, 20, 40, 10, 30}
>>>
>>> s4=s1|s2
>>> print(s4,type(s4) )---------{50, 20, 40, 10, 30} <class 'set'>
```

4) Bitwise AND Operator (&):

=>Syntax: resultantvar= value1 & value2

=>The Functionality of Bitwise AND Operator (&) is shown in the following table.

Value1	Value2	Value1 & Value2
0	0	0
0	1	0
1	0	0
1	1	1

Examples:

```
>>>a=5------------------> 0101
>>>b=4----------------->  0100
-----------------------------------------------
```

```
>>>c=a&b------------->  0100---------------Result is 4
>>>print(c)---------4
>>> print(15&10)-------10
>>> print(7&2)--------2
```

Special Case:

```
>>>s1={10,20,30}
>>>s2={30,40,50}
>>>s3=s1.intersection(s2)
>>>print(s3)-----{30)

>>>s4=s1&s2
>>>print(s4)-------{30}
```

----

5) Bitwise complement Operator (~)

Syntax:-   resultantvar= ~Value

Examples:

```
>>>a=10----------------------->   1010
>>>b=~a------------------------>  ~(1010+0001)
                       ---------------> - (1010
                                          0001)
                                        ------------
                                        -( 1011 )------------->result is  -11
```

Formula:-   >>> ~varval   -----------> -(varval+1)

```
>>>print(~20)------------> -21
>>> a=98
>>> c=~a
>>> print(c)-----------99
>>> print(~20)---------21
>>> print(~(-101))------100
```

===

6) Bitwise XOR Operator (^)

=>Syntax: resultantvar= value1 ^ value2

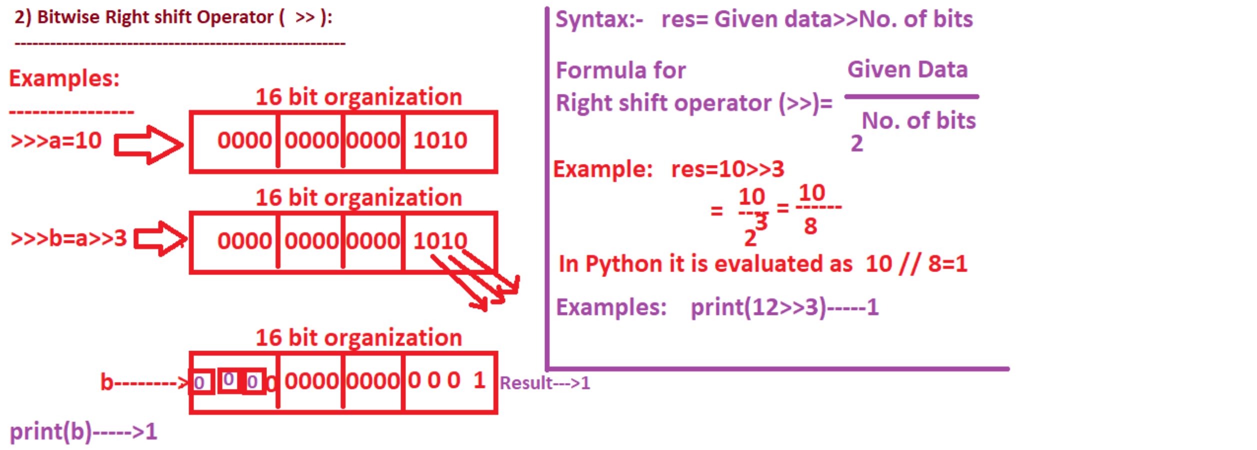
=>The Functionality of Bitwise XOR Operator (^) is shown in the following table.

Value1	Value2	Value1 ^ Value2
0	0	0
0	1	1
1	0	1
1	1	0

Examples:

```
>>>a=10----------- 1010
>>>b=4------------ 0100
--------------------------------------------
>>>c=a^b----------1110------------>Result is 14
```

```
---------------------------------------------------------
>>> a=15
>>> b=10
>>> print(a^b)------------5
------------------------------------------------------------------
Special case:
---------------------
>>>s1={10,20,30}
>>>s2={30,40,50}
>>>s3=s1.symmetric_difference(s2)
>>> print(s3)--------{40, 10, 50, 20}
>>> s4=s1^s2
>>> print(s4)-------{40, 10, 50, 20}
-----------------------------------------------------------------
```



2) Bitwise Right shift Operator (>>):

Examples:

```
>>>a=10
>>>b=a>>3
b--------->
print(b)----->1
```

Syntax:- res= Given data>>No. of bits

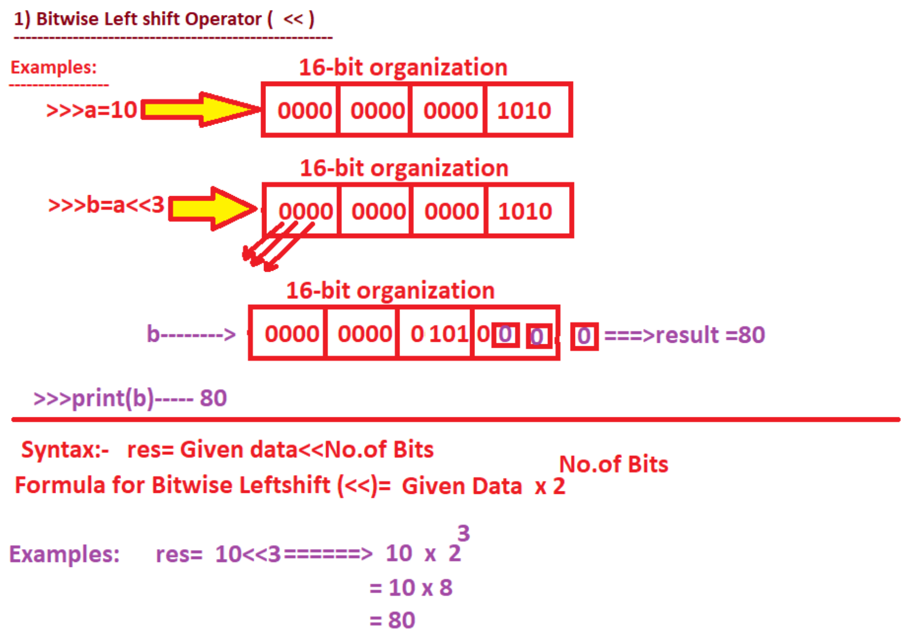
Formula for Right shift operator (>>)= $\frac{\text{Given Data}}{2^{\text{No. of bits}}}$

Example: res=10>>3

$= \frac{10}{2^3} = \frac{10}{8}$

In Python it is evaluated as 10 // 8=1

Examples: print(12>>3)-----1



1) Bitwise Left shift Operator (<<)

Examples:

16-bit organization

>>>a=10 → 0000 0000 0000 1010

16-bit organization

>>>b=a<<3 → 0000 0000 0000 1010

16-bit organization

b--------> 0000 0000 0 101 0 0 0 0 ===>result =80

>>>print(b)----- 80

Syntax:- res= Given data<<No.of Bits

Formula for Bitwise Leftshift (<<)= Given Data x $2^{\text{No.of Bits}}$

Examples: res= 10<<3======> 10 x 2^{3}

= 10 x 8

= 80

Identity Operators

=>The purpose of Identity Operators is that to "To compare the memory addresses of two objects"

=>In Python Programming, we have 2 types of Identity Operators. They are

a) is
b) is not

a) is:

Syntax:-   obj1 is obj2

=>"is" operator returns True provided both obj1 and obj2 contains same memory address otherwise it returns False

b) is not:

Syntax:-   obj1 is not obj2

=>"is not " operator returns True provided both obj1 and obj2 contains different memory address otherwise it returns False

Examples:

```
>>> v1=None
>>> v2=None
>>> print(v1,id(v1))
None 140703467931640
>>> print(v2,id(v2))
```

```
None 140703467931640
>>> v1 is v2
True
>>> v1 is not v2
False
---------------------------------
>>> d1={10:"Apple",20:"Mango"}
>>> d2={10:"Apple",20:"Mango"}
>>> print(d1,id(d1))
{10: 'Apple', 20: 'Mango'} 1772239617856
>>> print(d2,id(d2))
{10: 'Apple', 20: 'Mango'} 1772239667328
>>> d1 is d2
False
>>> d1 is not d2
True
--------------------------------------------------------
>>> s1={10,20,30}
>>> s2={10,20,30}
>>> print(s1,id(s1))
{10, 20, 30} 1772239875520
>>> print(s2,id(s2))
{10, 20, 30} 1772239874176
>>> s1 is s2
False
>>> s1 is not  s2
True
>>> fs1=frozenset(s1)
>>> fs2=frozenset(s2)
>>> print(fs1,id(fs1))
frozenset({10, 20, 30}) 1772239874400
>>> print(fs2,id(fs2))
frozenset({10, 20, 30}) 1772239875968
>>> fs1 is fs2
False
>>> fs1 is not fs2
True
----------------------------------------------------
>>> l1=[10,20,30,40]
>>> l2=[10,20,30,40]
>>> print(l1,id(l1))
[10, 20, 30, 40] 1772239681536
>>> print(l2,id(l2))
[10, 20, 30, 40] 1772239728128
>>> l1 is l2
False
>>> l1 is not l2
True
>>> t1=tuple(l1)
>>> t2=tuple(l2)
>>> print(t1,id(t1))
(10, 20, 30, 40) 1772239704032
>>> print(t2,id(t2))
(10, 20, 30, 40) 1772239581520
```

```
>>> t1 is t2
False
>>> t1 is not t2
True
------------------------------------------------------------
>>> s1="PYTHON"
>>> s2="PYTHON"
>>> print(s1, id(s1))
PYTHON 1772239928944
>>> print(s2, id(s2))
PYTHON 1772239928944
>>> s1 is s2
True
>>> s1 is not s2
False
>>> s1="kvr"
>>> s2="kvr"
>>> s1 is s2
True
>>> s1 is not s2
False
>>> print(s1,id(s1))
kvr 1772239979184
>>> print(s2,id(s2))
kvr 1772239979184
-----------------------------------------------------------------------
>>> l1=[10,20,30]
>>> b1=bytes(l1)
>>> b2=bytes(l1)
>>> print(b1,type(b1))
b'\n\x14\x1e' <class 'bytes'>
>>> print(b1,id(b1))
b'\n\x14\x1e' 1772239838192
>>> print(b2,id(b2))
b'\n\x14\x1e' 1772239837472
>>> b1 is b2
False
>>> b1 is not b2
True
>>> ba=bytearray(l1)
>>> bb=bytearray(l1)
>>> print(ba, id(ba))
bytearray(b'\n\x14\x1e') 1772239979696
>>> print(bb, id(bb))
bytearray(b'\n\x14\x1e') 1772239979312
>>> ba is bb
False
>>> ba is not bb
True
--------------------------------------------------------------------------
>>> r1=range(10,20)
>>> r2=range(10,20)
>>> print(r1,id(r1))
range(10, 20) 1772239837424
```

```
>>> print(r2,id(r2))
range(10, 20) 1772239837760
>>> r1 is r2
False
>>> r1 is not r2
True
---------------------------------------------------------------------------
>> a=2+3j
>>> b=2+3j
>>> print(a, id(a))
(2+3j) 1772239378832
>>> print(b, id(b))
(2+3j) 1772239378576
>>> a is b
False
>>> a is not b
True
----------------------------------------
>>> b1=True
>>> b2=True
>>> print(b1, id(b1))
True 140703467879272
>>> print(b2, id(b2))
True 140703467879272
>>> b1 is b2
True
>>> b1 is not b2
False
------------------------------------------
>>> a=12.34
>>> b=12.34
>>> print(a, id(a))
12.34 1772239374960
>>> print(b, id(b))
12.34 1772239374352
>>> a is b
False
>>> a is not b
True
------------------------------------------------
>>> a=10
>>> b=10
>>> print(a, id(a))
10 1772238340624
>>> print(b, id(b))
10 1772238340624
>>> a is b
True
>>> a is not b
False
>>> a=256
>>> b=256
>>> print(a, id(a))
256 1772238348496
```

```
>>> print(b, id(b))
256 1772238348496
>>> a is b
True
>>> a is not b
False
>>> b=257
>>> a=257
>>> print(a, id(a))
257 1772239378416
>>> print(b, id(b))
257 1772239378832
>>> a is b
False
>>> a is not b
True
>>> a=101
>>> b=101
>>> a is b
True
>>> a is not b
False
>>> a=-3
>>> b=-3
>>> print(a, id(a))
-3 1772238340208
>>> print(b, id(b))
-3 1772238340208
>>> a is b
True
>>> a is not b
False
>>> a=-5
>>> b=-5
>>> print(a, id(a))
-5 1772238340144
>>> print(b, id(b))
-5 1772238340144
>>> a is b
True
>>> a is not b
False
>>> a=-6
>>> b=-6
>>> print(a, id(a))
-6 1772239378608
>>> print(b, id(b))
-6 1772239378416
>>> a  is b
False
>>> a  is not b
True
---------------------------------------------------------------------------
```

```
>>> a,b=300,300
>>> print(a,id(a))
300 1772239378832
>>> print(b,id(b))
300 1772239378832
>>> a is b
True
>>> a is not b
False
>>> a,b=-10,-10
>>> print(a,id(a))
-10 1772239378768
>>> print(b,id(b))
-10 1772239378768
>>> a is b
True
>>> b is a
True
================================================
>>> a,b=[10,20],[10,20]
>>> print(a,id(a))
[10, 20] 1772239681536
>>> print(b,id(b))
[10, 20] 1772239928704
>>> a is b
False
>>> a is not b
True
>>> a,b={10:"App",20:"Mango"},{10:"App",20:"Mango"}
>>> print(a,id(a))
{10: 'App', 20: 'Mango'} 1772239722496
>>> print(b,id(b))
{10: 'App', 20: 'Mango'} 1772239617920
==============================X================================
```

Membership Operators

=>The purpose of Membership Operators in python is that "To verify / check the existence of whether the value present in sequence or collection obejcts"

=>In Python Programming, we have 2 Membership Operators. They are

a) in

b) not in

a) in:

```
Syntax:-          Value in Sequence / Collection object
```

=>"value" represents the value to check in Sequence or Collection object

=>Here sequence objects represents (str,bytes, bytearray and range) and collect objects represents (list, tuple,set,frozenset,dict)

=>Here "in" operator returns True provided Value present / exists in Sequence / Collection objects"

=>Here "in" operator returns False provided Value not present / exists in Sequence / Collection objects".

```
------------------
Examples:
-----------------
>>> l1=[10,20,30,40,50,-34]
>>> 20 in l1--------True
>>> -34 in l1----True
>>> 34 in l1-----False
>>> "KVR" in l1-------False
>>> -43 in l1-----False
------------------------------------------------------
b) not in:
-------------------
Syntax:-          Value not in Sequence / Collection object

=>"value" represents the value to check in Sequence or Collection object
=>Here sequence objects represents (str,bytes, bytearray and range) and
collect objects represents (list, tuple,set,frozenset,dict)
=>Here "not in" operator returns True provided Value not present / exists
in Sequence / Collection objects"
=>Here "not in" operator returns False provided Value present / exists  in
Sequence / Collection objects".
--------------------------------
Examples:
--------------------------------
>>> l1=[10,20,30,40,50,-34]
>>> 100 not in l1----------True
>>> 10 not in l1--------False
>>> "KVR" not in l1--------True
>>> 30 not in l1-----------False
--------------------------------------------------------------------------
>>> s="PYTHON"
>>> print(s,type(s))------------PYTHON <class 'str'>
>>> "p" in s-------False
>>> "P" in s--------True
>>> "ON" in s-----True
>>> "HON" in s----True
>>> "HON" not in s-------False
>>> "NO" in s-----False
>>> "HNO" not in s----True
>>> "HNO"  in s-----False
>>> print(s,type(s))-----PYTHON <class 'str'>
>>> s[0] not in s----------False
>>> s[0:2:-1] not in s---------False
>>> s[::] not in s[::-1]--------True
>>> s[::]!=s[::-1]--------True
>>> s[::]==s[::-1]------False
=========================X=====================================
#swapxor.py
a=int(input("Enter Value of a:"))
b=int(input("Enter Value of b:"))
print("-"*40)
print("Original Value of a:{}".format(a))
print("Original Value of b:{}".format(b))
print("-"*40)
```

```
#swapping logic by busing XOR ( ^ )
a=a^b
b=a^b
a=a^b
print("Swapped Value of a:{}".format(a))
print("Swapped Value of b:{}".format(b))
print("-"*40)
```

Flow control statements in python

=>The purpose of Flow control statements in python is that "To perform certain operation one time (Perform X-Operation in the case of True (or) Peform Y-Operation in the case of False) (or) Peform certain operation repeatedly for finite number of times until Condition is False. "

=>In Python Programming, we have 3 types of Flow control statements in python. They are

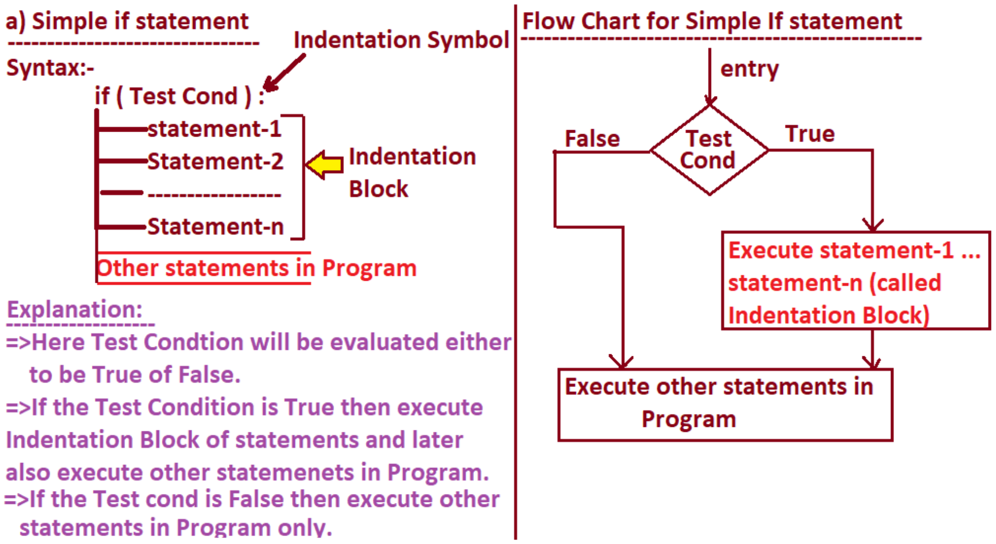
1) Conditional (or) Selection (or) Branching Statements
2) Looping (or) Iterative (or) Repeatative Statements
3) Misc Control statements.

1) Conditional (or) Selection (or) Branching Statements

=>The purpose of Conditional statements is that "To perform Certain Operation i.e X-operation in the case of True or Y-Operation in the case of False only Once."

=>In Python Programming, we have 4 Conditional statements. They are

a) Simple if statement
b) if..else statement
c) if..elif..else statement
d) match...case statement(Python 3.10 version)



a) Simple if statement

Syntax:-

```
if ( Test Cond ) :
    statement-1
    Statement-2
    ---------------
    Statement-n
Other statements in Program
```

Indentation Symbol

Indentation Block

Explanation:

=>Here Test Condtion will be evaluated either to be True of False.

=>If the Test Condition is True then execute Indentation Block of statements and later also execute other statemenets in Program.

=>If the Test cond is False then execute other statements in Program only.

Flow Chart for Simple If statement

entry
Test Cond
False
True
Execute statement-1 ... statement-n (called Indentation Block)
Execute other statements in Program

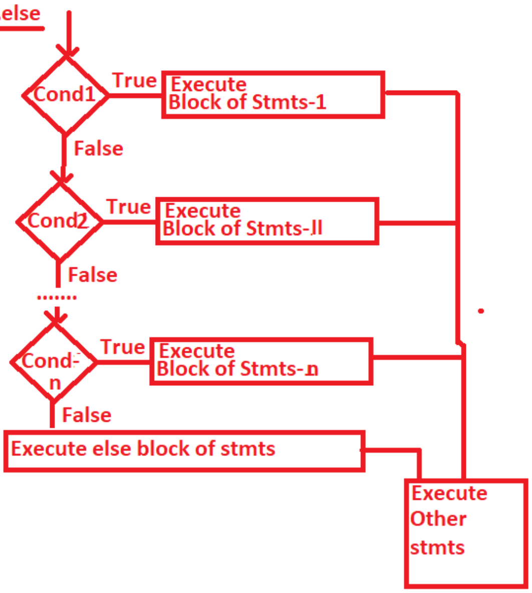
```
#big.py
a=int(input("Enter Value of a:")) # a=10
b=int(input("Enter Value of b:")) #b=20
if(a==b):
        print("Both the values are Equal")
if(a>b):
        print("big({},{})={}".format(a,b,a))   #  big(100,20)=100
if(b>a):
        print("big({},{})={}".format(a,b,b))
```

```
#bigthree.py
a=int(input("Enter First Value:")) # 10
b=int(input("Enter Second Value:"))# 10
c=int(input("Enter Third Value:"))# c=10
if(a>b) and (a>c):
        print("big({},{},{})={}".format(a,b,c,a))
if(b>a) and (b>c):
        print("big({},{},{})={}".format(a,b,c,b))
if(c>a) and (c>b):
        print("big({},{},{})={}".format(a,b,c,c))
if(a==b)and (b==c):
        print("ALL VALUES ARE EQUAL")
```

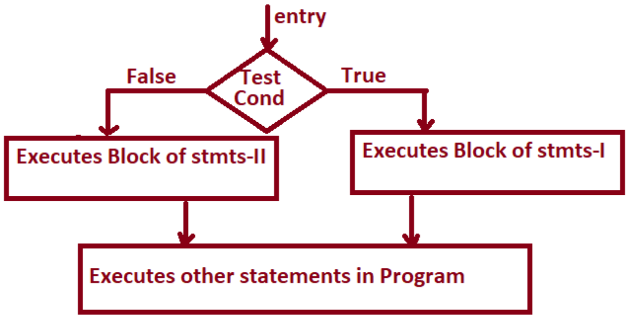
```
#moviee.py
tkt=input("Do u have a ticket(yes/no):")
if(tkt=="yes"):
        print("Enter into theater")
        print("watch the moviee")
        print("Eat the snacks!")
print("\nGoto Home:")
```

```
#zeroposneg.py
n=float(input("Enter a value:"))    #  n= -5
if(n==0):
        print("{} is ZERO".format(n))
if(n>0):
        print("{} is +VE".format(n))

if(n<0):
        print("{} is -VE".format(n))

print("Program execution over")
```

flow chart for if..elif..else



flow chart of if..else statement


entry
False
Test Cond
True
Executes Block of stmts-II
Executes Block of stmts-I
Executes other statements in Program

2) if ..else statement:

syntax:

```
if ( Test Cond ):
    statement-1
    statement-2       Block of stmts-I
    ------------------
    statement-n
else:
    statement-11
    statement-12      Block of stmts-II
    ------------------
    statement-nn
Other statements in Program
```

Explanation:

=>If the Test Cond is True then PVM executes Block of statements-I and also executes Other statements in program(without executing block of statement-II)

=>If the Test Cond is False then PVM executes Block of statements-II and also executes Other statements in program(without executing block of statement-I)

```
#digit.py
d=int(input("Enter a Digit:"))  # d=123
if(d==0):
        print("{} is ZERO".format(d))
elif(d==1):
        print("{} is ONE".format(d))
elif(d==2):
        print("{} is TWO".format(d))
elif(d==3):
        print("{} is THREE".format(d))
elif(d==4):
        print("{} is FOUR".format(d))
elif(d==6):
        print("{} is SIX".format(d))
elif(d==7):
        print("{} is SEVEN".format(d))
elif(d==5):
        print("{} is FIVE".format(d))
elif(d==8):
        print("{} is EIGHT".format(d))
elif(d==5):
        print("{} is NINE".format(d))
else:
        print("It is a number:")

print("\nProgram execution over")
```

```
                ========================================
                       match ...case   concept
               ========================================
Syntax:-
-------------
       match    ChoiceExpression:
               case  label1:   Block of statement-1
               case  label2:   Block of statement-2
               --------------------------------------------------
               --------------------------------------------------
               case  label-n:   Block of stetement-n
               case _:
                       default Case Block statements
        ----------------------------------------------------------
        Other statements in Program
        ----------------------------------------------------------
```

```
-----------------------
Explanation:
------------------------
```

=>The ChoiceExpression can be either int, str, bool etc (except float and complex)
=>If the Value of ChoiceExpression is equal to Case Label1 then PVM executes corresponding Block of stateements-1 and later executes other statements in Program.
=>If the Value of ChoiceExpression is not equal to Case Label1 then PVM compares Value of ChoiceExpression with Case Label2 and if it is equal

then executes corresponding Block of stateements-2 and later executes other statements in Program.

=>This process will be continued with all case labels. In general if the value of Choice Expression is equal to any of the specified Case Labels then PVM executes corresponding block of statements and later executes Other statements in Program.

=>If the Value of ChoiceExpression is not matching with any case labels then PVM executes the block of statements written within default case block (case _ :) and later exeutes Other statements in Program.

```
#matchcaseex1.py
wkno=int(input("Enter Week Number:"))
match wkno:
        case 1:
                       print("Its MONDAY")
        case 2:
                       print("Its TUESDAY")
        case 3:
                       print("Its WEDNESDAY")
        case 4:
                       print("Its THURSDAY")
        case 5:
                       print("Its FRIDAY")
        case 6:
                       print("Its SATDAY")
        case 7:
                       print("Its SUNDAY")
        case _:
                       print("Its not a week number--learn weeks  ")
print("Program over")
```

```
#matchcaseex2.py
wkno=int(input("Enter Week Number:"))
match wkno:
        case 1|2|3|4|5|6:
                       print("Its working")
        case 7:
                       print("Its SUNDAY_holy Day and Joy day")
        case _:
                       print("Its not a week number--learn weeks  ")
print("Program over")
#matchcaseex3.py
wkno=input("Enter Week Name:")
match wkno[0:3].lower():
        case "mon"|"tue"|"wed"|"thu"|"fri":
                       print("{} is working".format(wkno))
        case "sun":
                       print("{} is Holiday".format(wkno))
        case "sat":
                       print("{} is week end".format(wkno))
        case _:
                       print("Its not a week number--learn weeks  ")
print("Program over")
```

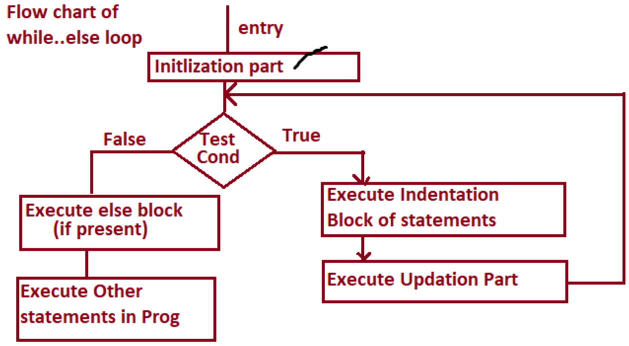
```
#matchcaseex5.py
d={"MONDAY":1,
      "TUESDAY":2,
      "WEDNESSDAY":3,
       "THURSDAY":4,
      "FRIDAY":5,
      "SATURDAY":6,
          "SUNDAY":7}
wkn=input("Enter Week Name:")
if (d.get(wkn.upper())==None):
        print("Invalid Week Name:")
else:
        match wkn[0:3].lower():
                case "mon"|"tue"|"wed"|"thu"|"fri":
                               print("{} is working".format(wkn))
                case "sun":
                               print("{} is Holiday".format(wkn))
                case "sat":
                               print("{} is week end".format(wkn))
```

```
#payslip.py
eno=int(input("Enter Employee Number:"))
ename=input("Enter Employee Name:")
basicsal=float(input("Enter Basic Salary of employee:"))
if(basicsal<=0):
        print("Invalid salary:")
else:
        if(basicsal>=10000):
                da=basicsal*(20/100)
                ta=basicsal*(15/100)
                hra=basicsal*(15/100)
                ma=basicsal*(5/100)
                gpf=basicsal*(2/100)
                lic=basicsal*(2/100)
        else:
                da=basicsal*(30/100)
                ta=basicsal*(25/100)
                hra=basicsal*(20/100)
                ma=basicsal*(10/100)
                gpf=basicsal*(1/100)
                lic=basicsal*(1/100)
        netsal=(basicsal+da+ta+hra+ma)-(gpf+lic)
        print("*"*50)
        print("Employee Number:{}".format(eno))
        print("Employee Name:{}".format(ename))
        print("Employee Basic Salary:{}".format(basicsal))
        print("Employee DA:{}".format(da))
        print("Employee TA:{}".format(ta))
        print("Employee HRA:{}".format(hra))
        print("Employee MA:{}".format(ma))
        print("Employee GPF:{}".format(gpf))
        print("Employee LIC:{}".format(lic))
        print("-"*50)
        print("Net Salary:{}".format(netsal))
        print("*"*50)
```

```
                 =====================================
                       a) while (or) while ...else
                 =====================================
Syntax1:-
============

     -------------------------
     while( Test Cond ):
               statement-1
               statement-2
               -------------------
               -------------------
               statement-n
     ------------------------------
     ------------------------------
     Other statements in Prog
     -------------------------------------

Syntax2:-
============
 -------------------------
     while( Test Cond ):
               statement-1
               statement-2
               -------------------
               -------------------
               statement-n
     else:
             else block of statements

     ------------------------------
     ------------------------------
     Other statements in Prog
     -------------------------------------
```

Explanation:

=>Here 'while' and 'else' are the keywords
=>Test condition result may be True of False
=>In the while loop, if the test condition is true then PVM executes Indentation block of statements and once again PVM control goes to Test Cond. If the Test Cond is once again True then PVM executes Indentation block of statements once again. This Process will be continued until Test Cond becomes False.
=>Once The test cond becomes False then PVM execute else block of statememnts, which are written in else block and later also executes other statements in program.



```
#Factors.py
n=int(input("Enter a number to find its Factors:"))
if(n<=0):
        print("{} is invalid input:".format(n))
else:
        print("-"*40)
        print("Factors of {}".format(n))
        print("-"*40)
        i=1
        while(i<=n//2):
                if(n%i==0):
                        print("\t{}".format(i))
                i=i+1
        else:
                print("-"*40)
```

```
#MulTable.py
n=int(input("Enter a number:"))
if(n<=0):
        print("{} is invalid input:".format(n))
else:
        print("-"*50)
        print("Mul Table for :{}".format(n))
        print("-"*50)
        i=1
        while(i<=10):
                print("\t{} x {} = {}".format(n,i,n*i))
                i=i+1
        else:
                print("-"*50)
```

```
#NatNumsSum.py
n=int(input("Enter a Natural Number:"))
if(n<=0):
        print("{} is invalid input:".format(n))
else:
        print("-"*50)
        print("\tNat Nums\tSquares\t\tCubes")
        print("-"*50)
        s,ss,cs=0,0,0
        i=1
        while(i<=n):
                print("\t{}\t\t{}\t\t{}".format(i,i**2,i**3))
                s=s+i
                ss=ss+i**2
                cs=cs+i**3
                i=i+1
        else:
                print("-"*50)
                print("\t{}\t\t{}\t\t{}".format(s,ss,cs))
                print("-"*50)
```

```
#NumGenEx1.py
n=int(input("Enter How many number u want to generate:"))  # 10  -10   0
if (n<=0):
        print("{} is invalid input:".format(n))
else:
        print("-"*50)
        print("Numbers within {}".format(n))
        print("-"*50)
        i=1 # initlization part
        while(i<=n): # cond part
                print("\t\t{}".format(i))
                i=i+1   #updation part
        else:
                print("*"*50)
        print("Program execution completed:")
```

```
#NumGenEx2.py
#Program to generate 1 to n
n=int(input("Enter How many number u want to generate:"))  # 10  -10   0
if (n<=0):
        print("{} is invalid input:".format(n))
else:
        print("-"*50)
        print("Numbers within {}".format(n))
        print("-"*50)
        i=1 # initlization part
        while(i<=n): # cond part
                print("\t\t{}".format(i))
                i=i+1   #updation part
        print("*"*50)
        print("Program execution completed:")
```

```
#NumGenEx3.py
#Program to generate n to 1
n=int(input("Enter How many number u want to generate:"))  # 10  -10   0
if (n<=0):
        print("{} is invalid input:".format(n))
else:
        print("-"*50)
        print("Numbers within {}".format(n))
        print("-"*50)
        i=n # initlization part
        while(i>=1): # cond part
                print("\t\t{}".format(i))
                i=i-1   #updation part
        print("*"*50)
        print("Program execution completed:")
```

```
#DigitsSum.py
n=int(input("Enter the number:")) # n=123    -123   0
if(n<=0):
        print("{} invalid input:".format(n))
else:
        s=0
        while(n>0):
                d=n%10
                s=s+d
                n=n//10
        else:
                print("Sum of Digits={}".format(s))
```

```
#program for generating 10 12 14 16 18 20
#forex1.py
for i in range(10,21,2):
        print("Value of i=",i)
else:
        print("i am from else block:")
print("Program execution completed!")
```

```
#program for fenerating mul table
#forex2.py
import time
n=int(input("Enter a number:"))
if(n<=0):
        print("{} is invalid input:".format(n))
else:
        print("Mul table for {}".format(n))
        print("-------------------------------------")
        for i in range(1,11):
                print("\t{} x {} = {}".format(n,i,n*i))
                time.sleep(1)
        else:
                print("-------------------------------------")
```

```
#forex3.py
s=0
n=input("Enter a number:")
for i in n:
        x=int(i)
        s=s+x
else:
        print("Sum({})={}".format(n,s))
```

```
#searchex1.py
n=int(input("Enter How Many numbers u have:"))
if(n<=0):
        print("{} is invalid input:".format(n))
else:
        l=list()  # creating empty list
        for i in range(1,n+1):
                val=input("Enter {} value:".format(i))
                l.append(val)
        else:
                print("-----------------------------------------")
                print("Content of list={}".format(l))
                print("-----------------------------------------")
                element=input("Enter which element u want to search:")
                res=l.count(element)
                if(res>0):
                        print("Search is sucessful:")
                else:
                        print("Search is Un-sucessful:")
```

```
#sumavg.py
n=int(input("Enter How Many numbers u have:"))
if(n<=0):
        print("{} is invalid input:".format(n))
else:
        l=list()  # creating empty list
        for i in range(1,n+1):
                val=float(input("Enter {} value:".format(i)))
                l.append(val)
        else:
                print("-----------------------------------------")
                print("Content of list={}".format(l))
                print("-----------------------------------------")
                print("Sum={}".format(sum(l)))
                print("Avg={}".format(sum(l)/len(l) ))
```

```
#sumavg1.py
n=int(input("Enter How Many numbers u have:"))
if(n<=0):
        print("{} is invalid input:".format(n))
else:
        l=list()  # creating empty list
        for i in range(1,n+1):
                val=float(input("Enter {} value:".format(i)))
                l.append(val)
        else:
                s=0
                print("-----------------------------------------")
                print("Content of list={}".format(l))  # [10 -10  20  -20
30]
                print("-----------------------------------------")
                for val in l:
                        s=s+val
                else:
                        print("-------------------------------")
                        print("sum={}".format(s))
                        print("Avg={}".format(s/n))
```

for loop (or) for...else

```
Syntax-1
-------------
for varname in Iterable_Object:
       statement-1
       statement-2
       ----------------
       statement-n
----------------------------
Other Statements in program
---------------------------------
              (OR)
Syntax-2
-------------
for varname in Iterable_Object:
       statement-1
       statement-2
       ----------------
       statement-n
else:
       else Block of Statements
---------------------------------
Other Statements in program
---------------------------------
```

Explanation:

=>here 'for' and 'in' are the keywords

=>The execution process of for loop is that " Each element of Iterable-object kept in varname and executes Indentation Block of statements until all elements in iterable object are completed"

=>here writing 'else' block is optional.

=>After for loop excution, condition becomes false and PVM executes else block of statements(if we write else) and later executes Other statements in Program

break statement

=>break is a key word

=>The purpose of break statement is that "To terminate the execution of loop logically when certain condition is satisfied and PVM control comes of corresponding loop and executes other statements in the program".

=>Syntax:

```
-------------------
                      for var in Iterable_object:
                             ------------------------------
                             if (test cond):
                                    break
                             ------------------------------
                             ------------------------------
```

```
-----------------
=>Syntax:
------------------
                     while(Test Cond-1):
                            -----------------------------
                            if (test cond-2):
                                   break
                            -----------------------------
                            -----------------------------
===========================X===============================
#breakex1.py
s="PYTHON PROG"
for val in s:
       print("\t{}".format(val))
else:
       print("Line-6 i am from else") # here it is executed
print("--------------------------------------")

for val in s:
       if(val=="O"):
              break
       print("\t{}".format(val))
else:
       print("Line-14:-i am from else block") # here it is not executed
print("---------------------------------------")
```

```
#breakex1.py
lst=[10,20,30,40,50,60,-40,70,80]
for val in lst:
       print("\t{}".format(val))
else:
       print("line-6-i am from else part") # executed
       print("-------------------------------------")
#print the elements 10 20 30 40 50  60 only
for val in lst:
       if(val == -40):
              break
       else:
              print("\t{}".format(val))
else:
       print("line-15-i am from else part") # executed
print("--------------------------------------")
```

continue statement

=>continue is a keyword

=>continue statement is used for making the PVM to go to the top of the loop without executing the following statements which are written after continue statement for that current Iteration only.

=>continue statement to be used always inside of loops.

=>when we use continue statement else part of corresponding loop also executes provided loop condition becomes false.

```
-----------------
=>Syntax:-
----------------
                      for varname   in Iterable-object:
                           ------------------------------------------
                           if ( Test Cond):
                                continue
                           statement-1  # written after continue statement
                           statement-2
                           statement-n
                      -----------------------------------------
                      -----------------------------------------
===========================X===================================
#continueex1.py
s="PYTHON"
for val in s:
        print("\t{}".format(val))
print("----------------------------------")
#display      PYTON
for val in s:
        if(val=="H"):
                continue
        print("\t{}".format(val))
else:
        print("\nI am from else part:")
```

```
#continueex2.py
s="PYTHON"
#display      PYHN
for val in s:
        if(val=="T") or  (val=="O"):
                continue
        print("\t{}".format(val))
else:
        print("\nI am from else part:")
```

```
#continueex3.py
tpl=(10,20,30,40,50,60,70,80)
#display      PYHN
for val in tpl:
        if(val==20) or  (val==50) or (val==70):
                continue
        print("\t{}".format(val))
else:
        print("\nI am from else part:")
```

```
#continueex4.py
n=int(input("Enter How Many Numbes u have:"))
if(n<=0):
        print("{} is invalid input:".format(n))
else:
        lst=list()
        for i in range(1,n+1):
                value=float(input("Enter {} value: ".format(i)))
                lst.append(value)
        else:
                print("Content of list={}".format(lst)) # [12.3, 34.5, -
3.4, -5.6, 12.0]
                #get only Possitive Elements
                pslist=[]
                for val in lst:
                        if(val<=0):
                                continue
                        pslist.append(val)
                else:
                        print("Possitive Values:{}".format(pslist))
                        print("-----------------------------------------------
---------")
                nslist=[]
                for val in lst:
                        if(val>=0):
                                continue
                        nslist.append(val)
                else:
                        print("Negatuve Values:{}".format(nslist))
                        print("-----------------------------------------------
---------")
```

```
#primeno.py
n=int(input("Enter a number:"))  # n=5
if(n<=1):
        print("{} is invalid input:".format(n))
else:
        result="PRIME"
        for i in range(2,n):
                if(n%i==0):
                        result="NOT RIME"
                        break
        if(result=="PRIME"):
                print("{} is a Prime Number:".format(n))
        else:
                print("{} is a not Prime Number:".format(n))
```

```
#voterex1.py
age=int(input("Enter the age:"))
if(age>=18):
        print("Citizen is eligible to Vote:")
else:
        print("Citizen is not eligible to Vote:")
```

```
#voterex2.py
while(True):
        age=int(input("Enter the Correct age:"))
        if(age>=18)  and  ( age<=100):
                break

print("Citizen is eligible to Vote:")
```

```
                  =======================================
                       Nested (or) Inner Loops in Python
                  =======================================
```

=>The Process of defining one loop inside of another is called Nested / Inner Loop.

=>The Execution Process of Inner Loops is that "For Every value of Outer Loop inner loop executed many times".

```
------------------
=>Syntax1:
------------------
               for varname1 in Iterable_object1:  # Outer loop
                     -----------------------------------
                     for vaname2 in Iterbale_object2:  # Inner Loop
                             ---------------------------
                             ---------------------------
                     else:
                          --------------------------------
               else:
                      -----------------------------------



------------------
=>Syntax2:
------------------
               ----------------------------
               while(Test Cond1):  # outer loop
                      -----------------------
                      -----------------------
                      while(Test Cond2): # inner loop
                             ---------------------
                             ---------------------
                      else:
                            ----------------------
                else:
                      ----------------------------
Syntax-3
--------------
               for varname1 in Iterable_object1:  # Outer loop
                     -----------------------------------
                    while(Test Cond2):  # inner loop
                             ---------------------
                             ---------------------
                      else:
                            ----------------------
               else:
                      -----------------------------------

=>Syntax4:
------------------
               ----------------------------
               while(Test Cond1):   # outer loop
                      -----------------------
```

```
                       ------------------------
                       for varname in iterable_object:  # inner loop
                              ----------------------
                              ----------------------
                       else:
                             -----------------------
                 else:
                       -----------------------------
```

```
#innerliipex1.py
for i in range(1,6):
        print("Val of i (outer Loop)=",i)
        print("-------------------------------------")
        for j in range(1,4):
                print("Val of j (Inner Loop)=",j)
        else:
                print("I am out inner loop")
                print("-------------------------------------")
else:
        print("i am out of outer loop")

"""
E:\KVR-PYTHON-7AM\LOOPS>py innerliipex1.py
Val of i (outer Loop)= 1
-------------------------------------
Val of j (Inner Loop)= 1
Val of j (Inner Loop)= 2
Val of j (Inner Loop)= 3
I am out inner loop
-------------------------------------
Val of i (outer Loop)= 2
-------------------------------------
Val of j (Inner Loop)= 1
Val of j (Inner Loop)= 2
Val of j (Inner Loop)= 3
I am out inner loop
-------------------------------------
Val of i (outer Loop)= 3
-------------------------------------
Val of j (Inner Loop)= 1
Val of j (Inner Loop)= 2
Val of j (Inner Loop)= 3
I am out inner loop
-------------------------------------
Val of i (outer Loop)= 4
-------------------------------------
Val of j (Inner Loop)= 1
Val of j (Inner Loop)= 2
Val of j (Inner Loop)= 3
I am out inner loop
-------------------------------------
Val of i (outer Loop)= 5
-------------------------------------
Val of j (Inner Loop)= 1
Val of j (Inner Loop)= 2
Val of j (Inner Loop)= 3
I am out inner loop
```

```
-------------------------------------
i am out of outer loop"""
```

```
#innerloopex2.py
i=1
while(i<6):
        print("Val of i (outer Loop)=",i)
        print("-------------------------------------")
        j=1
        while(j<4):
                print("Val of j (Inner Loop)=",j)
                j=j+1
        else:
                print("I am out of inner loop")
                i=i+1
                print("-------------------------------------")
else:
        print("i am out of outer loop")

"""
E:\KVR-PYTHON-7AM\LOOPS>py innerloopex2.py
Val of i (outer Loop)= 1
-------------------------------------
Val of j (Inner Loop)= 1
Val of j (Inner Loop)= 2
Val of j (Inner Loop)= 3
I am out of inner loop
-------------------------------------
Val of i (outer Loop)= 2
-------------------------------------
Val of j (Inner Loop)= 1
Val of j (Inner Loop)= 2
Val of j (Inner Loop)= 3
I am out of inner loop
-------------------------------------
Val of i (outer Loop)= 3
-------------------------------------
Val of j (Inner Loop)= 1
Val of j (Inner Loop)= 2
Val of j (Inner Loop)= 3
I am out of inner loop
-------------------------------------
Val of i (outer Loop)= 4
-------------------------------------
Val of j (Inner Loop)= 1
Val of j (Inner Loop)= 2
Val of j (Inner Loop)= 3
I am out of inner loop
-------------------------------------
Val of i (outer Loop)= 5
-------------------------------------
Val of j (Inner Loop)= 1
Val of j (Inner Loop)= 2
Val of j (Inner Loop)= 3
I am out of inner loop
-------------------------------------
i am out of outer loop"""
```

```
#innerloopex3.py
for i in range(5,0,-1):
        print("val of i (outer loop)=",i)
        print("----------------------------------")
        j=3
        while(j>0):
                print("Val of j=",j)
                j=j-1
        else:
                print("out of inner while loop")
                print("----------------------------------")
else:
        print("Out of outer for loop")
```

```
#innerloopex4.py
i=1
while(i<6):
        print("Val of i (outer Loop)=",i)
        print("-------------------------------------")
        for j in range(3,0,-1):
                print("Val of j(Inner Loop)=",j)
        else:
                print("I am out of inner loop")
                i=i+1
                print("-------------------------------------")
else:
        print("i am out of outer loop")

"""
Val of i (outer Loop)= 1
-------------------------------------
Val of j(Inner Loop)= 3
Val of j(Inner Loop)= 2
Val of j(Inner Loop)= 1
I am out of inner loop
-------------------------------------
Val of i (outer Loop)= 2
-------------------------------------
Val of j(Inner Loop)= 3
Val of j(Inner Loop)= 2
Val of j(Inner Loop)= 1
I am out of inner loop
-------------------------------------
Val of i (outer Loop)= 3
-------------------------------------
Val of j(Inner Loop)= 3
Val of j(Inner Loop)= 2
Val of j(Inner Loop)= 1
I am out of inner loop
-------------------------------------
Val of i (outer Loop)= 4
-------------------------------------
Val of j(Inner Loop)= 3
Val of j(Inner Loop)= 2
Val of j(Inner Loop)= 1
I am out of inner loop
-------------------------------------
Val of i (outer Loop)= 5
```

```
-------------------------------------
Val of j(Inner Loop)= 3
Val of j(Inner Loop)= 2
Val of j(Inner Loop)= 1
I am out of inner loop
-------------------------------------
i am out of outer loop
"""
```

```
#innerloopex5.py
lst=[-45,3,14,19,9,7,0,8]
for n in lst:  # outer loop supplies values from lst
        if(n<=0):
                print("{} is invalid input".format(n))
        else:
                print("--------------------------------")
                print("Mul Table of {}".format(n))
                print("--------------------------------")
                for i in range(1,11):  # inner loop generates mul table for
the val supplied by Outer loop
                        print("\t{} x {}={}".format(n,i,n*i))
                else:
                        print("--------------------------------")
```

```
#innerloopex6.py
#accept list of values
n=int(input("Enter How Many Values u have:"))
if(n<=0):
        print("{} is invalid input:".format(n))
else:
        lst=list()
        for i in range(1,n+1):
                val=int(input("Enter {} value:".format(i)))
                lst.append(val)
        else:
                print("---------------------------------------")
                print("Content of List=",lst)  # [1, 14, 12, 13, 17]
                print("---------------------------------------")
                pnlst=[]
                i=0
                while(i<len(lst)):
                        n=lst[i]
                        if(n<=1):
                                print("{} is invalid Input:".format(n))
                        else:
                                result="Prime"
                                for j in range(2,n):
                                        if(n%j==0):
                                                result="Not Prime"
                                                break
                                if(result=="Prime"):
                                        pnlst.append(n)
                        i=i+1
                else:
                        print("Prime Numbers List={}".format(pnlst))
```

Introduction to Functions

=>The Purpose of Function Concept in any Programming language is that "To Perform Certain Operation and Provides Code Re-Usability".

=>Def. of Function :   A part of main program is called Function.
(OR)
Sub Program of main program is called Function.

=>Types of Functions.

=>We have two types of Functions. They are
a) Pre-defined (or) Built-in Functions.
b) Programmer / User / Custom Defined Functions.

=>Pre-defined (or) Built-in Functions are those which are already developed and available in Python API and They re-used by Python Programmers for dealing with Unversal Purpose.
Examples: int()   float(), append(), print(), id() type()....etc

=>Programmer / User / Custom Defined Functions are developed by Python Programmers and re-used by other Python programmers and they are meant for performing common operations.

Examples: deposit()   withdraw()   balenq()   genotp()...etc

Maths----Functions

Q) Consider   f (x,y) = x+y   find 1) f(2,3) 2) f(5,-6)

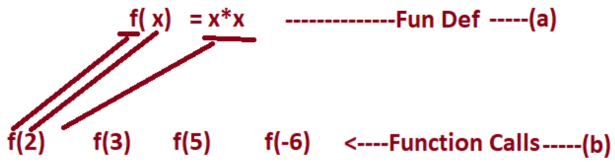
Sol:- $\boxed{\overset{(1)}{f}\,(\overset{(2)}{x, y}) = x \overset{(3)}{+} y}$ **<--------Function Definition**

1) f (2 , 3)<---Function Call
= 5

2) f (5, -6)
= -1

3) g (4,5)----NameError
Function call---req. Function Definition

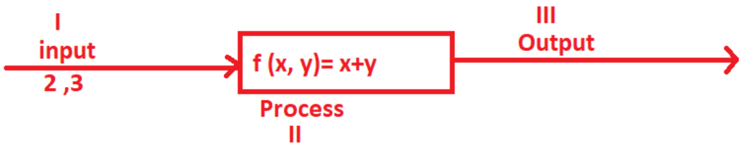
Parts of Functions--->



f(x) = x*x --------------Fun Def -----(a)

f(2) f(3) f(5) f(-6) <----Function Calls-----(b)

Phases in Functions

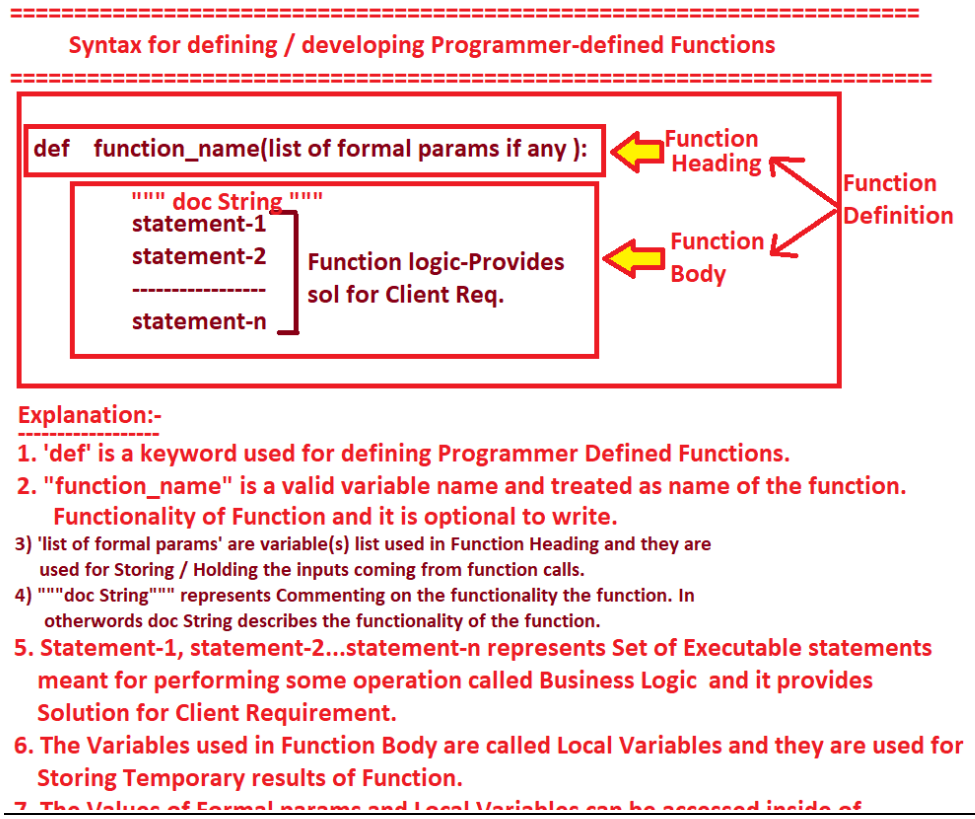


I
input
2 ,3

f (x, y)= x+y

Process
II

III
Output



==

Syntax for defining / developing Programmer-defined Functions

==

```
def   function_name(list of formal params if any ):
        """ doc String """
        statement-1
        statement-2
        ----------------
        statement-n
```

Explanation:-

----

1. 'def' is a keyword used for defining Programmer Defined Functions.
2. "function_name" is a valid variable name and treated as name of the function.
   Functionality of Function and it is optional to write.

3) 'list of formal params' are variable(s) list used in Function Heading and they are used for Storing / Holding the inputs coming from function calls.

4) """doc String""" represents Commenting on the functionality the function. In otherwords doc String describes the functionality of the function.

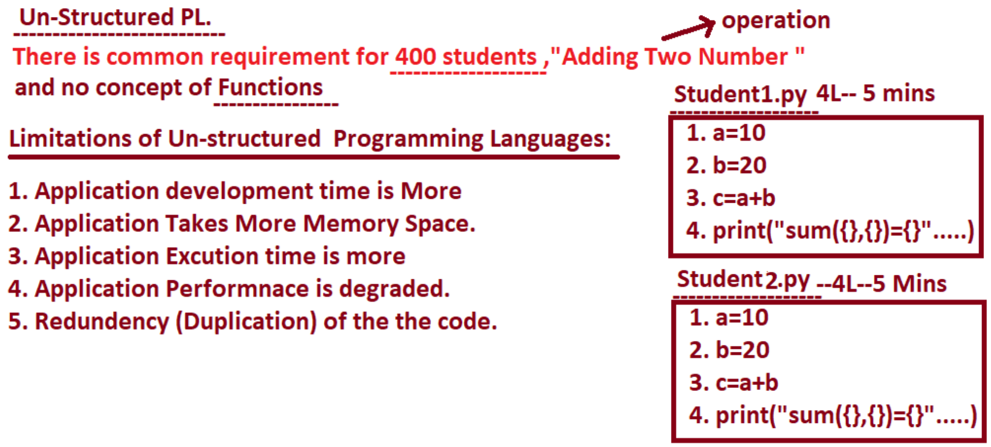
5. Statement-1, statement-2...statement-n represents Set of Executable statements meant for performing some operation called Business Logic and it provides Solution for Client Requirement.
6. The Variables used in Function Body are called Local Variables and they are used for Storing Temporary results of Function.
7. The Values of Formal params and Local Variables can be accessed inside of

```
                    ========================================
                               Types of Languages
                    ========================================
=>In the context of Functions, we can classify the Programming languages into
two types. They are
                    a) Un-Structured Programming Languages
                    b) Structured Programming Languages
-----------------------------------------------------------------------------
---------
a) Un-Structured Programming Languages:
---------------------------------------------------------------
=>In Un-Structured Programming Languages, we don't have the concept of
Functions.
Example:-    GW-BASIC

=>Since Un- Un-Structured Programming Languages does not contain Functions
concept and It has the following Limitations.

                    1. Application development time is More
                    2. Application Takes More Memory Space.
                    3. Application Excution time is more
                    4. Application Performnace is degraded.
                    5. Redundency (Duplication) of the the code.
-----------------------------------------------------------------------------
---------
b) Structured Programming Languages:
---------------------------------------------------------------
=>In Structured Programming Languages, we  have the concept of Functions.
Example:-    C,CPP,JAVA,PYTHON,.NET....etc
```

=>Since Structured Programming Languages contains Functions concept and It has the following Adavantages.

1. Application development time is Less
2. Application Takes Less Memory Space.
3. Application Excution time is Less.
4. Application Performnace is Enhanced(Improved).
5. Redundency (Duplication) of the the Minimized.

=========================X===================================



Un-Structured PL.

There is common requirement for 400 students ,"Adding Two Number " and no concept of Functions

operation

Limitations of Un-structured Programming Languages:

1. Application development time is More
2. Application Takes More Memory Space.
3. Application Excution time is more
4. Application Performnace is degraded.
5. Redundency (Duplication) of the the code.

Student1.py 4L-- 5 mins

```
1. a=10
2. b=20
3. c=a+b
4. print("sum({},{})={}".....)
```

Student2.py --4L--5 Mins

```
1. a=10
2. b=20
3. c=a+b
4. print("sum({},{})={}".....)
```

Phases in Functions
----

a) Every Function takes INPUT
b) Every Function PROCESS the input
c) Every Function gives OUTPUT / RESULT

---X-----------------------------------

Approaches to develop a function for problem solving

Approach1:

INPUT:- Takes Inputs from Function Calls (Outside)
PROCESS: Proces the input inside of Function Body(Inside)
OUTPUT:- Function gives result to the Function call(Outside)

```
#approach1.py
def   sumop(a,b):  # here 'a' and 'b' are called Formal Params
        c=a+b  # here 'c' is called local variable
        return c
```

```
#main program
x=float(input("Enter First Value:"))
y=float(input("Enter Second Value:"))
res=sumop(x,y)  # Function Call
print("sum({},{})={}".format(x,y,res))
-----------------------------------------------------------------------------
Approach2:
--------------------
INPUT:-  Takes Inputs  in Function Body (Inside)
PROCESS:  Proces the input inside of Function Body(Inside)
OUTPUT:- Function gives result within Function Body(Inside)

#approach2.py
def   sumop():
        a=float(input("Enter First Value:"))
        b=float(input("Enter Second Value:")) # INPUT
        c=a+b  # PROCESS
        print("\nsum({},{})={}".format(a,b,c)) # OUTPUT

#main program
sumop() # Function Call
-----------------------------------------------------------------------------
Approach3:
-----------------------------------------------------------------------------
INPUT:-  Takes Inputs  in Function Body (Inside)
PROCESS:  Proces the input inside of Function Body(Inside)
OUTPUT:- Function gives result to the Function Call(outside)

#approach3.py
def   sumop():
        a=float(input("Enter First Value:"))
        b=float(input("Enter Second Value:"))
        c=a+b
        return("sum {} and {}={}".format(a,b,c))

#main program
result=sumop()
print(result)
---------------------
Approach4:
--------------------
INPUT:-  Takes Inputs  from Function Calls (outside)
PROCESS:  Proces the input inside of Function Body(Inside)
OUTPUT:- Function gives result within Function Body(Inside)

#approach4.py
def   sumop(a,b):
        c=a+b
        print("sum({},{})={}".format(a,b,c))


#main program
a=float(input("Enter First Value:"))
b=float(input("Enter Second Value:"))
sumop(a,b)
==============================X=============================
#approach1.py
With formal parameters and with return value
def   sumop(a,b):  # here 'a' and 'b' are called Formal Params
        c=a+b  # here 'c' is called local variable
        return c

#main program
x=float(input("Enter First Value:"))
y=float(input("Enter Second Value:"))
res=sumop(x,y)  # Function Call
print("sum({},{})={}".format(x,y,res))
```

```
#approach2.py
Without formal parameters and without return value
def   sumop():
        a=float(input("Enter First Value:"))
        b=float(input("Enter Second Value:")) # INPUT
        c=a+b  # PROCESS
        print("\nsum({},{})={}".format(a,b,c)) # OUTPUT

#main program
sumop() # Function Call
```

```
#approach3.py
Without formal parameters and with return values
def   sumop():
        a=float(input("Enter First Value:"))
        b=float(input("Enter Second Value:"))
        c=a+b
        return a,b,c  # In python, return can return one or more number of
values.

#main program
x,y,z=sumop() # Multi Line assignment statement.
print("sum of {} and {}={}".format(x,y,z))
print("=============OR===============")
res=sumop()  # here res is variable of type <class, 'tuple'> and it can hold
many values returned by return statement.
print("sum of {} and {}={}".format(res[0],res[1],res[2] ) )
print("sum of {} and {}={}".format(a,b,c ) )
```

```
#approach4.py
With formal parameters and without return value
def   sumop(a,b):
        c=a+b
        print("sum({},{})={}".format(a,b,c))


#main program
a=float(input("Enter First Value:"))
b=float(input("Enter Second Value:"))
sumop(a,b)
```

```
#sqrootex1.py
#Approach-1
def    sqroot(n):
        res=n**0.5
        return res

#main program
```

```
n=int(input("Enter a number:"))
result=sqroot(n)   # function call
print("sqrt({})={}".format(n,result))
```

```
#sqrootex2.py
#Approach-2
def  sqroot():
        n=int(input("Enter a number:"))
        res=n**0.5
        print("sqrt({})={}".format(n,res))

#main program
sqroot()  # function call
```

```
#sqrootex3.py
#Approach-3
def   sqroot():
        n=int(input("Enter a number:"))
        res=n**0.5
        return n,res

#main program
n,res=sqroot()
print("sqrt({})={}".format(n,res))
print("===========OR===========")
result=sqroot()
print("sqrt({})={}".format(result[0],result[1]))
```

```
#sqrootex4.py
#Approach-4
def   sqroot(n):
        res=n**0.5
        print("sqrt({})={}".format(n,res))

#main program
n=int(input("Enter a number:"))
sqroot(n)   # function call
```

```
#multable.py
def   table(n):
        if(n<=0):
                print("{} is invalid input:".format(n))
        else:
                print("-"*50)
                print("Mul table for {}".format(n))
                print("-"*50)
                for i in range(1,11):
                        print("\t{} x {} = {}".format(n,i,n*i))
                else:
                        print("-"*50)
#main program
x=int(input("Enter a number:"))
table(x)  # Function Call
```

```
#collectionsvalues.py
def  disp(obj):
        print("type of obj=",type(obj))
        for val in obj:
                print("\t{}".format(val))

def  show(obj):
        for k,v in obj.items():
                print("\t{}\t{}".format(k,v))

#main program
print("List of Values:")
lst=[10,23,45,4,56,123,-45,-6]
disp(lst)
print("set of values")
s1={23,"Rossum",56.78,True}
disp(s1)
print("Dict Values")
d1={10:"Python",20:"Java",30:"DS",40:"ML"}
show(d1)
```

```
#sumavg.py
def  readvalues():
        lst=[]
        print("Enter how many values u have:")
        n=int(input())
        for i in range(1,n+1):
                val=float(input("Enter {} value:".format(i)))
                lst.append(val)
        return lst

def  computesumavg(lst):
        s=0
        print("-----------------------------------")
        for val in lst:
                print("\t{} ".format(val))
                s=s+val
        else:
                print("-----------------------------------")
                print("\tsum={}".format(s))
                print("\tAvg={}".format(s/len(lst)))
                print("-----------------------------------")


#main program
lst=readvalues() # function call
computesumavg(lst)
```

===

Arguments and Parameters

===

=>Arguments and Parameters are representing Variable Names.

=>Arguments are the variables which are used in Function Calls. Arguments are also called Actual Parameters / arguments.

=>Parameters are the variables, which are used two places in Function Definition. The Parameters used in Function Heading are called Formal Parameters and the Parameters used in Function Body are called Local Parameters / Variables.

=>All the values Arguments are passing Parameters and it known as Agrument / Parameter Passing Mechanisms.

Agrument (or) Parameter Passing Mechanisms.

=>Based on the values of arguments passing to Parameters , The mechanism of values passing are classfied into 5 types. They are

1) Possitional Parameters / Aguments (default)
2) Default Parameters / Aguments
3) Keyword Parameters / Aguments
4) Variable length Parameters / Aguments
5) Keyword Variable length Parameters / Aguments

1) Possitional Arguments (or) Parameters

=>The Concept of Possitional Parameters (or) arguments says that "The Number of Arguments (Actual Parameters) must be equal to the number of formal paraemeters ".

=>This Parameter mechanism also recommends Order of Parameters for Higher accuracy.

=>Python Programming Environment follows by default Possitional Parameters.

Syntax for Function Definition :

```
       def    functionname(param1,param2.....param-n):
              -------------------------------------------------
              -------------------------------------------------
```

Syntax for Function Call:

```
              functionname(arg1,arg2....arg-n)
```

=>Here the values of arg1,arg2...arg-n are passing to param-1,param-2..param-n respectively.

```
#posparamex1.py
def   dispstuddet(stno,sname,marks):
       print("\t{} \t{}\t{}".format(stno,sname,marks))

#main program
print("----------------------------------------")
print("\tStudent Information:")
print("----------------------------------------")
print("\tstno\tName\tMarks")
print("----------------------------------------")
dispstuddet(10,"RS",34.56)
dispstuddet(20,"JG",24.56)
dispstuddet(30,"DR",84.56)
print("----------------------------------------")
```

2) Default Parameters (or) arguments

=>When there is a Common Value for family of Function Calls then Such type of Common Value(s) must be taken as default parameter with common value (But not recommended to pass by using Posstional Parameters)

```
Syntax: for Function Definition with Default Parameters
---------------------------------------------------------------------------
--------------
def   functionname(param1,param2,....param-n-1=Val1, Param-n=Val2):
          -----------------------------------------------------------------
--
         ------------------------------------------------------------------
-
```

Here param-n-1 and param-n are called "default Parameters"
   and param1,param-2... are called "Possitional paramsters"

Rule-: When we use default parameters in the function definition, They must be used as last Parameter(s) otherwise we get Error(SyntaxError: non-default argument follows default argument).

```
#defualtparamex1.py
def   dispstuddet(stno,sname,marks,crs="PYTHON",cnt="INDIA"):
       print("\t{} \t{}\t{}\t{}\t{}".format(stno,sname,marks,crs,cnt))

#main program
print("------------------------------------------------------------------")
print("\tStudent Information:")
print("-----------------------------------------------------------------")
print("\tstno\tName\t\tMarks\tCourse\tCountry")
print("------------------------------------------------------------------")
dispstuddet(10,"Chaitanya",34.56)
dispstuddet(20,"Manasa    ",24.56)
dispstuddet(30,"Minakshi",84.56)
dispstuddet(40,"Adarsh  ",14.56,"Java")
dispstuddet(50,"Rossum   ",11.56)
dispstuddet(60,"Ritche     ",14.56,"C","USA")
dispstuddet(70,"Travis      ",17.56,"DS")
print("-----------------------------------------------------------------")
```

```
#defualtparamex2.py
def  area(r,PI=3.14):
       ac=PI*r**2
       print("Area of Circle={}".format(ac))

def  peri(PI=3.14):
       r=float(input("Enter Radius for cal peri:"))
       pc=2*PI*r
       print("Peri. of Circle={}".format(pc))

#main program
r=float(input("Enter Radius for cal Area:"))
area(r)
print("----------------------------------------")
peri()
```

4) Variables Length Parameters (or) arguments

=>When we have familiy of multiple function calls with Variable number of values / arguments then with normal python programming, we must define mutiple function defintions. This process leads to more development time. To overcome this process, we must use the concept of Variable length Parameters .

=>To Impelement, Variable length Parameters concept, we must define single Function Definition and takes a formal Parameter preceded with a symbol called astrik (* param) and the formal parameter with astrik symbol is called Variable length Parameters and whose purpose is to hold / store any number of values coming from similar function calls and whose type is <class, 'tuple'>.

Syntax for function definition with Variables Length Parameters:

```
        def   functionname(list of formal params,  *param) :
                ---------------------------------------------------
                ---------------------------------------------------
```

=>Here *param is called Variable Length parameter and it can hold any number of argument values (or) variable number of argument values and *param type is <class,'tuple'>

=>Rule:- The *param must always written at last part of Function Heading and it must be only one (but not multiple)

=>Rule:- When we use Variable length and default parameters in function Heading, we use default parameter as last and before we use variable length parameter and in function calls, we should not use default parameter as Key word argument bcoz Variable number of values are treated as Posstional Argument Value(s)

```
#varlenex1.py----This program will not execute
def disp(x,y,z):
        print(x,y,z)

def   disp(x):
        print(x)

def  disp(x,y):
        print(x,y)
#main program
disp(10)  # function call-1
disp(10,20)  # function call-2
disp("RS","DR","TR")  # function call-3
```

```
#varlenex2.py----This program will  execute
def disp(x,y,z):  # Function Definition
        print(x,y,z)

disp("RS","DR","TR")  # function call-1

def   disp(x):
        print(x)

disp(10)  # function call-2
```

```
def  disp(x,y):
       print(x,y)

disp(10,20)  # function call-3
```

```
#varlenex4.py
def findsum(name, *vals,crs="PYTHON"):
        s=0
        print("-"*40)
        print("Hi, {} ur crs={}".format(name,crs))
        for val in vals:
                print("{}".format(val),end=" ")
                s=s+val
        else:
                print("Sum=",s)
                print()
#main program
findsum("RS",10,20)
findsum("DR",10,20,30)
findsum("MC",10,20,30,40)
findsum("TR",10,20,30,40,50)
findsum("JG",10,20,30,40,50,60)
findsum("RS1")
#findsum(10,20,30,40,50,60,"JG") error
#findsum(10,20,30,40,50,60,name="JG")  error
findsum("JG1",10,20,30,40,50,60,crs="Java")
#findsum("JG1",crs="Java",10,20,30,40,50,60) SyntaxError: positional argument
follows keyword argument
```

==

3) Keyword Parameters (or) arguments

==

=>In some of the circumstances, we know the function name and formal parameter names and we don't know the order of formal Parameter names and to pass the data / values accurately we must use the concept of Keyword Parameters (or) arguments.
=>The implementation of Keyword Parameters (or) arguments says that all the formal parameter names used as arguments in Function call(s) as keys.

```
Syntax for function definition:-
-------------------------------------------------
def    functionname(param1,param2...param-n):
         ---------------------------------------------
         ---------------------------------------------

Syntax for function call:-
-------------------------------------------------
       functionname(param-n=val-n,param1=val1,param-n-1=val-n-1,......)
```

Here param-n=val-n,param1=val1,param-n-1=val-n-1,...... are called Keywords arguments

==========================X===============================

```
#kwdargsex1.py
def dispempinfo(eno,ename,sal,dsg):
       print("\t{}\t{}\t{}\t{}".format(eno,ename,sal,dsg))




#main program
print("-"*50)
print("\tEmpno\tName\tSal\tDesg")
print("-"*50)
dispempinfo(111,"RS",5.6,"SE")
dispempinfo(112,"DR",dsg="TL",sal=6.7)
dispempinfo(sal=3.4,dsg="SE",eno=113,ename="TR")
dispempinfo(114, sal=4.4,dsg="TR",ename="JG")
#dispempinfo(sal=2.4,dsg="TR",ename="MC",115)  SyntaxError: positional
argument follows keyword argument
print("-"*50)
```

```
#kwdargsex2.py
def dispempinfo(eno,ename,sal,dsg,cnt="INDIA"):
       print("\t{}\t{}\t{}\t{}\t{}".format(eno,ename,sal,dsg,cnt))

#main program
print("-"*50)
print("\tEmpno\tName\tSal\tDesg\country")
print("-"*50)
dispempinfo(111,"RS",5.6,"SE")
dispempinfo(112,"DR",dsg="TL",sal=6.7)
dispempinfo(cnt="USA", sal=3.4,dsg="SE",eno=113,ename="TR")
dispempinfo(114, sal=4.4,dsg="TR",ename="JG")
#dispempinfo(sal=2.4,dsg="TR",ename="MC",115)  SyntaxError: positional
argument follows keyword argument
#dispempinfo(114, "ST",sal=4.4,dsg="TR","GER") SyntaxError: positional
argument follows keyword argument
dispempinfo(114, "ST",sal=4.4,dsg="TR",cnt="GER")
print("-"*50)
```

```
               ==================================================
                       Keyword Variable length Parameters (or) Aguments
               ==================================================
```

=>When we have familiy of multiple function calls with Keyword Variable length number of values / arguments then with normal python programming, we must define mutiple function defintions. This process leads to more development time. To overcome this process, we must use the concept of Keyword Variable length Parameters .
=>To Impelement, Keyword Variable length Parameters concept, we must define single Function Definition and takes a formal Parameter preceded with a symbol called double astrik (** param) and the formal parameter with double astrik symbol is called Keyword Variable length Parameter and whose purpose is to hold / store any number of keyword variable length values coming from similar function calls and whose type is <class, 'dict'>.

```
-----------------------------------------------------------------------------
Syntax for function definition with Keyword  Variables Length Parameters:
-----------------------------------------------------------------------------
       def   functionname(list of formal params,  **param) :
                    --------------------------------------------------
```

=>Here **param is called Keyword Variable Length parameter and it can hold any number of keyword variable length values / argument values and **param type is <class,'dict'>

=>Rule:- The **param must always written at last part of Function Heading and it must be only one (but not multiple)

```
#kwdvarlenex1.py
def  dispinfo(**x):  # here  **x is called kwd var length parameter--
<class,dict>
        print("-"*40)
        for k,v in x.items():
                print("\t{}\t{}".format(k,v))
        else:
                print("-"*40)

#main program
dispinfo(rname="Rossum")
dispinfo(sno=10,sname="RS")
dispinfo(eno=20,ename="RT",sal=4.6)
dispinfo(idno=111,name="Sandeep",hobby1="Reading",hobby2="practcing")
```

```
#kwdvarlenex1.py
def totalmarks(sname,cls, **infor):
        print("-"*40)
        print("Student Name:{}".format(sname))
        print("Student Studying in :{}".format(cls))
        print("-"*40)
        print("\tSubjects\tMarks")
        print("-"*40)
        totmarks=0
        for subj,marks in infor.items():
                print("\t{}\t\t{}".format(subj,marks))
                totmarks=totmarks+marks
        else:
                print("-"*40)
                print("\tTotal Marks={}".format(totmarks))




#main program
totalmarks("RS","X",Eng=67,Tel=66,Sci=88,maths=99,soc=88)
totalmarks("DR","XII",Phy=56,Che=58,Mathematics=74)
totalmarks("TR","B.Tech",C=60,Python=60)
totalmarks("MCK","Research")
```

==

Local and global Variables

==

=>Local Variables are those, which are used in Function Body for storing temporary results.
=>Local Variables can be accessed in corresponding Function Body only but not possible to access in the context other function definitions.

=>Global Variables are those, which are defined before all the function definitions.

=>The main purpose of Global Variables is that To store common Values for multiple different Functions.

=>Global Variable Values can be used in all functions bcoz they are common for all functions.

```
----------------------
Examples:
---------------------
#localglobalex1.py
#lang="PYTHON PROG"  # global variable
def   learncrs1():
        crs1="DS"  # local variable
        print("To implement '{}' , we a programming lang
'{}'".format(crs1,lang))
        #print(crs2,crs3) not possible to access
def   learncrs2():
        crs2="ML"  # local variable
        print("To implement '{}' , we a programming lang
'{}'".format(crs2,lang))
        #print(crs1,crs3) not possible to access
def   learncrs3():
        crs3="DL"  # local variable
        print("To implement '{}' , we a programming lang
'{}'".format(crs3,lang))
        #print(crs1,crs2) not possible to access
#main program
lang="PYTHON PROG"  # global variable
learncrs1()
learncrs2()
learncrs3()
-------------------------------------------------------------------------------
E:\KVR-PYTHON-7AM\FUNCTIONS>py localglobalex1.py
To implement 'DS' , we a programming lang 'PYTHON PROG'
To implement 'ML' , we a programming lang 'PYTHON PROG'
To implement 'DL' , we a programming lang 'PYTHON PROG'
```

```
#localglobalex1.py
#lang="PYTHON PROG"  # global variable
city="HYD"
def   learncrs1():
        crs1="DS"  # local variable
        print("To implement '{}' , we a programming lang
'{}'".format(crs1,lang))
        print(city)
        #print(crs2,crs3) not possible to access
def   learncrs2():
        crs2="ML"  # local variable
        print("To implement '{}' , we a programming lang
'{}'".format(crs2,lang))
        print(city)
        #print(crs1,crs3) not possible to access
def   learncrs3():
        crs3="DL"  # local variable
        print("To implement '{}' , we a programming lang
'{}'".format(crs3,lang))
        print(city)
        #print(crs1,crs2) not possible to access
```

```
#main program
lang="PYTHON PROG"  # global variable
learncrs1()
learncrs2()
learncrs3()
```

```
               ======================================
                       global key word
               ======================================
```

=>When we want MODIFY the GLOBAL VARIABLE values in side of function defintion then global variable names must be preceded with global keyword otherwise we get "UnboundLocalError: local variable names referenced before assignment"

```
Syntax:
-----------
        var1=val1
        var2=val2
        var-n=val-n    #  var1,var2...var-n are called global variable names.
        ------------------
        def   fun1():
               ------------------------
               global var1,var2...var-n
               # Modify var1,var2....var-n
            --------------------------
        def   fun2():
              ------------------------
                global var1,var2...var-n
             # Modify var1,var2....var-n
            --------------------------
```

```
#globalkwdex1.py
a=10
b=20   # here 'a' and 'b' are called global variables
def   operation1():
        d=a+b+c  # here 'd' is called local Variable
        print("sum={}".format(d))
def   operation2():
        d=a-b-c   # here 'd' is called local Variable
        print("sub={}".format(d))

#main program
c=30  # global variable
operation1()
operation2()
```

```
#globalkwdex2.py
a=10  # global variable
def  update1():
        print("i am in update1()")
        global a
        a=a+1
        print("Val of a in update1()=",a)
def  update2():
        print("i am in update2()")
        global a
        a=a*2
        print("Val of a in update2()=",a)
```

```
#main program
print("Val of a a in main program before updat1()={}".format(a)) # 10
update1()
print("Val of a a in main program after update1()={}".format(a)) # 11
update2()
print("Val of a a in main program after update2()={}".format(a)) # 22
```

```
#globalkwdex3.py
a=10
b=20   # here 'a' and 'b' are called global variable.
def  modifyvalues():
        global a,b  # refering global Variable Values 'a' and 'b'
        a=a+1
        b=b+1
        print("val of a in modifyvalues()={}".format(a))
        print("val of b in modifyvalues()={}".format(b))

#main program
print("Val of a before modifyvalues()={}".format(a))
print("Val of b before modifyvalues()={}".format(b))
modifyvalues()
print("Val of a after modifyvalues()={}".format(a))
print("Val of b after modifyvalues()={}".format(b))
```