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

**II. Sequence Category Data Types**

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

=>The purpose of Sequence Category Data Types is that "To store Sequence of Values".

=>We have 4 Data Types in Sequence Category. They are

1. str

2. bytes

3. bytearray

4. range

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

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

**1. str**

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

**Index**

----------------

=>What is str

=>Definition of str

=>Notations of str

=>Types of strs

=>Syntax for storing str data

=>Memory Management of str data

a) +Ve Indexing

b) -Ve Indexing

=>Operations on str data

a) Indexing

b) Slicing Operation

=>Programming Examples

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

**Properties**

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

=>'str' is one of the pre-defined class and treated as **Sequence Data Type.**

=>The purpose of str data type is that "To store String data or text data or Alphanumeric data or numeric data or special symbols within double Quotes or single quotes or tripple double quotes and tripple single quotes. "

------------------------

**Def. of str:**

-----------------------

=>str is a collection of Characters or Alphanumeric data or numeric data or any type of data enclosed within double Quotes or single quotes or tripple double quotes and tripple single quotes. "

----------------------------

**Types of Str data**

-----------------------------

=>In Python Programming, we have two types of Str Data. They are

1. Single Line String Data

2. Multi Line String Data

-----------------------------------------

**1. Single Line String Data:**

-----------------------------------------

=>Syntax1:- varname=" Single Line String Data "

(OR)

=>Syntax2:- varname=' Single Line String Data '

=>With the help double Quotes ("") and single Quotes (' ') we can store single line str data only but not possible to store multi line string data.

---------------------------------------------------------------------------------------------------------------------

-----------------------------------------

**2. Multi Line String Data:**

-----------------------------------------

=>Syntax1:- varname=""" String Data1

String Data2

------------------

String data-n """

(OR)

=>Syntax2:- varname=' ' ' String Data1

String Data2

------------------

String data-n ' ' '

=>With the help tripple double Quotes ("""""") and Tripple single Quotes (' ' ' ' ' ') we can store single line str data and multi line string data.

------------------------------------------------------------------------------------------------------------------------------------------

Examples:

--------------------------------------------------------------------------------------------------------------------------------------

>>> s1="Python"

>>> print(s1,type(s1))---------------------------Python <class 'str'>

>>> s2='Python'

>>> print(s2,type(s2))--------------------------Python <class 'str'>

>>> s3="A"

>>> print(s3,type(s3))--------------------------A <class 'str'>

>>> s4='A'

>>> print(s4,type(s4))-------------------------A <class 'str'>

>>> s1="123456"

>>> print(s1,type(s1))-------------------------123456 <class 'str'>

>>> s2="Python3.11"

>>> print(s2,type(s2))------------------------Python3.11 <class 'str'>

>>> s3="123$456\_abc"

>>> print(s3,type(s3))------------------------123$456\_abc <class 'str'>

>>> s4="@#$%^&8912"

>>> print(s4,type(s4))-------------------------@#$%^&8912 <class 'str'>

>>> s1="Python Programming"

>>> print(s1,type(s1))-----------------------Python Programming <class 'str'>

------------------------------------------

>>> addr1="Guido Van Rossum---------- SyntaxError: unterminated string literal (detected at line 1)

>>> addr1='Guido Van Rossum---------- SyntaxError: unterminated string literal (detected at line 1)

---------------------------------------------------------------------

>>> addr1="""Guido Van Rossum

... FNO:3-4, Hill Side

... Python Software Foundation

... Nether Lands-56 """

>>> print(addr1,type(addr1))

Guido Van Rossum

FNO:3-4, Hill Side

Python Software Foundation

Nether Lands-56 <class 'str'>

-------------------------------------------------------------------------------------------

>>> addr2= ' ' ' Travis Oliphant

... HNO:12-34, Sea Side

... Numpy Organization

... Nether lands-58 ' ' '

>>> print(addr2,type(addr2))

Travis Oliphant

HNO:12-34, Sea Side

Numpy Organization

Nether lands-58 <class 'str'>

-----------------------------------------------------------------

>>> s1="""Python Programming"""

>>> print(s1,type(s1))------------Python Programming <class 'str'>

>>> s1='''Python Programming'''

>>> print(s1,type(s1))-------------------Python Programming <class 'str'>

>>> s2="""A"""

>>> print(s2,type(s2))------------------A <class 'str'>

>>> s2='''A'''

>>> print(s2,type(s2))---------------A <class 'str'>

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

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

**Operations on str data---Most Imp**  ====================================================================

=>On str object Data , we can perform Two types of Operations. They are

1. Indexing

2. Slicing

------------------------------------------------------------------------------------------------------------------------------------------

**1. Indexing Operations**

------------------------------------------------------------------------------------------------------------------------------------------

=>The purpose of Indexing Operation is that "To get OR Extract One Value at a time from str object".

=>Syntax: strobj [ Index ]

=>Here strobj is an object of type <class,'str'>

=>Here Index Represents either +Ve Index or -Ve Index

=>If we enter Valid +ve or -ve Index then PVM gets Corresponding Indexed Value from strobj.

=>If we enter InValid +ve or -ve Index then PVM gives IndexError

--------------------------------------

**Examples**

--------------------------------------

>>> s="PYTHON"

>>> print(s[0])-------------------P

>>> print(s[2])-------------------T

>>> print(s[3])-------------------H

>>> print(s[1])-------------------Y

>>> print(s[4])-------------------O

>>> print(s[5])-------------------N

---------------------------------

>>> print(s[-1])------------------N

>>> print(s[-2])------------------O

>>> print(s[-3])------------------H

>>> print(s[-4])------------------T

>>> print(s[-5])------------------Y

>>> print(s[-6])------------------P

-------------------------------------

>>> s="PYTHON"

>>> print(s)------------------PYTHON

>>> print(s[10])---------------IndexError: string index out of range

>>> print(s[-8])----------------IndexError: string index out of range

----------------------------------------------

>>> s="VAMSHI HYDERABAD PYTHON"

>>> s[0]------------------'V'

>>> s[len(s)-1]---------'N'

>>> s[-1]-----------------'N'

>>> s[-len(s)]-----------'V'

>>> len(s)----------------23

------------------------------------------------------------------------------------------------------------------------------------------

**2. Slicing Operations**

------------------------------------------------------------------------------------------------------------------------------------------

=>The purpose of Slicing Operations is that "To get Sub String OR part of values from main str object".

=>Here Sub String Represents Range of values OR Part of String

=>We can perform Slicing Operations by using The following Syntaxes

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Syntax-1 : strobj[ BEGIN : END ]**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

=>This Syntax Generates OR Obtains The values from BEGIN Index to END-1 Index Provided BEGIN<END Otherwise get Space

or ' ' as Result.

------------------------------------

**Examples**

------------------------------------

>>> s="PYTHON"

>>> print(s,type(s))-----------------PYTHON <class 'str'>

>>> s[0:4]------------------------------'PYTH'

>>> s[4:0]------------------------------' '

OR

>>> print(s[4:0])----------------------Space

>>> s[2:5]------------------------------'THO'

>>> s[1:6]------------------------------'YTHON'

>>> s[3:6]------------------------------'HON'

>>> s[2:6]------------------------------'THON'

>>> s[4:2]------------------------------' '

----------------------------------------------------

>>> s="PYTHON"

>>> print(s,type(s))----------------PYTHON <class 'str'>

>>> s[-6:-2]--------------------------'PYTH'

>>> s[-2:-6]--------------------------' '

>>> s[-4:-1]--------------------------'THO'

>>> s[-5:-2]--------------------------'YTH'

>>> s[-6:-4]--------------------------'PY'

>>> s[-6:-1]--------------------------'PYTHO'

>>> s[-6:-5]--------------------------'P'

>>> s[-3:-1]--------------------------'HO'

>>> s[-3:-2]-------------------------'H'

----------------------------------------------------------------------------------------------

**Sub Points---No Condition Applied (Begin <End )--Most Imp**

**here Begin and end are hetrogeneous**

----------------------------------------------------------------------------------------------

>>> s="PYTHON"

>>> print(s,type(s))--------------------PYTHON <class 'str'>

>>> s[-6:-2]------------------------------'PYTH'

>>> s[-2:-6]-----------------------------' '

>>> s[-4:-1]----------------------------'THO'

>>> s[-5:-2]----------------------------'YTH'

>>> s[-6:-4]----------------------------'PY'

>>> s[-6:-1]----------------------------'PYTHO'

>>> s[-6:-5]----------------------------'P'

>>> s[-3:-1]----------------------------'HO'

>>> s[-3:-2]----------------------------'H'

>>> s[1:-2]-----------------------------'YTH'

>>> s[0:-1]----------------------------'PYTHO'

>>> s[2:-1]----------------------------'THO'

>>> s[3:-1]----------------------------'HO'

>>> s[4:-1]----------------------------'O'

>>> s[5:-1]----------------------------' '

>>> s[2:-5]----------------------------' '

>>> s[3:-6]----------------------------' '

----------------------------------------------------------------------------------------------

**Most Special Sub Points---No Condition Applied (Begin <End )--Most Imp**

----------------------------------------------------------------------------------------------

>>> s="PYTHON"

>>> print(s,type(s))----------------------------PYTHON <class 'str'>

>>> s[2:300]-------------------------------------'THON'

>>> s[2:1000]------------------------------------'THON'

>>> s[0:1000]------------------------------------'PYTHON'

>>> s[1:1000]------------------------------------'YTHON'

---------------------------------

>>> s="PYTHON"

>>> print(s,type(s))---------------------------PYTHON <class 'str'>

>>> s[-200:-1]----------------------------------'PYTHO'

>>> s[-100:-2]----------------------------------'PYTH'

>>> s[-10:-3]-----------------------------------'PYT'

>>> s[-11:-4]------------------------------------'PY'

--------------------------------------------------------------------

>>> s="PYTHON"

>>> print(s,type(s))------------------------PYTHON <class 'str'>

>>> s[-100:4]--------------------------------'PYTH'

>>> s[-100:1]--------------------------------'P'

>>> s[-100:6]--------------------------------'PYTHON'

>>> s[-1000:1000]-------------------------'PYTHON'

>>> s[-23:23]-------------------------------'PYTHON'

>>> s[4:-23]---------------------------------' '

>>> s[23:-56]-------------------------------' '

>>> s[5:-6]----------------------------------' '

>>> s[-200:0]-------------------------------' '

>>> s="PYTHON"

>>> print(s,type(s))----------------PYTHON <class 'str'>

>>> s[-len(s):len(s)]---------------'PYTHON'

>>> s[-6:6]---------------------------'PYTHON'

>>> s[1000:-len(s)]----------------' '

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Syntax-2: strobj[ BEGIN : ]**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

=>In this Syntax, we are spcifying BEGIN Index and Not spicifying END Index.

=>If we don't Specify END Index then PVM Takes END Index as len(strobj)

(OR)

=>In otherwords, If we don't Specify END Index then PVM Takes BEGIN Indexed Data to upto Last Character

-------------------------

Examples

-------------------------

>>> s="PYTHON"

>>> print(s,type(s))---------------PYTHON <class 'str'>

>>> s[4:]-----------------------------'ON'

>>> s[2:]----------------------------'THON'

>>> s[0:]-----------------------------'PYTHON'

>>> s[1:]-----------------------------'YTHON'

>>> s[3:]-----------------------------'HON'

>>> s[2:]-----------------------------'THON'

------------------------------

>>> s="PYTHON"

>>> print(s,type(s))--------------------PYTHON <class 'str'>

>>> s[-4:]----------------------------------'THON'

>>> s[-2:]---------------------------------'ON'

>>> s[-6:]---------------------------------'PYTHON'

>>> s[-3:]---------------------------------'HON'

>>> s[-5:]---------------------------------'YTHON'

>>> s[-1:]---------------------------------'N'

------------------------------------------------

>>> s="PYTHON"

>>> print(s,type(s))----------------PYTHON <class 'str'>

>>> s[-1000:]-----------------------'PYTHON'

>>> s[-120:]------------------------'PYTHON'

>>> s[120:]-------------------------' '

>>> s[6:]----------------------------' '

>>> s[5:]----------------------------'N'

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Syntax-3: strobj[ : END ]

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

=>In this Syntax, we are specifying END Index and Not specifying BEGIN Index.

=>If we don't Specify BEGIN Index then PVM Takes either 0th Indexed Value or -len(strobj) Value to END-1 Index

(OR)

=>In otherwords, If we don't Specify BEGIN Index then PVM Takes from First Character to END-1 Index

--------------------------------------------

Examples

--------------------------------------------

>>> s="PYTHON"

>>> print(s,type(s))-----------------------PYTHON <class 'str'>

>>> s[:4]-------------------------------------'PYTH'

>>> s[:5]-------------------------------------'PYTHO'

>>> s[:6]-------------------------------------'PYTHON'

>>> s[:3]------------------------------------'PYT'

>>> s[:2]-----------------------------------'PY'

>>> s[:1]-----------------------------------'P'

---------------------------

>>> s="PYTHON"

>>> print(s,type(s))--------------------PYTHON <class 'str'>

>>> s[:-1]---------------------------------'PYTHO'

>>> s[:-3]----------------------------------'PYT'

>>> s[:-2]----------------------------------'PYTH'

>>> s[:-4]----------------------------------'PY'

>>> s[:-5]----------------------------------'P'

-------------------------------------

>>> s="PYTHON"

>>> print(s,type(s))------------------------PYTHON <class 'str'>

>>> s[:-200]---------------------------------' '

>>> s[:-120]--------------------------------' '

>>> s[:400]---------------------------------'PYTHON'

>>> s[:100]---------------------------------'PYTHON'

>>> s[:0b1010]----------------------------'PYTHON'

>>> s[:0xF]--------------------------------'PYTHON'

>>> s[0xA:]--------------------------------' '

>>> s[-(0xA):]-----------------------------'PYTHON'

---------------------------------------------------------------------------

>>> s="PYTHON"

>>> print(s,type(s))-------------------PYTHON <class 'str'>

>>> s[len("HYD"):len("KVR")]---------------' '

>>> s[-len("HYDERABAD"):len("KVR")]-----------'PYT'

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Examples -------RULE-4**

-------------------------------------------------------------------------

>>> s="PYTHON"

>>> print(s)---------------------PYTHON

>>> s[5:0:1]--------------------' '

>>> s[15:0:2]------------------' '

-------------------------------------------------------------------------

**Examples -------RULE-5**

-------------------------------------------------------------------------

>>> s="PYTHON"

>>> print(s)------------------PYTHON

>>> s[5:-1:-1]---------------' '

>>> s[15:-1:-2]--------------' '

>>> s[15:-1:-3]--------------' '

=================================================x================================**Special Ponits---String Palindrome**

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

>>> s="LIRIL"

>>> s[::]==s[::-1]

True

>>> s="DAD"

>>> s[::]==s[::-1]

True

>>> s="PYTHON"

>>> s[::]==s[::-1]

False

>>>"MADAM"=="MADAM"[::-1]

True

>>>"ROSSUM"=="ROSSUM"[::-1]

False

>>>"MALAYALAM"=="MALAYALAM"[::-1]

True

>>>"PYTHON"[::2]=="PYTHON"[-2::-1][::-1]

False

>>>"PYTHON"[::2]=="PYTHON"[-2::-2][::-1]

True

>>>"WINDOW"[::-1]!="WINDOW"[::1]

True

>>>"8558"[::]=="8558"[::-1]

True

>>>"MISSISSIPPI"[::-1]=="MISSISSIPPI"[::]

False

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

**Type Casting Techniques in Python**

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

=>The Process of Converting One Type Value into another Possible Type of Value is Called Type Casting.

=>In Python Programming, we have 5 Fundamental Type Casting Techniques. They are

1) int()

2) float()

3) bool()

4) complex()

5) str()

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

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

**1) int( )**

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

=>int( ) is used converting One Type of Possible Value into int type Value.

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

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

**Example1 : float type into int type : POSSIBLE**

===========================================================================>>> a=10.23

>>> print(a,type(a))-----------10.23 <class 'float'>

>>> b=int(a)

>>> print(b,type(b))-----------10 <class 'int'>

===========================================================================**Example2: bool type into int type : 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'>

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

**Example3: complex type into int type : NOT POSSIBLE**

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

>>> a=2+3j

>>> print(a,type(a))--------------(2+3j) <class 'complex'>

>>> b=int(a)------------------------TypeError: int() argument must be a string, a bytes-like object or a real number, not 'complex'

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

**Example4: str type into int type**

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

Case-1 : int str into int : Possible

---------------------------------------------------------

>>> a="123"

>>> print(a,type(a))-------------------------123 <class 'str'>

>>> a--------------------------------------------'123'

>>> b=int(a)

>>> print(b,type(b))-------------------------123 <class 'int'>

---------------------------------------------------------

**Case-2: float str into int: NOT Possible**

---------------------------------------------------------

>>> a="12.34"

>>> print(a,type(a))------------------------12.34 <class 'str'>

>>> a-------------------------------------------'12.34'

>>> b=int(a)----------------------------------ValueError: invalid literal for int() with base 10: '12.34'

**. Is considered as a special symbol that is the reason we can’t convert the str float into int using typecasting.**

---------------------------------------------------------

**Case-3: bool str into int: NOT Possible**

---------------------------------------------------------

>>> a="True"

>>> print(a,type(a))----------------True <class 'str'>

>>> a----------------------------------'True'

>>> b=int(a)----------------------------ValueError: invalid literal for int() with base 10: 'True'

---------------------------------------------------------

**Case-4: complex str into int: Not Possible**

---------------------------------------------------------

>>> a="2+3j"

>>> print(a,type(a))------------------2+3j <class 'str'>

>>> a-------------------------------------'2+3j'

>>> b=int(a)----------------------------ValueError: invalid literal for int() with base 10: '2+3j'

---------------------------------------------------------

**Case-5 : Pure Str into int: Not Possible**

---------------------------------------------------------

>>> a="PYTHON"

>>> print(a,type(a))-------------------PYTHON <class 'str'>

>>> a--------------------------------------'PYTHON'

>>> b=int(a)-----------------------------ValueError: invalid literal for int() with base 10: 'PYTHON'

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

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

**2) float()**

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

=>float() is used converting One Type of Possible Value into float type Value.

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

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

**Example1 : int type into float type : POSSIBLE**

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

>>> a=12

>>> print(a,type(a))-------------------12 <class 'int'>

>>> b=float(a)

>>> print(b,type(b))------------------12.0 <class 'float'>

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

**Example2: bool type into float type : 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'>

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

**Example3: complex type into float type : 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'

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

**Example4: str type into float type**

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

Case-1 : int str into float : POSSIBLE

---------------------------------------------------------

>>> a="123"

>>> print(a,type(a))---------------123 <class 'str'>

>>> a----------------------------------'123'

>>> b=float(a)

>>> print(b,type(b))-----------------123.0 <class 'float'>

---------------------------------------------------------

Case-2: float str into float: Possible

---------------------------------------------------------

>>> a="12.34"

>>> print(a,type(a))----------------------12.34 <class 'str'>

>>> a-----------------------------------------'12.34'

>>> b=float(a)

>>> print(b,type(b))---------------------12.34 <class 'float'>

------------------

>>> a="12.34.45"

>>> print(a,type(a))------------------12.34.45 <class 'str'>

>>> a-------------------------------------'12.34.45'

>>> b=float(a)--------------------------ValueError: could not convert string to float: '12.34.45'

---------------------------------------------------------

Case-3: bool str into float: NOT Possible

---------------------------------------------------------

>>> a="True"

>>> print(a,type(a))-----------------------------True <class 'str'>

>>> a------------------------------------------------'True'

>>> b=float(a)-------------------------------------ValueError: could not convert string to float: 'True'

---------------------------------------------------------

Case-4: complex str into float: NOT POSSIBLE

---------------------------------------------------------

>>> a="2+3j"

>>> print(a,type(a))----------------2+3j <class 'str'>

>>> a-----------------------------------'2+3j'

>>> b=float(a)------------------------------ValueError: could not convert string to float: '2+3j'

---------------------------------------------------------

Case-5 : Pure Str into float: NOT POSSIBLE

---------------------------------------------------------

>>> a="Python"

>>> print(a,type(a))-----------------------Python <class 'str'>

>>> a------------------------------------------'Python'

>>> b=float(a)------------------------------ValueError: could not convert string to float: 'Python'

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

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

**3) bool()**

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

=>bool() is used converting One Type of Possible Value into bool type Value.

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

=>**ALL NON-ZERO VALUES ARE CONSIDERED AS TRUE AND ZERO VALUE CONSIDERED AS FALSE**

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

Example1 : int type into bool type :POSSIBLE

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

>>> 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'>

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

Example2: float type into bool type : POSSIBLE

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

>>> a=12.34

>>> print(a,type(a))-----------------------12.34 <class 'float'>

>>> b=bool(a)

>>> print(b,type(b))-----------------------True <class 'bool'>

>>> a=0.0

>>> print(a,type(a))----------------------0.0 <class 'float'>

>>> b=bool(a)

>>> print(b,type(b))--------------------False <class 'bool'>

-------------------

>>> a=0.0000000000000000000000000000000000000000000000000000000001

>>> print(a,type(a))-----------------------1e-58 <class 'float'>

>>> b=bool(a)

>>> print(b,type(b))----------------------True <class 'bool'>

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

Example3: complex type into bool type :POSSIBLE

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

>>> a=2+3j

>>> print(a,type(a))-----------------------(2+3j) <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'>

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

Example4: str type into bool type

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

Case-1 : int str into bool : POSSIBLE

---------------------------------------------------------

>>> a="123"

>>> print(a,type(a))--------------------123 <class 'str'>

>>> a----------------------------------------'123'

>>> b=bool(a)

>>> print(b,type(b))-------------------True <class 'bool'>

>>> a="0"

>>> print(a,type(a))--------------------0 <class 'str'>

>>> a--------------------------------------'0'

>>> b=bool(a)

>>> print(b,type(b))--------------------True <class 'bool'>

>>> len(a)---------------------------------1

---------------------------------------------------------

Case-2: float str into bool: POSSIBLE

---------------------------------------------------------

>>> a="12.34"

>>> print(a,type(a))--------------------12.34 <class 'str'>

>>> a---------------------------------------'12.34'

>>> b=bool(a)

>>> print(b,type(b))--------------------True <class 'bool'>

>>> a="0.0"

>>> print(a,type(a))---------------------0.0 <class 'str'>

>>> a----------------------------------------'0.0'

>>> b=bool(a)

>>> print(b,type(b))---------------------True <class 'bool'>

>>> len(a)----------------------------------3

---------------------------------------------------------

Case-3: bool str into bool: POSSIBLE

---------------------------------------------------------

>>> a="True"

>>> print(a,type(a))--------------------True <class 'str'>

>>> a---------------------------------------'True'

>>> b=bool(a)

>>> print(b,type(b))--------------------True <class 'bool'>

--------------

>>> a="False"

>>> print(a,type(a))------------------False <class 'str'>

>>> a-------------------------------------'False'

>>> b=bool(a)

>>> print(b,type(b))------------------True <class 'bool'>

>>> len(a)-------------------------------5

---------------------------------------------------------

Case-4: complex str into bool: POSSIBLE

---------------------------------------------------------

>>> a="2+3j"

>>> print(a,type(a))----------------------2+3j <class 'str'>

>>> a-----------------------------------------'2+3j'

>>> b=bool(a)

>>> print(b,type(b))----------------------True <class 'bool'>

>>> a="0+0j"

>>> print(a,type(a))-----------------------0+0j <class 'str'>

>>> a------------------------------------------'0+0j'

>>> b=bool(a)

>>> print(b,type(b))-----------------------True <class 'bool'>

----------------------------

>>> print(bool(str(0+0j)[0:1]))-------------True

>>> print(bool(int(str(0+0j)[0:1])))--------False

---------------------------------------------------------

Case-5 : Pure Str into bool: Possible

---------------------------------------------------------

>>> a="PYTHON"

>>> print(a,type(a))-------------------PYTHON <class 'str'>

>>> a-------------------------------------'PYTHON'

>>> b=bool(a)

>>> print(b,type(b))----------------True <class 'bool'>

>>> a=""

>>> print(a,type(a))---------------Space <class 'str'>

>>> a---------------------------------' '

>>> b=bool(a)

>>> print(b,type(b))---------------True <class 'bool'>

>>> len(a)---------------------------3

----------------------------------

>>> a="" # Empty Str

>>> print(a,type(a))------------------ empty space <class 'str'>

>>> a-------------------------------------' '

>>> b=bool(a)

>>> print(b,type(b))-----------------False <class 'bool'>

>>> len(a)------------------------------0

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

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

4) complex()

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

=>complex() is used converting One Type of Possible Value into complex type Value.

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

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

Example1 : int type into complex type : Possible

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

>>> a=10

>>> print(a,type(a))------------------10 <class 'int'>

>>> b=complex(a)

>>> print(b,type(b))------------------(10+0j) <class 'complex'>

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

Example2: float type into complex type : Possible

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

>>> a=12.3

>>> print(a,type(a))----------------12.3 <class 'float'>

>>> b=complex(a)

>>> print(b,type(b))---------------(12.3+0j) <class 'complex'>

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

Example3: bool type into complex type :Possible

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

>>> a=True

>>> print(a,type(a))----------------True <class 'bool'>

>>> b=complex(a)

>>> print(b,type(b))----------------(1+0j) <class 'complex'>

>>> a=False

>>> print(a,type(a))----------------False <class 'bool'>

>>> b=complex(a)

>>> print(b,type(b))----------------0j <class 'complex'>

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

Example4: str type into compelx type

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

Case-1 : int str into complex : Possible

---------------------------------------------------------

>>> a="12"

>>> print(a,type(a))----------------12 <class 'str'>

>>> a------------------------------------'12'

>>> b=complex(a)

>>> print(b,type(b))-----------------(12+0j) <class 'complex'>

---------------------------------------------------------

Case-2: float str into complex: Possible

---------------------------------------------------------

>>> a="1.5"

>>> print(a,type(a))---------------1.5 <class 'str'>

>>> a----------------------------------'1.5'

>>> b=complex(a)

>>> print(b,type(b))--------------(1.5+0j) <class 'complex'>

---------------------------------------------------------

Case-3: bool str into complex: Not Possible

---------------------------------------------------------

>>> a="True"

>>> print(a,type(a))--------------True <class 'str'>

>>> a----------------------------------'True'

>>> b=complex(a)------------------ValueError: complex() arg is a malformed string

---------------------------------------------------------

Case-4: complex str into complex: Possible

---------------------------------------------------------

>>> a="2+3.5j"

>>> print(a,type(a))-------------2+3.5j <class 'str'>

>>> a--------------------------------'2+3.5j'

>>> b=complex(a)

>>> print(b,type(b))--------------(2+3.5j) <class 'complex'>

---------------------------------------------------------

Case-5 : Pure Str into complex:Not Possible

---------------------------------------------------------

>>> a="Python+JavaJ"

>>> print(a,type(a))-------------------Python+JavaJ <class 'str'>

>>> a-------------------------------------'Python+JavaJ'

>>> b=complex(a)---------------------ValueError: complex() arg is a malformed string

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

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

5) str()

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

=>str() converts 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===================================

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

2. bytes

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

Properties

-----------------------------------------------------------------

=>'bytes' is one of the pre-defined class and treated as Sequence Data Type.

=>The purpose "bytes" data tytpe is that "**To Implemenet End-to-Encryption(Encrypted Format)** "

=>The bytes stores Numerical Integer Values ranges (0,256----allows us to stored 0 to 255 only)

=>bytes data type does not contain any symbolic notation for storing bytes data bcoz Programmer is unable store

End-to-Encryption(Encrypted Format) data. But we can convert any other types values into bytes type values by using bytes().

Syntax: bytesobj=bytes(Iterable object)

=>An object of bytes data type maintains Insertion Order (Insertion Order is nothing but, whatever the order we Organize the Data, In the Same Order the Data will be send)

=>On the object of bytes, we can Perform both Indexing and Slicing Operations.

=>An object of bytes belongs IMMUTABLE bcoz bytes object does not support Item Assignment.

------------------------------------------------------------------------------------------------------------------------------------------------------------------

Examples

------------------------------------------------------------------------------------------------------------------------------------------------------------------

>>> lst=[10,20,256,112,45,56]

>>> print(lst,type(lst))---------------------------[10, 20, 256, 112, 45, 56] <class 'list'>

>>> b=bytes(lst)----------------------------------ValueError: bytes must be in range(0, 256)

>>> lst=[10,0,20,255,112,-45,56]

>>> print(lst,type(lst))--------------------------[10, 0, 20, 255, 112, -45, 56] <class 'list'>

>>> b=bytes(lst)---------------------------------ValueError: bytes must be in range(0, 256)

>>> lst=[10,0,20,25,112,45,56]

>>> print(lst,type(lst))--------------------------[10, 0, 20, 25, 112, 45, 56] <class 'list'>

>>> b=bytes(lst)

>>> print(b,type(b))-----------------------------b'\n\x00\x14\x19p-8' <class 'bytes'> End-to-Encryption

-----------------------------------------------------

>>> lst=[10,0,20,255,112,45,56]

>>> print(lst,type(lst))----------------------[10, 0, 20, 255, 112, 45, 56] <class 'list'>

>>> b=bytes(lst)

>>> print(b,type(b))------------------------b'\n\x00\x14\xffp-8' <class 'bytes'>

>>> b[0]--------------------------------------10

>>> b[-1]-------------------------------------56

>>> b[-2]-------------------------------------45

>>> b[0:5]-------------------------------------b'\n\x00\x14\xffp'

>>> for kvr in b:

... print(kvr)

...

10

0

20

255

112

45

56

>>> for kvr in b[0:5]:

... print(kvr)

...

10

0

20

255

112

>>> for kvr in b[::-1]:

... print(kvr)

...

56

45

112

255

20

0

10

>>> for kvr in b[::2]:

... print(kvr)

...

10

20

112

56

------------------------------------------------------------

>>> lst=[10,0,20,255,112,45,56]

>>> print(lst,type(lst))-----------------------[10, 0, 20, 255, 112, 45, 56] <class 'list'>

>>> b=bytes(lst)

>>> print(b,type(b),id(a))---------------b'\n\x00\x14\xffp-8' <class 'bytes'> 1739275742320

>>> b[0]---------------------10

>>> b[0]=100 # trying to update-------------TypeError: 'bytes' object does not support item assignment

--------------------------------------------------------------------------------------------------------------

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

3. bytearray

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

Properties

-----------------------------------------------------------------

=>'bytearray' is one of the pre-defined class and treated as Sequence Data Type.

=>The purpose "bytearray" data tytpe is that "To Implemenet End-to-Encryption(Encrypted Format) "

=>The bytearray stores Numerical Integer Values ranges (0,256----allows us to stored 0 to 255 only)

=>bytearray data type does not contain any symbolic notation for storing bytearray data bcoz Programmer is unable store End-to-Encryption(Encrypted Format) data. But we can convert any other types values into bytearray type values by using bytearray().

**Syntax: bytesobj=bytearray(Iterable object)**

=>An object of bytearray data type maintains Insertion Order (Insertion Order is nothing but, whatever the order we Organize the Data, In the Same Order the Data will be send)

=>On the object of bytearray, we can Perform both Indexing and Slicing Operations.

=>An object of bytearray belongs MUTABLE bcoz bytearray object supports Item Assignment.

------------------------------------------------------------------------------------------------------------------------------------------

**Note: The Functionality of bytearray is exactly similar to bytes But an object of bytes belongs to IMMUTABLE where as an object of bytearray belongs to MUTABLE.**

------------------------------------------------------------------------------------------------------------------------------------------

Examples

------------------------------------------------------------------------------------------------------------------------------------------

>>> lst=[10,23,256,45,0,12,78]

>>> print(lst,type(lst))-----------------------[10, 23, 256, 45, 0, 12, 78] <class 'list'>

>>> **b=bytearray(lst)-------------------------ValueError: byte must be in range(0, 256)**

>>> lst=[10,-23,255,45,0,12,78]

>>> print(lst,type(lst))----------------------[10, -23, 255, 45, 0, 12, 78] <class 'list'>

>>> b=bytearray(lst)------------------------ValueError: byte must be in range(0, 256)

-----------------------------------------------------------------------------------------------------------------------------

>>> lst=[10,23,255,45,0,12,78]

>>> print(lst,type(lst))---------------------[10, 23, 255, 45, 0, 12, 78] <class 'list'>

>>> b=bytearray(lst)

>>> print(b,type(b))------------------------bytearray(b'\n\x17\xff-\x00\x0cN') <class 'bytearray'>

>>> b[0]----------------------------------------10

>>> b[-1]---------------------------------------78

>>> b[0:5]-------------------------------------bytearray(b'\n\x17\xff-\x00')

>>> for val in b:

... print(val)

...

10

23

255

45

0

12

78

>>> for val in b[0:5]:

... print(val)

...

10

23

255

45

0

>>> for val in b[::-1]:

... print(val)

...

78

12

0

45

255

23

10

-----------------------------------------------------------------------------------------------------------------------------------------

>>> lst=[10,23,255,45,0,12,78]

>>> print(lst,type(lst))------------------------[10, 23, 255, 45, 0, 12, 78] <class 'list'>

>>> b=bytearray(lst)

>>> print(b,type(b),id(b))-------------------bytearray(b'\n\x17\xff-\x00\x0cN') <class 'bytearray'> 1662541281776

>>> for val in b:

... print(val)

...

10

23

255

45

0

12

78

>>> b[0]--------------------10

>>> b[0]=100 ---------------- Updating the Value of bytearray--bytearray object supports Item Assignment.

>>> print(b,type(b),id(b))-------------bytearray(b'd\x17\xff-\x00\x0cN') <class 'bytearray'> 1662541281776

>>> for val in b:

... print(val)

...

100

23

255

45

0

12

78

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

**4. range**

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

=>'range' is one of the pre-defined Data type and treated as Sequence Data Type.

=>The purpose of range data type is that "**To Store Sequence of of Numerical Integer Values by Maintaining Equal Interval Value(Step) either in Forward or Backward Direction**".

=>An Object of range data type maintains Insertion Order

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

=>An object of range belongs to IMMUTABLE bcoz 'range' object does not support item assignment

=>To impement the purpose of range data type, we have Three Pre-defined Functions.

They are

------------------------------------------------------------------------------------------------------------------------------------------

**Syntax1: range(Value)**

------------------------------------------------------------------------------------------------------------------------------------------

=>The Syntax generates range of Values from 0 to Value-1

--------------------

Examples:

--------------------

>>> r=range(6)

>>> print(r,type(r))

range(0, 6) <class 'range'>

>>> for val in r:

... print(val)

...

0

1

2

3

4

5

>>> for val in range(6):

... print(val)

...

0

1

2

3

4

5

=>here Default Step is +1 and Default Direction is Forward Direction

------------------------------------------------------------------------------------------------------------------------------------------

**Syntax2: range(Start,Stop)**

------------------------------------------------------------------------------------------------------------------------------------------

=>The Syntax generates range of Values from Start to Stop-1

-------------------------

Examples

-------------------------

>>> r=range(100,106)

>>> print(r,type(r))--------------------range(100, 106) <class 'range'>

>>> for val in r:

... print(val)

...

100

101

102

103

104

105

>>> r[0]-------------------100

>>> r[-1]-------------------105

>>> r[-2]-------------------104

>>> r[-12]------------------IndexError: range object index out of range

>>> r[12]------------------IndexError: range object index out of range

>>> for val in range(10,16):

... print(val)

...

10

11

12

13

14

15

=>here Default Step is +1 and Default Direction is Forward Direction

=>The Above Two Syntaxes by default follows Default Step is +1 and Default Direction is Forward Direction

------------------------------------------------------------------------------------------------------------------------------------------**Syntax3: range(Start,Stop,Step)**

------------------------------------------------------------------------------------------------------------------------------------------

=>The Syntax generates range of Values from Start to Stop-1 by maintaining equal Interval of Value by following Specified Step Value.

-------------------------------

Examples

--------------------------------

>>> r=range(10,21,2)

>>> print(r,type(r))-------------------range(10, 21, 2) <class 'range'>

>>> for val in r:

... print(val)

...

10

12

14

16

18

20

>>> for val in range(10,21,3):

... print(val)

...

10

13

16

19

----------------------------

>>> for val in range(10,5,-1):

... print(val)

...

10

9

8

7

6

If the step in negative and then end+1 (values up to end+1)

----------------------------------------

>>> r=range(10,21,2)

>>> for val in r:

... print(val)

...

10

12

14

16

18

20

>>> for val in r[0:3]:

... print(val)

...

10

12

14

--------------------------

>>> r=range(10,21,2)

>>> print(r)---------range(10, 21, 2)

>>> for val in r:

... print(val)

...

10

12

14

16

18

20

**>>> r[1]=110------------------TypeError: 'range' object does not support item assignment**

------------------------------------------------------------------------------------------------------------------------------------------

**Examples**

------------------------------------------------------------------------------------------------------------------------------------------

Q1) 0 1 2 3 4 5 6 7 8 9 -----------range(10) OR range(0,10) OR range(0,10,1)

>>> for v in range(10):

... print(v)

...

0

1

2

3

4

5

6

7

8

9

>>> for v in range(0,10):

... print(v)

...

0

1

2

3

4

5

6

7

8

9

>>> for v in range(0,10,1):

... print(v)

...

0

1

2

3

4

5

6

7

8

9

------------------------------------------------------------------------------------------------------------------------------------------

Q2) 10 11 12 13 14 15 16 17 18 19 20-----range(10,21) OR range(10,21,1)

>>> for v in range(10,21):

... print(v)

...

10

11

12

13

14

15

16

17

18

19

20

>>> for v in range(10,21,1):

... print(v)

...

10

11

12

13

14

15

16

17

18

19

20

------------------------------------------------------------------------------------------------------------------------------------------

Q3) 1000 1001 1002 1003 1004 1005---range(1000,1006) OR range(1000,1006,1)

>>> for val in range(1000,1006):

... print(val)

...

1000

1001

1002

1003

1004

1005

>>> for val in range(1000,1006,1):

... print(val)

...

1000

1001

1002

1003

1004

1005

------------------------------------------------------------------------------------------------------------------------------------------

Q4) 10 12 14 16 18 20--range(10,21,2)

>>> for v in range(10,21,2):

... print(v)

...

10

12

14

16

18

20

------------------------------------------------------------------------------------------------------------------------------------------Q5) 1000 1050 1100 1150,1200-----range(1000,1201,50)

>>> for v in range(1000,1201,50):

... print(v)

...

1000

1050

1100

1150

1200

------------------------------------------------------------------------------------------------------------------------------------------

Q6) -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

------------------------------------------------------------------------------------------------------------------------------------------

Q7) 100 90 80 70 60 50 40 30 20 10 ------range(100,9,-10)

>>> for v in range(100,9,-10):

... print(v)

...

100

90

80

70

60

50

40

30

20

10

------------------------------------------------------------------------------------------------------------------------------------------

Q8) -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

------------------------------------------------------------------------------------------------------------------------------------------

Q9) -100 -90 -80 -70 -60 -50-----range(-100,-49,10)

>>> for v in range(-100,-49,10):

... print(v)

...

-100

-90

-80

-70

-60

-50

------------------Reverse of Above Data--------------------

>>> for v in range(-100,-49,10)[::-1]:

... print(v)

...

-50

-60

-70

-80

-90

-100

------------------------------------------------------------------------------------------------------------------------------------------

Q10) -5 -4 -3 -2 -1 0 1 2 3 4 5--------range(-5,6,1) OR range(-5,6)

>>> for val in range(-5,6,1):

... print(val)

...

-5

-4

-3

-2

-1

0

1

2

3

4

5

>>> for val in range(-5,6):

... print(val)

...

-5

-4

-3

-2

-1

0

1

2

3

4

5

>>> for val in range(-5,6,1)[::-1]:

... print(val)

...

5

4

3

2

1

0

-1

-2

-3

-4

-5

------------------------------------------------------------------------------------------------------------------------------------------

**Special Points**

-----------------------------------------------

>>> print(range(100,101,10)[-1])---------------100

>>> print(range(100,201,10)[-1])------------------200

>>> print(range(100,201,10)[True])--------------110

>>> print(range(100,201,10)[len("HYD")])---------130

>>> print(range(1000,2001,100)[::-1][-1])---------1000

------------------------------------------------------------------------------------------------------------------------------------------

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

**Mutable and Immutable Objects**

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

----------------------------

**Mutable Object**

----------------------------

=>A Mutable Object object is One, whose Content can be Changed / Modified / Updated at Same Address.

Examples:

--------------------------

**bytearry, list,set,dict**

----------------------------

**Immutable Object**

----------------------------

=>An Immutable Object object is One,Which will satisfy the following Properties.

a) Immutable Object Values not Possible Change / Update at same Address( Immutable Object Values Modified But whose Modified Values placed at Different Address)

b) Immutable Objects does not support Item Assignment.

**Examples: int,float,bool,complex,str,bytes,range,tuple,set,frozenset,NoneType**

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

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

**III. List Category Data Types (Collections Data Types)**

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

=>The purpose of List Category Data Types is that "To Store Multiple Values of Same Type OR Different Type Or the Both the Types with Unique and Duplicate Values in Single Object".

=>In we TWO Data Types in List Category. They are

1. list (Mutable)

2. tuple (Immutable)

===========================================x======================================

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

**list**

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

Index

--------------------

=>Properties of list

=>What is list

=>Notations of list

=>Types of list

=>Operations on list

=>Pre-defined Functions in list

=>Nested OR Inner List

=>Programming Examples

------------------------------------------------------------------------------------------------------------------------------------------

**Properties of list**

------------------------------------------------------------------------------------------------------------------------------------------

=>'list' is one of the Pre-Defined Class and Treated as List Data Type.

=>The purpose of 'list' data type is that "To Store Multiple Values of Same Type OR Different Type Or the Both the Types with Unique and Duplicate Values in Single Object

=>To store the Elements OR Values in the object of list, we use Square Brackets [ ] and the Values separated by comma.

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

=>An object of list belongs to **MUTABLE** bcoz list object allows us to perform Item Assignment.

=>An Object of list maintains Insertion Order.

=>In Python Programming, we can create Two Types of list objects. They are

1) Empty List

2) Non-Empty List

------------------------------------

**1) Empty List**

------------------------------------

=>An Empty List is one which does not contain any Elements and whose length is 0

**=>Syntax: varname=[ ] (OR) varname=list()**

---------------------------------

Examples

---------------------------------

>>> lst1=[]

>>> print(lst1,type(lst1))--------------[] <class 'list'>

>>> len(lst1)------------------------------0

>>> lst2=list()

>>> print(lst2,type(lst2))-------------[] <class 'list'>

>>> len(lst2)-----------------------------0

------------------------------------

**2) Non-Empty List**

------------------------------------

=>A Non-Empty List is one which contains Elements and whose length is >0

**=>Syntax: varname=[Val1,Val2,.....,Val-n] OR varname=list(object)**

------------------------------------------------------------------------------------------------------------------------------------------

Examples

------------------------------------------------------------------------------------------------------------------------------------------

>>> lst1=[10,20,30,10,20,50,60]

>>> print(lst1,type(lst1))-------------[10, 20, 30, 10, 20, 50, 60] <class 'list'>

>>> lst2=[10,"Rossum",34.56,True,2+3j]

>>> print(lst2,type(lst2))--------------[10, 'Rossum', 34.56, True, (2+3j)] <class 'list'>

----------------------------

>>> lst2=[10,"Rossum",34.56,True,2+3j]

>>> print(lst2,type(lst2),id(lst2))-----------------[10, 'Rossum', 34.56, True, (2+3j)] <class 'list'> 2759247987200

>>> lst2[0]----------------------------------------------10

>>> lst2[-1]---------------------------------------------(2+3j)

>>> lst2[-2]--------------------------------------------True

>>> lst2[2]-------------------34.56

>>> lst2[12]-------------------IndexError: list index out of range

>>> lst2[0:3]----------------[10, 'Rossum', 34.56]

>>> lst2[-5::2]--------------[10, 34.56, (2+3j)]

>>> lst2[::-1]---------------[(2+3j), True, 34.56, 'Rossum', 10]

----------------------------

>>> lst2=[10,"Rossum",34.56,True,2+3j]

>>> print(lst2,type(lst2),id(lst2))-----------[10, 'Rossum', 34.56, True, (2+3j)] <class 'list'> 2759250362560

>>> lst2[0]=100 # We Updating the value of list

>>> print(lst2,type(lst2),id(lst2))--------[100, 'Rossum', 34.56, True, (2+3j)] <class 'list'> 2759250362560

>>> lst2[-3]=99.99 # We Updating the value of list

>>> print(lst2,type(lst2),id(lst2))----------[100, 'Rossum', 99.99, True, (2+3j)] <class 'list'> 2759250362560

-----------------------

>>> lst2=[10,"Rossum",34.56,True,2+3j]

>>> print(lst2,type(lst2),id(lst2))--------[10, 'Rossum', 34.56, True, (2+3j)] <class 'list'> 2759247987200

>>> lst2[0:2]=[100,"Travis"] # Slice Based Updation

>>> print(lst2,type(lst2),id(lst2))------------[100, 'Travis', 34.56, True, (2+3j)] <class 'list'> 2759247987200

>>> lst2[::2]=[1000,66.66,3.5+5.6j] # Slice Based Updation

>>> print(lst2,type(lst2),id(lst2))-------[1000, 'Travis', 66.66, True, (3.5+5.6j)] <class 'list'> 2759247987200

------------------------------------------------

>>> b=bytes([100,200,250,120])

>>> print(b,type(b))------------------------b'd\xc8\xfax' <class 'bytes'>

>>> l1=list(b)

>>> print(l1,type(l1))--------------------[100, 200, 250, 120] <class 'list'>

--------------------

>>> s="PYTHON"

>>> print(s,type(s))----------------PYTHON <class 'str'>

>>> l1=list(s)

>>> print(l1,type(l1))-----------------['P', 'Y', 'T', 'H', 'O', 'N'] <class 'list'>

>>> s="MISSISSIPPI"

>>> print(s,type(s))----------------MISSISSIPPI <class 'str'>

>>> l1=list(s)

>>> print(l1,type(l1))-------------['M', 'I', 'S', 'S', 'I', 'S', 'S', 'I', 'P', 'P', 'I'] <class 'list'>

>>> s="ABRAKADABRA"

>>> print(s,type(s))----------------ABRAKADABRA <class 'str'>

>>> l1=list(s)

>>> print(l1,type(l1))---------------['A', 'B', 'R', 'A', 'K', 'A', 'D', 'A', 'B', 'R', 'A'] <class 'list'>

-------------------------------------------

>>> l1=[10]

>>> print(l1,type(l1))---------------[10] <class 'list'>

==========================================x=======================================

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

**Pre-defined Functions in list**

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

=>We know that on the object of list, we can perform Both Indexing and Slicing.

=>With the Help of Indexing and Slicing, we can get the Value(s) from List object and we can also update the value(s) of list.

=>To perform Various Operations on list object, we use pre-defined functions of list object and they are given bellow

------------------------------------------------------------------------------------------------------------------------------------------

**1. append()**

------------------------------------------------------------------------------------------------------------------------------------------

=>Syntax: listobj.append(Value)

=>This Function is used for adding the value to list at the end( called appending).

------------------------------------------

Examples

------------------------------------------

>>> lst1=[10,"Rossum"]

>>> print(lst1,id(lst1))--------[10, 'Rossum'] 2759245740544

>>> lst1.append(34.56)

>>> print(lst1,id(lst1))-----------[10, 'Rossum', 34.56] 2759245740544

>>> lst1.append(True)

>>> print(lst1,id(lst1))-----------[10, 'Rossum', 34.56, True] 2759245740544

------------------------------------------------------------------------------------------------------------------------------------------

**2) insert()**

------------------------------------------------------------------------------------------------------------------------------------------

=>Syntax: listobj.insert(Index,Value)

=>This Function is used for Inserting the Specified Value in list object at Perticular Existing Index.

=>If we Specify Invalid Possitive Index Then Value inserted in list object at Last / End

=>If we Specify Invalid Negative Index Then Value inserted in list object at First / Begin

-------------------------------

Examples

------------------------------

>>> lst2=[10,"Rossum",34.56,True,2+3j]

>>> print(lst2,id(lst2))------------------[10, 'Rossum', 34.56, True, (2+3j)] 2759248008512

>>> lst2.insert(2,"PYTHON")

>>> print(lst2,id(lst2))-----------[10, 'Rossum', 'PYTHON', 34.56, True, (2+3j)] 2759248008512

>>> lst2[3]=44.44

>>> print(lst2,id(lst2))---------[10, 'Rossum', 'PYTHON', 44.44, True, (2+3j)] 2759248008512

>>> lst2.insert(1,"GUIDO VAN")

>>> print(lst2,id(lst2))--------[10, 'GUIDO VAN', 'Rossum', 'PYTHON', 44.44, True, (2+3j)] 2759248008512

>>> lst2[1]=lst2[1]+""+lst2[2]

>>> print(lst2,id(lst2))----[10, 'GUIDO VAN Rossum', 'Rossum', 'PYTHON', 44.44, True, (2+3j)] 2759248008512

-----------------------------

>>> lst1=[10,20,30,40,50,10,60]

>>> print(lst1,id(lst1))---------[10, 20, 30, 40, 50, 10, 60] 2759247937088

>>> lst1.insert(3,300)

>>> print(lst1,id(lst1))---------[10, 20, 30, 300, 40, 50, 10, 60] 2759247937088

>>> lst1.insert(-1,70)

>>> print(lst1,id(lst1))---------[10, 20, 30, 300, 40, 50, 10, 70, 60] 2759247937088

-----------------------------

>>> lst1=[10,20,30,40]

>>> print(lst1,id(lst1))----------------[10, 20, 30, 40] 2759245740544

>>> lst1.insert(2000,70)

>>> print(lst1,id(lst1))-------------[10, 20, 30, 40, 70] 2759245740544

>>> lst1.insert(20999900,70)

>>> print(lst1,id(lst1))--------------[10, 20, 30, 40, 70, 70] 2759245740544

>>> lst1.insert(0,99)

>>> print(lst1,id(lst1))--------------------[99, 10, 20, 30, 40, 70, 70] 2759245740544

>>> lst1.insert(-1000,100)

>>> print(lst1,id(lst1))-------------------[100, 99, 10, 20, 30, 40, 70, 70] 2759245740544

------------------------------------------------------------------------------------------------------------------------------------------

**3) clear()**

------------------------------------------------------------------------------------------------------------------------------------------

=>Syntax: listobj.clear()

=>This Function is used for Removing all the elements of list

=>**When we call this Function on empty list then we get None Or Space as Result**

--------------------------------

Examples

--------------------------------

>>> lst=[10,"RS",34.56,"Python"]

>>> print(lst,id(lst))--------------[10, 'RS', 34.56, 'Python'] 1714677404096

>>> len(lst)-------------------------4

>>> lst.clear()

>>> print(lst,id(lst))-------------[] 1714677404096

>>> len(lst)-----------------------0

-------------------------------------------

>>> print(lst.clear())-------------None

(OR

>>> lst.clear()-----------------------Sapce (bcoz nothing to remove from empty list)

>>> print([].clear())---------------None

>>> print(list().clear())----------None

------------------------------------------------------------------------------------------------------------------------------------------**4) remove()--Value Based Removal**

------------------------------------------------------------------------------------------------------------------------------------------

Syntax: listobj.remove(Value)

=>This Function is used for Removing First Occurence of Specified Value from Non-empty list object

=>If the Specified Value does not exist Non-empty list object then we get **ValueError**

=>When we call this Function on empty list then we get **ValueError**

---------------------------

**Example1**

--------------------------

>>> lst=[10,"RS",34.56,"Python"]

>>> print(lst,id(lst))--------------------------[10, 'RS', 34.56, 'Python'] 1714679266688

>>> lst.remove("Python")

>>> print(lst,id(lst))-------------------------[10, 'RS', 34.56] 1714679266688

>>> lst.remove("RS")

>>> print(lst,id(lst))------------------------[10, 34.56] 1714679266688

>>> lst.remove(10)

>>> print(lst,id(lst))------------------------[34.56] 1714679266688

>>> lst.remove(34.56)

>>> print(lst,id(lst))-----------------------[] 1714679266688

>>> lst.remove(10)-----------------------ValueError: list.remove(x): x not in list

--------------------------------

**Examples2**

--------------------------------

>>> lst1=[10,20,30,10,40,20,50]

>>> print(lst1,id(lst1))----------------------[10, 20, 30, 10, 40, 20, 50] 1714679316864

>>> lst1.remove(10)

>>> print(lst1,id(lst1))---------------------[20, 30, 10, 40, 20, 50] 1714679316864

>>> lst1.remove(10)

>>> print(lst1,id(lst1))--------------------[20, 30, 40, 20, 50] 1714679316864

>>> lst1.remove(10)----------------------ValueError: list.remove(x): x not in list

>>> lst1.remove(20)

>>> print(lst1,id(lst1))------------------[30, 40, 20, 50] 1714679316864

------------------------------------------

>>> [10,20,30,40].remove(30)

>>> list([10,20,30,40]).remove(30)

>>> list([10,20,30,40]).remove(300)---------ValueError: list.remove(x): x not in list

------------------------------------------------------------------------------------------------------------------------------------------

**5) pop(index)-----Index Based Removal**

------------------------------------------------------------------------------------------------------------------------------------------

=>Listobj.pop(Index)

=>This Functrion is used for Removing the Element of list based on Index.

=>Here Index can either +ve Or -ve

=>When we this function on empty list then we get **IndexError**

---------------------------------------

Examples

---------------------------------------

>>> lst=[10,"RS",34.56,"Python"]

>>> print(lst,id(lst))-------------------------------[10, 'RS', 34.56, 'Python'] 1714679266752

>>> lst.pop(2)--------------------------------------34.56

>>> print(lst,id(lst))-------------------------------[10, 'RS', 'Python'] 1714679266752

>>> lst.pop(-1)-------------------------------------'Python'

>>> print(lst,id(lst))-------------------------------[10, 'RS'] 1714679266752

>>> lst.pop(-2)-------------------------------------10

>>> print(lst,id(lst))------------------------------['RS'] 1714679266752

>>> lst.pop(0)------------------------------------'RS'

>>> print(lst,id(lst))-------------------------------[] 1714679266752

>>> lst.pop(2)------------------------------------IndexError: pop from empty list

------------------

>>> lst1=[10,20,30,10,40,20,50]

>>> print(lst1,id(lst1))------------------------[10, 20, 30, 10, 40, 20, 50] 1714679266688

>>> lst1.pop(3)---------------------------------10

>>> print(lst1,id(lst1))------------------------[10, 20, 30, 40, 20, 50] 1714679266688

>>> lst1.pop(-5)--------------------------------20

>>> print(lst1,id(lst1))------------------------[10, 30, 40, 20, 50] 1714679266688

>>> lst1.pop(13)-------------------------------IndexError: pop index out of range

>>> lst1.pop(-13)-----------------------------IndexError: pop index out of range

------------------------------

>>> [].pop(0)----------------------------IndexError: pop from empty list

>>> [10,20,30,40,10].pop(0)---------10

>>> list(["Python","C",10,True]).pop(1)------'C'

>>> list(["Python","C",10,True]).pop(-1)-------True

>>> list(["Python","C",10,True]).pop(-11)--------IndexError: pop index out of range

------------------------------------------------------------------------------------------------------------------------------------------

**6) pop()**

------------------------------------------------------------------------------------------------------------------------------------------

=>Syntax: listobj.pop()

=>This Function is used for Removing the Last Element of Non-empty list

=>If we call this function on empty list object then we get **IndexError**

-------------------------

Examples

-------------------------

>>> lst=[10,"RS",34.56,"Python"]

>>> print(lst,id(lst))---------------[10, 'RS', 34.56, 'Python'] 1714679264640

>>> lst.pop()-----------------------'Python'

>>> print(lst,id(lst))--------------[10, 'RS', 34.56] 1714679264640

>>> lst.pop()----------------------34.56

>>> print(lst,id(lst))-------------[10, 'RS'] 1714679264640

>>> lst.pop()---------------------'RS'

>>> print(lst,id(lst))------------[10] 1714679264640

>>> lst.pop()---------------------10

>>> print(lst,id(lst))------------[] 1714679264640

>>> lst.pop()------------------IndexError: pop from empty list

>>> [].pop()-------------------IndexError: pop from empty list

>>> list().pop()---------------IndexError: pop from empty list

>>> [10,20,30,40].pop()------40

---------------------------------------------------------------------------------------------------------------------------------------------------------------------------

**NOTE: del operator**

Syntax1: del MuTableobject[Index]----Removes Element of Mutable Object Based on Valid **Index**

Syntax2: del Mutableobject[Begin:End:Step]------Removes Elements of Mutable Object **Based on Slicing Operations**

Syntax3: del anyobjectname------Removes Complete object(Elements + Object Memory space)

**Examples**

--------------------

>>> lst=[10,"RS",34.56,"Python"]

>>> print(lst,id(lst))---------------[10, 'RS', 34.56, 'Python'] 1714679264640

>>> del lst[1]

>>> print(lst,id(lst))---------------[10, 34.56, 'Python'] 1714679264640

>>> del lst[-1]

>>> print(lst,id(lst))---------------[10, 34.56] 1714679264640

>>> del lst[1]

>>> print(lst,id(lst))-----------------[10] 1714679264640

>>> del lst[11]--------------------------IndexError: list assignment index out of range

---------------------------------

>>> lst=[10,"RS",34.56,"Python",True,2+3j]

>>> print(lst,id(lst))---------------------[10, 'RS', 34.56, 'Python', True, (2+3j)] 1714679266816

>>> del lst[::2]

>>> print(lst,id(lst))--------------------['RS', 'Python', (2+3j)] 1714679266816

>>> del lst[0:2]

>>> print(lst,id(lst))-------------------[(2+3j)] 1714679266816

>>> del lst[::]

>>> print(lst,id(lst))------------------[] 1714679266816

>>> del lst

>>> print(lst,id(lst))-------------------NameError: name 'lst' is not defined.

-------------------------

**Examples on del operator with str data--immutable**

------------------------------------

>>> s="PYTHON"

>>> s[0]="J"---------------------------------TypeError: 'str' object does not support item assignment

>>> del s[-2]----------------------------------TypeError: 'str' object doesn't support item deletion

>>> del s[0:6:2]-----------------------------TypeError: 'str' object does not support item deletion

>>> del s

>>> print(s)-----------------------------------NameError: name 's' is not defined

------------------------------------------------------------------------------------------------------------------------------------------**7) copy()**

------------------------------------------------------------------------------------------------------------------------------------------**=>Syntax: listobj2=listobj1.copy()**

=>This Function is used for Implementing Shallow Copy.

--------------------------------------

Examples:

---------------------------------------

>> lst1=[10,"RS"]

>>> print(lst1,id(lst1))--------------------[10, 'RS'] 1714679266816

>>> lst2=lst1.copy() # Shallow Copy

>>> print(lst2,id(lst2))-------------------[10, 'RS'] 1714679266688

>>> lst1.append("PYTHON")

>>> lst2.insert(1,"NL")

>>> print(lst1,id(lst1))------------------[10, 'RS', 'PYTHON'] 1714679266816

>>> print(lst2,id(lst2))------------------[10, 'NL', 'RS'] 1714679266688

-----------------------------------

Examples----Deep Copy

----------------------------------------

>>> lst1=[10,"RS"]

>>> print(lst1,id(lst1))-----------------[10, 'RS'] 1714679264640

>>> lst2=lst1 # Deep Copy

>>> print(lst2,id(lst2))----------------[10, 'RS'] 1714679264640

>>> lst1.append("Python")

>>> print(lst1,id(lst1))----------------[10, 'RS', 'Python'] 1714679264640

>>> print(lst2,id(lst2))----------------[10, 'RS', 'Python'] 1714679264640

>>> lst2.insert(1,"NL")

>>> print(lst1,id(lst1))----------------[10, 'NL', 'RS', 'Python'] 1714679264640

>>> print(lst2,id(lst2))----------------[10, 'NL', 'RS', 'Python'] 1714679264640

------------------------------------------------

Slice Based Copy

-------------------------------------------------

>>> lst1=[10,"RS","Python",34.56,True]

>>> print(lst1,id(lst1))------------------------[10, 'RS', 'Python', 34.56, True] 1714679266688

>>> lst2=lst1[::2] # Slice Based Operation

>>> print(lst2,id(lst2))----------------------[10, 'Python', True] 1714679266816

----------------------------

>>> lst1=[10,"RS","Python",34.56,True]

>>> print(lst1,id(lst1))--------------------------[10, 'RS', 'Python', 34.56, True] 1714679264640

>>> lst2=lst1[::] # Slice Based Operation

>>> print(lst2,id(lst2))--------------------------[10, 'RS', 'Python', 34.56, True] 1714679266688

------------------------------------------------------------------------------------------------------------------------------------------

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

Copy Techniques in Python

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

=>In Python Programming, we have 2 Types of Copy Techniques. They are

1. Shallow Copy

2. Deep Copy

------------------------------------------------------------------------------------------------------------------------------------------

1. Shallow Copy

------------------------------------------------------------------------------------------------------------------------------------------

=>The Properties of Shallow Copy are

a) Initial Content of Both the Objects are Same

b) The Memory Address of Both the Objects are Different

c) Modifications are Independent

(Whatever the changes we do on One object, They are not Reflecting to another object )

=>To Implement Shallow Copy, we use copy()

**=>Syntax: object2=object1.copy()**

-----------------------------------------

Examples

-----------------------------------------

>> lst1=[10,"RS"]

>>> print(lst1,id(lst1))--------------------[10, 'RS'] 1714679266816

>>> lst2=lst1.copy() # Shallow Copy

>>> print(lst2,id(lst2))-------------------[10, 'RS'] 1714679266688

>>> lst1.append("PYTHON")

>>> lst2.insert(1,"NL")

>>> print(lst1,id(lst1))------------------[10, 'RS', 'PYTHON'] 1714679266816

>>> print(lst2,id(lst2))------------------[10, 'NL', 'RS'] 1714679266688

------------------------------------------------------------------------------------------------------------------------------------------

2. Deep Copy

------------------------------------------------------------------------------------------------------------------------------------------

=>The Properties of Deep Copy are

a) Initial Content of Both the Objects are Same

b) The Memory Address of Both the Objects are Same

c) Modifications are Dependent

(Whatever the changes we do on One object, They are Reflecting to another object bcoz Both the objects Pointing to Same Memory Address )

=>To Implement Deep Copy, we use Assigment Operator ( = )

=>Syntax: object1=object2

-----------------------------------------

Examples

----------------------------------------

>>> lst1=[10,"RS"]

>>> print(lst1,id(lst1))-----------------[10, 'RS'] 1714679264640

>>> lst2=lst1 # Deep Copy

>>> print(lst2,id(lst2))----------------[10, 'RS'] 1714679264640

>>> lst1.append("Python")

>>> print(lst1,id(lst1))----------------[10, 'RS', 'Python'] 1714679264640

>>> print(lst2,id(lst2))----------------[10, 'RS', 'Python'] 1714679264640

>>> lst2.insert(1,"NL")

>>> print(lst1,id(lst1))----------------[10, 'NL', 'RS', 'Python'] 1714679264640

>>> print(lst2,id(lst2))----------------[10, 'NL', 'RS', 'Python'] 1714679264640

--------------------------------------------------------------------------------------------------------------------------------------

Slice Based Copy

-------------------------------------------------

>>> lst1=[10,"RS","Python",34.56,True]

>>> print(lst1,id(lst1))------------------------[10, 'RS', 'Python', 34.56, True] 1714679266688

>>> lst2=lst1[::2] # Slice Based Operation

>>> print(lst2,id(lst2))----------------------[10, 'Python', True] 1714679266816

----------------------------

>>> lst1=[10,"RS","Python",34.56,True]

>>> print(lst1,id(lst1))--------------------------[10, 'RS', 'Python', 34.56, True] 1714679264640

>>> lst2=lst1[::] # Slice Based Operation

>>> print(lst2,id(lst2))--------------------------[10, 'RS', 'Python', 34.56, True] 1714679266688

------------------------------------------------------------------------------------------------------------------------------------------

**8) reverse()**

------------------------------------------------------------------------------------------------------------------------------------------

Syntax: listobj.reverse()

=>This Function is used for Obtaining Reverse of Given List of Elements(Front Elements goes to back and Back Elements come come to Front).

------------------------------------

Examples

-------------------------------------

>>> lst=[10,"RS",34.56,"Python"]

>>> print(lst,id(lst))-------------------[10, 'RS', 34.56, 'Python'] 1331814191552

>>> lst.reverse()

>>> print(lst,id(lst))------------------['Python', 34.56, 'RS', 10] 1331814191552

---------------------------------------------------

>>> lst=[10,"RS",34.56,"Python"]

>>> print(lst,id(lst))---------------------[10, 'RS', 34.56, 'Python'] 1331818680512

>>> lst1=lst.reverse() # Most IMP

>>> print(lst,id(lst))--------------------['Python', 34.56, 'RS', 10] 1331818680512

>>> print(lst1)--------------------------None

------------------------------------------------------------------------------------------------------------------------------------------

**9) sort()---Most Imp**

------------------------------------------------------------------------------------------------------------------------------------------

Syntax1: listobj.sort()--------------------->Sorts the Homogeneous Data in ASCENDING Order

Syntax2: listobj.sort(reverse=False)--->Sorts the Homogeneous Data in ASCENDING Order

Syntax2: listobj.sort(reverse=True)---->Sorts the Homogeneous Data in DECENDING Order

--------------------------------------------

>>> lst1=[10,12,-5,23,11,4,16,0,3,11]

>>> print(lst1,id(lst1))----------------------[10, 12, -5, 23, 11, 4, 16, 0, 3, 11] 1331816301056

>>> lst1.sort()

>>> print(lst1,id(lst1))---------------------[-5, 0, 3, 4, 10, 11, 11, 12, 16, 23] 1331816301056

---------------------------------------------------

>>> lst1=[10,12,-5,23,11,4,16,0,3,11]

>>> print(lst1,id(lst1))-----------------[10, 12, -5, 23, 11, 4, 16, 0, 3, 11] 1331818680512

>>> lst2=lst1.sort()

>>> print(lst1,id(lst1))-----------------[-5, 0, 3, 4, 10, 11, 11, 12, 16, 23] 1331818680512

>>> print(lst2)------------------------------None

>>> #---------------------------------------------------------------

>>> lst1=[10,12,-5,23,11,4,16,0,3,11]

>>> print(lst1,id(lst1))------------------[10, 12, -5, 23, 11, 4, 16, 0, 3, 11] 1331816301056

>>> lst1.sort()

>>> print(lst1,id(lst1))------------------[-5, 0, 3, 4, 10, 11, 11, 12, 16, 23] 1331816301056

>>> lst1.reverse()

>>> print(lst1,id(lst1))-------------------[23, 16, 12, 11, 11, 10, 4, 3, 0, -5] 1331816301056

--------------------------OR----------------

>>> lst1=[10,12,-5,23,11,4,16,0,3,11]

>>> print(lst1,id(lst1))------------[10, 12, -5, 23, 11, 4, 16, 0, 3, 11] 1331818680512

>>> lst1.sort()

>>> print(lst1,id(lst1))------------[-5, 0, 3, 4, 10, 11, 11, 12, 16, 23] 1331818680512

>>> lst1=lst1[::-1]

>>> print(lst1,id(lst1))-----------[23, 16, 12, 11, 11, 10, 4, 3, 0, -5] 1331816301056

>>> #----------------------------------------------

>>> lst1=[10,12,-5,23,11,4,16,0,3,11]

>>> print(lst1,id(lst1))-------------[10, 12, -5, 23, 11, 4, 16, 0, 3, 11] 1331818680512

>>> lst1.sort(reverse=True)

>>> print(lst1,id(lst1))-----------[23, 16, 12, 11, 11, 10, 4, 3, 0, -5] 1331818680512

>>> lst1.sort()

>>> print(lst1,id(lst1))-----------[-5, 0, 3, 4, 10, 11, 11, 12, 16, 23] 1331818680512

>>> #---------------------------------------------------

>>> lst1=[10,12,-5,23,11,4,16,0,3,11]

>>> print(lst1,id(lst1))---------------[10, 12, -5, 23, 11, 4, 16, 0, 3, 11] 1331816301056

>>> lst1.sort(reverse=False)

>>> print(lst1,id(lst1))--------[-5, 0, 3, 4, 10, 11, 11, 12, 16, 23] 1331816301056

----------------------------------------------------------

>>> lst1=["Rossum","Biden","Trump","Ritche","Travis","Adam"]

>>> print(lst1)------------------['Rossum', 'Biden', 'Trump', 'Ritche', 'Travis', 'Adam']

>>> lst1.sort()

>>> print(lst1)-------------------['Adam', 'Biden', 'Ritche', 'Rossum', 'Travis', 'Trump']

------------------------

>>> lst1=["Rossum","Biden","Trump","Ritche","Travis","Adam"]

>>> print(lst1)-----------['Rossum', 'Biden', 'Trump', 'Ritche', 'Travis', 'Adam']

>>> lst1.sort(reverse=True)

>>> print(lst1)----------['Trump', 'Travis', 'Rossum', 'Ritche', 'Biden', 'Adam']

----------------------------------------------------

>>> lst1=[10,"RS",34.56,True,2+3j]

>>> print(lst1)-----------------[10, 'RS', 34.56, True, (2+3j)]

>>> lst1.sort()-----------------TypeError: '<' not supported between instances of 'str' and 'int'

------------------------------------------------------------------------------------------------------------------------------------------

**10) index()**

------------------------------------------------------------------------------------------------------------------------------------------

=>Syntax: listobj.index(Value)

=>This Function is used for Finding Index of First Occurence of Specified Value from non-empty list object.

=>If the Specified Value does not exist then we get **ValueError**

=>If we call this Function on empty list then we get **ValueError**

--------------------------------

Examples

---------------------------------------

>>> lst1=[10,"RS",34.56,True,2+3j]

>>> print(lst1)-----------------[10, 'RS', 34.56, True, (2+3j)]

>>> lst1.index(34.56)-----------2

>>> lst1.index("RS")-----------1

>>> lst1.index(2+3j)------------4

**>>> lst1.index(100)---------------ValueError: 100 is not in list**

**>>> [].index(10)-----------------ValueError: 10 is not in list**

**>>> list().index(0)---------------ValueError: 0 is not in list**

>>> [10,20,30,40].index(20)-------1

------------------------------

>>> lst=[10,20,10,30,10,20]

>>> print(lst)--------------[10, 20, 10, 30, 10, 20]

>>> lst.index(10)----------0

>>> [10,10,10,10].index(10)------0

----------------------------------------------------------------------------

NOTE: enumerate()--->This gives Index of Every of Value Object

-------------------

Examples

-------------------

>>> lst=[10,20,30,10,20,40,10,15]

>>> print(lst)-----------[10, 20, 30, 10, 20, 40, 10, 15]

>>> for k,v in enumerate(lst):

... print(k,"--->",v)

...

0 ---> 10

1 ---> 20

2 ---> 30

3 ---> 10

4 ---> 20

5 ---> 40

6 ---> 10

7 ---> 15

>>> for k,v in enumerate(lst):

... if(v==10):

... print(k,"--->",v)

...

0 ---> 10

3 ---> 10

6 ---> 10

>>> s="MISSISSIPPI"

>>> for k,v in enumerate(s):

... print(k,"--->",v)

...

0 ---> M

1 ---> I

2 ---> S

3 ---> S

4 ---> I

5 ---> S

6 ---> S

7 ---> I

8 ---> P

9 ---> P

10 ---> I

>>> for k,v in enumerate(s):

... if(v=="S"):

... print(k,"--->",v)

...

2 ---> S

3 ---> S

5 ---> S

6 ---> S

>>> for k,v in enumerate(s):

... if(v=="I"):

... print(k,"--->",v)

...

1 ---> I

4 ---> I

7 ---> I

10 ---> I

---------------------------------------

>>> for k,v in enumerate("ABRAKADABRA"):

... if(v=="A"):

... print(k,"--->",v)

...

0 ---> A

3 ---> A

5 ---> A

7 ---> A

10 ---> A

>>> for k,v in enumerate("ABRAKADABRA"):

... if(v=="B"):

... print(k,"--->",v)

...

1 ---> B

8 ---> B

------------------------------------------------------------------------------------------------------------------------------------------

**11) count()**

------------------------------------------------------------------------------------------------------------------------------------------=>Syntax: listobj.count(Value)

=>This function is used for Finding Number of occurences of Specified Value

=>If the Specified Value does not exist then we get 0 as count

--------------------------------

Examples

--------------------------------

>>> lst=[10,20,30,10,40,50,10,20]

>>> print(lst)-----------[10, 20, 30, 10, 40, 50, 10, 20]

>>> lst.count(10)----------3

>>> lst.count(20)----------2

>>> lst.count(40)---------1

>>> lst.count(50)--------1

>>> lst.count(500)------0

>>> [].count(10)---------0

>>> list().count(10)------0

>>> list("MISSISSIPPI").count("I")-------4

>>> list("MISSISSIPPI").count("M")------1

>>> list("MISSISSIPPI").count("S")-------4

>>> list("MISSISSIPPI").count("K")------0

>>> list("MISSISSIPPI").count("SIS")-------0

>>> list("MISSISSIPPI")-----------['M', 'I', 'S', 'S', 'I', 'S', 'S', 'I', 'P', 'P', 'I'] ( Reason for the above ANS)

>>> list("MISSISSIPPI").count("SSI")--------0

--------------------

Most Imp

--------------------

>>> list(["121234"]).count(1)

0

>>> list(["121234"])-----------['121234']

**>>> list("121234").count(1)-------0**

>>> list("121234")--------------------['1', '2', '1', '2', '3', '4']

>>> **list("121234").count('1')-----------2**

>>> list("121234").count('12')---------0

------------------------------------------------------------------------------------------------------------------------------------------

12) extend()

------------------------------------------------------------------------------------------------------------------------------------------

=>Syntax: listobj1.extend(listobj2)

=>This Function is used for Merging ( Extending the functionality of listobj1 with the content of listobj2) of the content of listobj1 with listobj2 and result places in listobj1. This Function can merge Two List objects only but not Multiple list objects.

=>To merge Multiple list objects content, we use + Operator

=>Syntax: listobj1=listobj1+ listobj2.....+ listobj-n

-------------------------------------------------------

Examples

-------------------------------------------------------

>>> lst1=[10,20,30,40]

>>> lst2=["RS","TR","KN"]

>>> print(lst1,id(lst1))----------[10, 20, 30, 40] 1331818690112

>>> print(lst2,id(lst2))--------------['RS', 'TR', 'KN'] 1331816301056

>>> lst1.extend(lst2)

>>> print(lst1,id(lst1))-------------[10, 20, 30, 40, 'RS', 'TR', 'KN'] 1331818690112

----------------

>>> lst1=[10,20,30,40]

>>> lst2=["RS","TR","KN"]

>>> print(lst1,id(lst1))------------[10, 20, 30, 40] 1331818690624

>>> print(lst2,id(lst2))------------['RS', 'TR', 'KN'] 1331818690112

>>> lst2.extend(lst1)

>>> print(lst2,id(lst2))----------['RS', 'TR', 'KN', 10, 20, 30, 40] 1331818690112

---------------------

>>> lst1=[10,20,30,40]

>>> lst2=["RS","TR","KN"]

>>> lst3=[1.2,3.4]

>>> lst1.extend(lst2,lst3)----------------TypeError: list.extend() takes exactly one argument (2 given)

To Solve the Above Problem, we use the following

>>> lst1.extend(lst2)

>>> lst1.extend(lst3)

>>> print(lst1)------------[10, 20, 30, 40, 'RS', 'TR', 'KN', 1.2, 3.4]

----------------------------

>>> lst1=[10,20,30,40]

>>> lst2=["RS","TR","KN"]

>>> lst3=[1.2,3.4]

>>> lst1.extend(lst2,lst3)----------------TypeError: list.extend() takes exactly one argument (2 given)

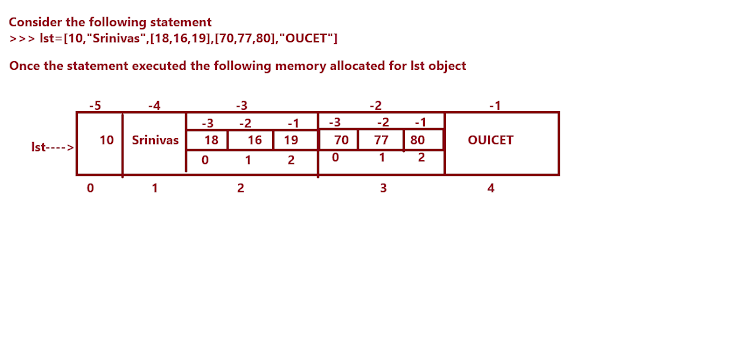
To Solve the Above Problem, we use the following

>>> lst1=lst1+lst2+lst3 # used + Operator for Merging Multiple List objects

>>> print(lst1)--------------[10, 20, 30, 40, 'RS', 'TR', 'KN', 1.2, 3.4]

---------------------------------------------------------------------------------------------------------------------------------------------------------------------------

**Pre-defined Functions in list**  
**Nested OR Inner List**

[](https://drive.google.com/file/d/1kX1ENZrmzqoKBvW6QOMNT38U1953JsH_/view?usp=drive_web&authuser=0" \t "https://classroom.google.com/c/NjM3OTA1MzM3MDkz/p/_blank)

[Memory Management of Inner List.png](https://drive.google.com/file/d/1kX1ENZrmzqoKBvW6QOMNT38U1953JsH_/view?usp=drive_web&authuser=0" \t "https://classroom.google.com/c/NjM3OTA1MzM3MDkz/p/_blank)

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

Nested OR Inner List

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

=>The Process of Defining One List inside of another List is called Nested OR Inner List

=>Syntax: listobj=[ Val1,Val2, [ Val11,Val12...Val1n],[Val21,Val22,...Val2n],........Val-n ]

=>Here [ Val11,Val12...Val1n] represents One Inner OR Nested List

=>Here [ Val21,Val22...Val2n] represents Another Inner OR Nested List

=>Here [ Val1,Val2, [ Val11,Val12...Val1n],[Val21,Val22,...Val2n],........Val-n ]--- called Outer List

=>On Inner List object, we can Perform Both Indexing and Slicing Operations

=>We can also Apply all the pre-defined functions of list on inner list objects.

------------------------------------------------------------------------------------------------------------------------------------------

Examples

------------------------------------------------------------------------------------------------------------------------------------------

>>> lst=[10,"Srinivas",[18,16,19],[70,77,80],"OUCET"]

>>> print(lst,type(lst))----------------[10, 'Srinivas', [18, 16, 19], [70, 77, 80], 'OUCET'] <class 'list'>

>>> for val in lst:

... print(val,type(val),type(lst))

...

10 <class 'int'><class 'list'>

Srinivas <class 'str'><class 'list'>

[18, 16, 19] <class 'list'><class 'list'>

[70, 77, 80] <class 'list'><class 'list'>

OUCET <class 'str'><class 'list'>

>>> lst[0]

10

>>> lst[1]

'Srinivas'

>>> lst[2]

[18, 16, 19]

>>> lst[-3]

[18, 16, 19]

>>> lst[3]

[70, 77, 80]

>>> lst[-2]

[70, 77, 80]

>>> lst[4]

'OUCET'

>>> lst[-1]

'OUCET'

>>> lst[2][1]

16

>>> lst[2][-2]

16

>>> lst[-3][1]

16

>>> lst[-3][-2]

16

>>> lst[3][2]

80

>>> lst[2][1:3]

[16, 19]

>>> lst[2][1:]

[16, 19]

>>> lst[2][-2:]

[16, 19]

>>> lst[3][::2]

[70, 80]

>>> lst[3][0::2]

[70, 80]

>>> lst[3][0:3:2]

[70, 80]

>>> lst[-2][0:3:2]

[70, 80]

>>> lst[-2][-3::2]

[70, 80]

>>> print(lst)

[10, 'Srinivas', [18, 16, 19], [70, 77, 80], 'OUCET']

>>> lst[2][1]=20

>>> print(lst)

[10, 'Srinivas', [18, 20, 19], [70, 77, 80], 'OUCET']

>>> lst[-2][-2]=75

>>> print(lst)

[10, 'Srinivas', [18, 20, 19], [70, 75, 80], 'OUCET']

>>> lst[-2][::2]=[72,75]

>>> print(lst)

[10, 'Srinivas', [18, 20, 19], [72, 75, 75], 'OUCET']

>>> lst[2].append(15)

>>> print(lst)

[10, 'Srinivas', [18, 20, 19, 15], [72, 75, 75], 'OUCET']

>>> lst[-2].insert(1,80)

>>> print(lst)

[10, 'Srinivas', [18, 20, 19, 15], [72, 80, 75, 75], 'OUCET']

>>> lst[-3].sort()

>>> print(lst)

[10, 'Srinivas', [15, 18, 19, 20], [72, 80, 75, 75], 'OUCET']

>>> lst[-2].sort(reverse=True)

>>> print(lst)

[10, 'Srinivas', [15, 18, 19, 20], [80, 75, 75, 72], 'OUCET']

>>> del lst[-2][1:]

>>> print(lst)

[10, 'Srinivas', [15, 18, 19, 20], [80], 'OUCET']

>>> del lst[2][::2]

>>> print(lst)

[10, 'Srinivas', [18, 20], [80], 'OUCET']

>>> lst[2].clear()

>>> print(lst)

[10, 'Srinivas', [], [80], 'OUCET']

>>> lst[-2].remove(80)

>>> print(lst)

[10, 'Srinivas', [], [], 'OUCET']

>>> del lst[2:4]

>>> print(lst)

[10, 'Srinivas', 'OUCET']

>>> lst.insert(2,[15,18,13])

>>> print(lst)

[10, 'Srinivas', [15, 18, 13], 'OUCET']

>>> lst.insert(-1,[20,19,16])

>>> print(lst)

[10, 'Srinivas', [15, 18, 13], [20, 19, 16], 'OUCET']

------------------------------------------------------------------------------------------------------------------------------------------

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

2. tuple ======================================================

Index

--------------------

=>Properties of tuple

=>What is tuple

=>Notations of tuple

=>Types of tuple

=>Operations on tuple

=>Pre-defined Functions in tuple

=>Nested OR Inner tuple

=>Combination tuples with list

=>Programming Examples

------------------------------------------------------------------------------------------------------------------------------------------

Properties of tuple

------------------------------------------------------------------------------------------------------------------------------------------=>'tuple' is one of the Pre-Defined Class and Treated as List Data Type.

=>The purpose of tuple data type is that "To Store Multiple Values of Same Type OR Different Type Or the Both the Types with Unique and Duplicate Values in Single Object . In other words tuple used for storing Constant Values of Multiple Values of Same Type OR Different Type Or the Both the Types with Unique and Duplicate Values in Single Object.

=>To store the Elements OR Values in the object of tuple, we use braces ( ) and the Values separated by comma.

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

=>An object of tuple belongs to IMMUTABLE bcoz tuple object does not allows us to perform Item Assigment.

=>An Object of tuple maintains Insertion Order.

=>In Python Programming, we can create Two Types of tuple objects. They are

1) Empty tuple

2) Non-Empty tuple

------------------------------------

1) Empty tuple

------------------------------------

=>An Empty tuple is one which does not contain any Elements and whose length is 0

=>Syntax: varname=() (OR) varname=tuple()

---------------------------------

2) Non-Empty tuple

------------------------------------

=>A Non-Empty tuple is one which contains Elements and whose length is >0

=>Syntax: varname=(Val1,Val2,.....,Val-n)

OR

varname=tuple(object)

OR

varname=Val1,Val2,.....,Val-n

OR

varname=(Val1,)

------------------------------------------------------------------------------------------------------------------------------------------

Note: The Functionality of tuple is exactly similar to list. But an object of list belongs to MUTABLE where as an object of tuple belongs to IMMUTABLE

------------------------------------------------------------------------------------------------------------------------------------------Examples

---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

>>> t1=(10,20,30,40,10,20,45)

>>> print(t1,type(t1))--------------------(10, 20, 30, 40, 10, 20, 45) <class 'tuple'>

>>> t2=(10,"RS","PYTHON","NL",34.56)

>>> print(t2,type(t2))----------------(10, 'RS', 'PYTHON', 'NL', 34.56) <class 'tuple'>

------------------------------------------

>>> t2=(10,"RS","PYTHON","NL",34.56)

>>> print(t2,type(t2))--------------(10, 'RS', 'PYTHON', 'NL', 34.56) <class 'tuple'>

>>> t2[0]-------------------------------10

>>> t2[-1]------------------------------34.56

>>> t2[0:4]----------------------------(10, 'RS', 'PYTHON', 'NL')

>>> t2[::2]-----------------------------(10, 'PYTHON', 34.56)

>>> t2[::-1]----------------------------(34.56, 'NL', 'PYTHON', 'RS', 10)

----------------------------------------------------

>>> t2=(10,"RS","PYTHON","NL",34.56)

>>> print(t2,type(t2))------------------(10, 'RS', 'PYTHON', 'NL', 34.56) <class 'tuple'>

>>> t2[0]=100----------------TypeError: 'tuple' object does not support item assignment--Immutable

------------------------------------------------------

>>> t1=()

>>> print(t1,type(t1))---------------------() <class 'tuple'>

>>> len(t1)-----------------------------------0

>>> t2=tuple()

>>> print(t2,type(t2))---------------------() <class 'tuple'>

>>> len(t2)-----------------------------------0

>>> t2=(10,"RS","PYTHON","NL",34.56)

>>> print(t2,type(t2))--------------------(10, 'RS', 'PYTHON', 'NL', 34.56) <class 'tuple'>

>>> len(t2)-----------------------------------5

------------------------------------------------------------

>>> s="PYTHON"

>>> print(s,type(s))-------------------------PYTHON <class 'str'>

>>> t=tuple(s) # Converted str object into tuple

>>> print(t,type(t))--------------------------('P', 'Y', 'T', 'H', 'O', 'N') <class 'tuple'>

>>> l1=[10,20,30,40,50]

>>> print(l1,type(l1))----------------------[10, 20, 30, 40, 50] <class 'list'>

>>> t1=tuple(l1) # Converted list object into tuple

>>> print(t1,type(t1))---------------------(10, 20, 30, 40, 50) <class 'tuple'>

>>> l1[0]=100

>>> print(l1,type(l1))--------------[100, 20, 30, 40, 50] <class 'list'>

>>> t1[0]=100----------------------------TypeError: 'tuple' object does not support item assignment

-------------------------------------------------------------------------------------

Most Imp:

----------------

>>> t1=100,"TR",23.45,"PYTHON"

>>> print(t1,type(t1))--------------(100, 'TR', 23.45, 'PYTHON') <class 'tuple'>

----------------

>>> x=(100)

>>> print(x,type(x))-------------------100 <class 'int'>

>>> x=[100]

>>> print(x,type(x))-------------------[100] <class 'list'>

>>> x=(100,)

>>> print(x,type(x))-------------------(100,) <class 'tuple'>

>>> x=("PYTHON")

>>> print(x,type(x))------------PYTHON <class 'str'>

>>> x=("PYTHON",)

>>> print(x,type(x))--------------('PYTHON',) <class 'tuple'>

---------------------------------------------------------------------------------------------------------------------------

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

Pre-defined Function in tuple

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

=>We know that on the object of tuple we can perform Both Indexing and Slicing Operations.

=>Along with these operations, we can also perform other operations by using the following

pre-defined Functions present in tuple.

1)index()

2)count()

-------------------

Examples:

-------------------

>>> t1=(10,"RS",45.67)

>>> print(t1,type(t1))------------(10, 'RS', 45.67) <class 'tuple'>

>>> t1.index(10)---------0

>>> t1.index("RS")------1

>>> t1=(10,"RS",45.67)

>>> print(t1,type(t1))-------(10, 'RS', 45.67) <class 'tuple'>

>>> t1.count(10)-------1

>>> t1.count(100)------0

>>> t1=(10,0,10,10,20,0,10)

>>> print(t1,type(t1))---------(10, 0, 10, 10, 20, 0, 10) <class 'tuple'>

>>> t1.count(10)---------------4

>>> t1.count(0)-----------------2

>>> t1.count(100)--------------0

-----------------------------------------------------

>>> t1=(10,20,30,40,50,10)

>>> print(t1,id(t1),type(t1))---------(10, 20, 30, 40, 50, 10) 2420310634464 <class 'tuple'>

>>> **t2=t1 # Deep Copy Possible but Not Shallow Copy**

>>> print(t2,id(t2),type(t2))----------(10, 20, 30, 40, 50, 10) 2420310634464 <class 'tuple'>

>>> t3=t1 # Deep Copy Possible but Not Shallow Copy

>>> print(t3,id(t3),type(t3))---------(10, 20, 30, 40, 50, 10) 2420310634464 <class 'tuple'>

---------------------------

The Functions not present in tuple

---------------------------

append()

insert()

remove()

clear()

pop(index)

pop()

reverse()

sort()

copy()

extend()

---------------------------------------

**NOTE:- By Using del Operator we can't delete values of tuple object By using Indexing and slicing bcoz tuple object belongs to Immutable but we can delete entire tuple object** .

Examples:

---------------

>>> t1=(10,-34,0,10,23,56,76,21)

>>> print(t1,type(t1))--------------(10, -34, 0, 10, 23, 56, 76, 21) <class 'tuple'>

>>> del t1[0]------TypeError: 'tuple' object doesn't support item deletion

>>> del t1[0:4]----TypeError: 'tuple' object does not support item deletion

>>> del t1 # Here we are removing complete object.

>>> print(t1,type(t1))-----NameError: name 't1' is not defined.

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

MOST IMP:

---------------

**sorted():** This Function is used for Sorting the data of immutable object tuple and gives

the sorted data in the form of list.

=>Syntax: listobj=sorted(tuple object)

--------------------------------------------------------

Examples:

--------------------------------------------------------

>>> t1=(10,23,-56,-1,13,15,6,-2)

>>> print(t1,type(t1))------------(10, 23, -56, -1, 13, 15, 6, -2) <class 'tuple'>

**>>> t1.sort()----------------------AttributeError: 'tuple' object has no attribute 'sort'**

>>> x=sorted(t1)

>>> print(x,type(x))-----------[-56, -2, -1, 6, 10, 13, 15, 23] <class 'list'>

>>> t1=tuple(x) # Converted sorted list into tuple

>>> print(t1,type(t1))---------(-56, -2, -1, 6, 10, 13, 15, 23) <class 'tuple'>

>>> t2=t1[::-1]

>>> print(t2,type(t2))------(23, 15, 13, 10, 6, -1, -2, -56) <class 'tuple'>

OR

>>> t1=(10,-4,12,34,16,-6,0,15)

>>> print(t1,type(t1))---------------------(10, -4, 12, 34, 16, -6, 0, 15) <class 'tuple'>

>>> l1=list(t1)

>>> print(l1,type(l1))-----------------[10, -4, 12, 34, 16, -6, 0, 15] <class 'list'>

>>> l1.sort()

>>> print(l1,type(l1))-------------------[-6, -4, 0, 10, 12, 15, 16, 34] <class 'list'>

>>> t1=tuple(l1)

>>> print(t1,type(t1))---------------(-6, -4, 0, 10, 12, 15, 16, 34) <class 'tuple'>

>>>t1=t1[::-1]

>>> print(t1,type(t1))----------------(34, 16, 15, 12, 10, 0, -4, -6) <class 'tuple'>

========================================x========================================================

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

Nested OR Inner tuple

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

=>The Process of Defining One tuple Inside of Another tuple is called Inner OR Nested tuple.

=>Syntax: tplobj=(Val1,Val2......(Val11,Val12....Val-1n), (Val21,Val22,...Val-2n),....Val-n)

=>Here (Val1,Val2......,....Val-n) is ccalled Outer tuple Eleemnts

=>Here (Val11,Val12....Val-1n) is called Inner tuple Elements

=>Here (Val21,Val22,...Val-2n) is also another Inner tuple Elements.

=>On the Inner tuple Objects, we can also Perform Both Indexing and Slicing Operations.

=>On the Objects Inner tuple, we can apply all the Pre-defined Functions of tuple(index(), count())

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

Examples

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

>>> t1=(10,"Rossum",(17,16,18),(77,78,66),"OUCET")

>>> print(t1,type(t1))------------(10, 'Rossum', (17, 16, 18), (77, 78, 66), 'OUCET') <class 'tuple'>

>>> t1[0]----------------------------10

>>> t1[1]----------------------------'Rossum'

>>> t1[2]----------------------------(17, 16, 18)

>>> t1[3]----------------------------(77, 78, 66)

>>> t1[2][1]------------------------16

>>> t1[-2][-1]----------------------66

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

Possibility 1: List in Tuple

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

=>Syntax: tplobj=(Val1,Val2......[Val11,Val12....Val-1n], [Val21,Val22,...Val-2n],....Val-n)

=>Here (Val1,Val2......,....Val-n) is called Outer tuple Eleemnts

=>Here [Val11,Val12....Val-1n] is called Inner list Elements

=>Here [Val21,Val22,...Val-2n] is also another Inner list Elements.

---------------------------------------

Examples

---------------------------------------

>>> t1=(10,"Rossum",[17,16,18],[77,78,66],"OUCET")

>>> print(t1,type(t1))---------------(10, 'Rossum', [17, 16, 18], [77, 78, 66], 'OUCET') <class 'tuple'>

>>> print(t1[2],type(t1[2]))-----------[17, 16, 18] <class 'list'>

>>> print(t1[3],type(t1[3]))-----------[77, 78, 66] <class 'list'>

>>> t1[2].sort()

>>> print(t1,type(t1))------------------(10, 'Rossum', [16, 17, 18], [77, 78, 66], 'OUCET') <class 'tuple'>

>>> t1[3].sort(reverse=True)

>>> print(t1,type(t1))------------------(10, 'Rossum', [16, 17, 18], [78, 77, 66], 'OUCET') <class 'tuple'>

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

Possibility 2: tuple in list

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

=>Syntax: listobj=[Val1,Val2......(Val11,Val12....Val-1n), (Val21,Val22,...Val-2n),....Val-n]

=>Here [Val1,Val2......,....Val-n] is called Outer list Eleemnts

=>Here (Val11,Val12....Val-1n]) is called Inner tuple Elements

=>Here (Val21,Val22,...Val-2n) is also another Inner tuple Elements.

---------------------------------------

Examples

---------------------------------------

>>> l1=[10,"Rossum",(17,16,18),(77,78,66),"OUCET"]

>>> print(l1,type(l1))-----------------[10, 'Rossum', (17, 16, 18), (77, 78, 66), 'OUCET'] <class 'list'>

>>> l1[1]---------------------------------'Rossum'

>>> print(l1[2],type(l1[2]))---------(17, 16, 18) <class 'tuple'>

>>> print(l1[3],type(l1[3]))---------(77, 78, 66) <class 'tuple'>

==================================x============================================

NOTE:

--------

=>One can define One List in another List

=>One can define One Tuple in another Tuple

=>One can define One List in another Tuple ( tuple of lists)

=>One can define One tuple in another List (list of tuples)

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

Set Category Data Types (Collections Data Types)

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

=>The purpose of Set Category Data Types is that " To store Multiple Values either of Same Type OR Different Type OR Both the Types in single Object with Unique Values (No Duplicates)"

=>In Python Programming, we have Two Data Types in Set Category. They are

1. set (Mutable and Immutable)

2. frozenset (Immutable)

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

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

set =====================================================

Index

----------

=>Properties of set

=>Notation of for set

=>Operations on set

=>Types of Sets

=>Pre-defined functions in set

=>Nested sets

=>Comibination of list, tuple with sets

=>Programming Examples

------------------------------------------------------------------------------------------------------------------------------------------

Properties of set

------------------------------------------------------------------------------------------------------------------------------------------

=>"set" is one of the pre-defined class and treated as Set Data Type.

=>The purpose of set data type is that To store Multiple Values either of Same Type OR Different Type OR Both the Types in single Object with Unique Values (No Duplicates)".

=>To store the values OR Elements in set object, we use Curly Braces { } and Values must separated by comma.

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

=>An object of set never maintains Insertion Order bcoz PVM displays any order of its Possibilities.

=>On the Object of set , we can't Perform Both Indexing and Slicing Operations bcoz set object never maintains Insertion Order.

**=>An Object of set belongs to IMMUTABLE bcoz 'set' object does not support item assignment and set object belongs to MUTABLE bcoz we can add the data to set object at Same Memory Address.**

=>In Python Programming, we can create Two Types of set objects. They are

1) Empty set

2) Non-Empty set

------------------------------------

1) Empty set

------------------------------------

=>An Empty set is one which does not contain any Elements and whose length is 0

=>Syntax: varname=set()

---------------------------------

2) Non-Empty set

------------------------------------

=>A Non-Empty set is one which contains Elements and whose length is >0

=>Syntax: varname={Val1,Val2,.....,Val-n}

OR

varname=set(object)

------------------------------------------------------------------------------------------------------------------------------------------

Examples

------------------------------------------------------------------------------------------------------------------------------------------

>>> s1={10,20,30,40,50,60,20,90,60,10}

>>> print(s1,type(s1))-----------{50, 20, 90, 40, 10, 60, 30} <class 'set'>

>>> print(s1,type(s1))------------{50, 20, 90, 40, 10, 60, 30} <class 'set'>

>>> s2={10,"Rossum",45.67,"Python","NL"}

>>> print(s2,type(s2))------------{'NL', 'Python', 'Rossum', 10, 45.67} <class 'set'>

-------------------------------------------------------

>>> s2={10,"Rossum",45.67,"Python","NL"}

>>> print(s2,type(s2))----------{'NL', 'Python', 'Rossum', 10, 45.67} <class 'set'>

>>> s2[0]---------------------------TypeError: 'set' object is not subscriptable

>>> s2[-1]--------------------------TypeError: 'set' object is not subscriptable

>>> s2[0:3]------------------------TypeError: 'set' object is not subscriptable

-------------------------------------------------------------------

>>> s2={10,"Rossum",45.67,"Python","NL"}

>>> print(s2,type(s2))--------------{'NL', 'Python', 'Rossum', 10, 45.67} <class 'set'>

>>> s2[2]="Travis"---------------------------TypeError: 'set' object does not support item assignment

--------------------------------------

>>> s2={10,"Rossum",45.67,"Python","NL"}

>>> print(s2,type(s2),id(s2))-----------{'NL', 'Python', 'Rossum', 10, 45.67} <class 'set'> 2293106097088

>>> s2.add("HYD")

>>> print(s2,type(s2),id(s2))-------{'NL', 'Python', 'HYD', 'Rossum', 10, 45.67} <class 'set'> 2293106097088

----------------------------------------------------------------

>>> s1=set()

>>> print(s1,type(s1))-----------set() <class 'set'>

>>> len(s1)------------------------0

>>> s2={10,"Rossum",45.67,"Python","NL"}

>>> print(s2,type(s2))----------{'NL', 'Python', 'Rossum', 10, 45.67} <class 'set'>

>>> len(s2)-------------------------5

-------------------------------------------------------------------------------

>>> s="PYTHON"

>>> print(s,type(s))-------------PYTHON <class 'str'>

>>> s1=set(s)

>>> print(s1,type(s1))-----------{'P', 'N', 'O', 'T', 'H', 'Y'} <class 'set'>

>>> s="MISSISSIPPI"

>>> print(s,type(s))--------------MISSISSIPPI <class 'str'>

>>> s1=set(s)

>>> print(s1,type(s1))----------{'S', 'P', 'I', 'M'} <class 'set'>

>>> l1=[10,20,30,40,5010,20,10,20,45]

>>> print(l1,type(l1))--------------[10, 20, 30, 40, 5010, 20, 10, 20, 45] <class 'list'>

>>> s1=set(l1)

>>> print(s1,type(s1))------------{40, 10, 45, 5010, 20, 30} <class 'set'>

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

**Pre-defined functions in set**

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

=>On the object of set, we can perform Various Operations by using functions present in set object.

=>The pre-defined functions of set are

------------------------------------------------------------------------------------------------------------------------------------------

**1.add()**

------------------------------------------------------------------------------------------------------------------------------------------

=>Syntax: setobj1.add(Value)

=>This function is used adding the value to set obejct.

----------------------------

**Examples**

----------------------------

>>> s1={10,"RS"}

>>> print(s1,id(s1))-------------{'RS', 10} 2293106096864

>>> s1.add("PYTHON")

>>> print(s1,id(s1))-------------{'PYTHON', 'RS', 10} 2293106096864

>>> s1.add(34.56)

>>> print(s1,id(s1))-------------{'PYTHON', 'RS', 10, 34.56} 2293106096864

>>> s1.add(True)

>>> print(s1,id(s1))-------------{True, 34.56, 'RS', 10, 'PYTHON'} 2293106096864

>>> s1.add(2+3j)

>>> print(s1,id(s1))-------------{True, 34.56, 'RS', 10, (2+3j), 'PYTHON'} 2293106096864

--------------------------

>>> s1=set()

>>> print(s1,id(s1))----------set() 2293106801376

>>> s1.add(100)

>>> s1.add("Travis")

>>> s1.add(45.67)

>>> print(s1,id(s1))-----------{100, 45.67, 'Travis'} 2293106801376

------------------------------------

>>> s1={10,"RS"}

>>> print(s1,id(s1))----------{'RS', 10} 2293106097088

>>> s1.add(10)

>>> print(s1,id(s1))---------{'RS', 10} 2293106097088

>>> s1.add("RS")

>>> s1.add("RS")

>>> s1.add("RS")

>>> s1.add("RS")

>>> print(s1,id(s1))-----{'RS', 10} 2293106097088

------------------------------------------------------------------------------------------------------------------------------------------

**2) clear()**

------------------------------------------------------------------------------------------------------------------------------------------

=>Syntax: setobj.clear()

=>This Function is used for Removing all the Elements of set

=>When we call this Function on empty set then we get Space OR None as a result

---------------------------

**Examples**

---------------------------

>>> s1={10,20,30,40,10,40,50}

>>> print(s1,type(s1),id(s1))------------{50, 20, 40, 10, 30} <class 'set'> 2293106090144

>>> len(s1)-----------------------------------5

>>> s1.clear()

>>> print(s1,type(s1),id(s1))-----------set() <class 'set'> 2293106090144

>>> len(s1)---------------------------------0

>>> print(s1.clear())---------------------None

>>> set().clear()---------------------------Space

(OR)9

>>> print(set().clear())----------------------None

------------------------------------------------------------------------------------------------------------------------------------------**3) remove()**

------------------------------------------------------------------------------------------------------------------------------------------

=>Syntax: setobj.remove(value)

=>This Function is used for Removing the Value from non-empty set object

=>If the specified values does not exist in set object then we get KeyError

--------------------------------------

**Examples**

--------------------------------------

>>> s1={10,20,30,40,50,60}

>>> print(s1,id(s1))-----------{50, 20, 40, 10, 60, 30} 2293106097088

>>> s1.remove(10)

>>> print(s1,id(s1))-----------{50, 20, 40, 60, 30} 2293106097088

>>> s1.remove(20)

>>> print(s1,id(s1))----------{50, 40, 60, 30} 2293106097088

>>> s1.remove(30)

>>> print(s1,id(s1))----------{50, 40, 60} 2293106097088

>>> s1.remove(200)--------------KeyError: 200

-----------------------------------------------

>>> **set().remove(100)------------KeyError: 100**

**>>> set("PYTHON").remove("K")---KeyError: 'K'**

------------------------------------------------------------------------------------------------------------------------------------------

**4) discard()**

------------------------------------------------------------------------------------------------------------------------------------------

=>Syntax: setobj.discard(Value)

=>This Function is used for Removing the Value from non-empty set object

=>If the specified values does not exist in set object then we never get KeyError

-----------------------------

**Examples**

----------------------------

>>> s1={10,20,30,40,50,60}

>>> print(s1,id(s1))-----------{50, 20, 40, 10, 60, 30} 2293106096864

>>> s1.discard(40)

>>> print(s1,id(s1))----------{50, 20, 10, 60, 30} 2293106096864

>>> s1.discard(30)

>>> print(s1,id(s1))----------{50, 20, 10, 60} 2293106096864

>>> s1.discard(10)

>>> print(s1,id(s1))----------{50, 20, 60} 2293106096864

>>> s1.discard(300)

>>> s1.remove(300)----------KeyError: 300

--------------------

>>> set().discard(100)-------Nothing Removed

>>> set().remove(100)-------KeyError: 100

------------------------------------------------------------------------------------------------------------------------------------------

**5) pop()**

------------------------------------------------------------------------------------------------------------------------------------------=>Syntax: setobj.pop()

=This Function is used for Removing any Arbitrary Element from non-empty set object.

=>When we call this Function on empty set object then we get KeyError

--------------------------------------------------------------------------------------------------------

**Examples1:---without Order of Display---Removed Arbitrary Element**

--------------------------------------------------------------------------------------------------------

>>> s1={10,20,30,40,50,60}

>>> s1.pop()--------------------50

>>> s1.pop()--------------------20

>>> s1.pop()-------------------40

>>> s1.pop()-------------------10

>>> s1.pop()-------------------60

>>> s1.pop()-------------------30

>>> print(s1,id(s1))----------set() 2293106096864

>>> s1.pop()-------------------KeyError: 'pop from an empty set'

>>> set().pop()----------------KeyError: 'pop from an empty set'

>>> set("MISSISSIPPI").pop()--------'S'

-------------------------------------------------------------------------------------------------------

**Examples2---Order of display---Removes First Elements always**

------------------------------------------------------------------------------------------------------

>>> s1={100,200,300,400,500,125,115}

>>> print(s1,id(s1))--------------{400, 115, 100, 500, 200, 300, 125} 2293106097088------Order of display

>>> s1.pop()-----------------------400

>>> print(s1,id(s1))------------{115, 100, 500, 200, 300, 125} 2293106097088

>>> s1.pop()--------------------115

>>> s1.pop()--------------------100

>>> s1.pop()--------------------500

>>> s1.pop()--------------------200

>>> s1.pop()--------------------300

>>> s1.pop()-------------------125

>>> s1.pop()--------------------KeyError: 'pop from an empty set'

------------------------------------------------------------------------------------------------------------------------------------------

**6) isdisjoint()**

------------------------------------------------------------------------------------------------------------------------------------------

=>Syntax: setobj1.isdisjoint(setobj2)

=>This Function returns True Provided setobj1 and setobj2 does not contain common Element(s)

=>This Function returns False Provided setobj1 and setobj2 contains common Element(s)

---------------------------------

Examples

---------------------------------

>>> s1={10,20,30,40}

>>> s2={10,15,25}

>>> s3={15,25,35}

>>> s1.isdisjoint(s3)------True

>>> s1.isdisjoint(s2)------False

>>> s2.isdisjoint(s3)------False

------------------------------

>>> s1.isdisjoint(set())-----True

>>> set().isdisjoint(set())----True

------------------------------------------------------------------------------------------------------------------------------------------

**7) issuperset()**

------------------------------------------------------------------------------------------------------------------------------------------

=>Syntax: setobj1.issuperset(setobj2)

=>This Function returns True provided all the elements of setobj2 present in setobj1 OR setobj1 contains all the elements of

setobj2 otherwise It returns False

---------------

**Examples**

----------------

>>> s1={10,20,30,40}

>>> s2={10,20}

>>> s3={10,25,35,45}

>>> s1.issuperset(s2)----------True

>>> s1.issuperset(s3)----------False

>>> s2.issuperset(s1)---------False

>>> s2.issuperset(s3)---------False

-----------------------

=>>>> s1={10,20,30,40}

>>> s1.issuperset(s1)----True

>>> s1.issuperset(set())---True

>>> set().issuperset(s1)---False

>>> set().issuperset(set())-----True

------------------------------------------------------------------------------------------------------------------------------------------

**8) issubset()**

------------------------------------------------------------------------------------------------------------------------------------------

=>Syntax: setobj1.issubset(setobj2)

=>This Function returns True Provided all the elements of setobj1 are present in setobj2 Otherwise It returns False

------------------

**Examples**

------------------

>>> s1={10,20,30,40}

>>> s2={10,20}

>>> s3={10,25,35,45}

>>> s2.issubset(s1)---------True

>>> s2.issubset(s3)---------False

>>> s2.issubset(s3)---------False

>>> s1.issubset(s2)---------False

-------------------

>>> s1={10,20,30,40}

>>> s1.issubset(set())------False

>>> s1.issubset(s1)--------True

>>> set().issubset(s1)-----True

>>> set().issubset(set())-------True

------------------------------------------------------------------------------------------------------------------------------------------

**9. union()**

------------------------------------------------------------------------------------------------------------------------------------------

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

=>This Function takes all the Unique Values from Both setobj1 and setobj2 and whose result placed in setobj3.

----------------

**Examples**

----------------

>>> s1={10,20,30}

>>> s2={20,40,50}

>>> s3=s1.union(s2)

>>> print(s3,type(s3))-------{50, 20, 40, 10, 30} <class 'set'>

------------------------------------------------------------------------------------------------------------------------------------------

**10. intersection()**

------------------------------------------------------------------------------------------------------------------------------------------

=>Syntax: setobj3=setobj1. intersection(setobj2)

=>This Function takes all the Common Values from Both setobj1 and setobj2 and whose result placed in setobj3.

----------------------

**Examples**

----------------------

>>> s1={10,20,30}

>>> s2={20,40,50}

>>> s3=s1.intersection(s2)

>>> print(s3,type(s3))----------{20} <class 'set'>

>>> s1={10,20,30}

>>> s2={25,40,50}

>>> s3=s1.intersection(s2)

>>> print(s3,type(s3))---------set() <class 'set'>

------------------------------------------------------------------------------------------------------------------------------------------

**11. difference()**

------------------------------------------------------------------------------------------------------------------------------------------

=>Syntax: setobj3=setobj1. difference(setobj2)

=>This Function Removes all the Common Values from Both setobj1 and setobj2 and Takes the remaining Elements from setobj1 and place those Elements in setobj3.

-------------------------

**Examples**

------------------------

>>> s1={10,20,30}

>>> s2={20,40,50}

>>> s3=s1.difference(s2)

>>> print(s3,type(s3))------------{10, 30} <class 'set'>

>>> s3=s2.difference(s1)

>>> print(s3,type(s3))------------{40, 50} <class 'set'>

>>> s3={10,20,30}.difference({45,65,75})

>>> print(s3,type(s3))---------{10, 20, 30} <class 'set'>

>>> s3={45,65,75}.difference({10,20,30})

>>> print(s3,type(s3))-------------{65, 75, 45} <class 'set'>

>>> s3={10,20,30}.difference({10,20,30})

>>> print(s3,type(s3))-------------set() <class 'set'>

------------------------------------------------------------------------------------------------------------------------------------------

**12. symmetric\_difference()**

------------------------------------------------------------------------------------------------------------------------------------------

=>Syntax: setobj3=setobj1. symmetric\_difference(setobj2)

=>This Function Removes all the Common Values from Both setobj1 and setobj2 and Takes the remaining Elements from Both setobj1 and setobj2 and place those Elements in setobj3.

=>Syntax:

**setobj3=setobj1. symmetric\_difference(setobj2)==By Formula eqv to==> setobj1.union(setobj2).difference(setobj1.intersection(setobj2) )**

---------------------------------------

**Examples**

---------------------------------------

>>> s1={10,20,30}

>>> s2={20,40,50}

>>> s3=s1.symmetric\_difference(s2)

>>> print(s3,type(s3))-------------------{40, 10, 50, 30} <class 'set'>

>>> s3=s2.symmetric\_difference(s1)

>>> print(s3,type(s3))-------------------{40, 10, 50, 30} <class 'set'>

-------------------------

>>> s1={10,20,30}

>>> s2={20,40,50}

>>> s3=s1.union(s2).difference(s1.intersection(s2))

>>> print(s3,type(s3))-----{40, 50, 10, 30} <class 'set'>

>>> s3=s2.union(s1).difference(s2.intersection(s1))

>>> print(s3,type(s3))-----{40, 50, 10, 30} <class 'set'>

------------------------------------------------------------------------------------------------------------------------------------------

**13. symmetric\_difference\_update()**

------------------------------------------------------------------------------------------------------------------------------------------

=>Syntax: setobj1. symmetric\_difference\_update(setobj2)

=>This Function Removes all the Common Values from Both setobj1 and setobj2 and Takes the remaining Elements from Both setobj1 and setobj2 and place those Elements in setobj1 itself.

-----------------------------

Examples:

-----------------------------

**>>> s1={10,20,30}**

**>>> s2={20,40,50}**

**>>> s3=s1.symmetric\_difference\_update(s2)**

**>>> print(s3)----------None**

>>> print(s1)----------{40, 10, 50, 30}

>>> print(s2)----------{40, 50, 20}

------------------------------------------------------------------------------------------------------------------------------------------

**14. update()**

------------------------------------------------------------------------------------------------------------------------------------------

=>Syntax: setobj1.update(setobj2)

=>This Function is used for Updating / Merging all the values of setobj2 with setobj1 Values.

----------------------

**Examples**

----------------------

>>> s1={10,20,30}

>>> s2={"RS","TR","DR"}

>>> s3=s1.update(s2)

>>> print(s3)------------None

>>> print(s1)-----------{'DR', 20, 'RS', 'TR', 10, 30}

>>> print(s2)----------{'TR', 'DR', 'RS'}

---------------

>>> s1={10,20,30}

>>> s2={"RS",10,12.34}

>>> s1.update(s2)

>>> print(s1,type(s1))--------{20, 'RS', 10, 12.34, 30} <class 'set'>

------------------------------

>>> s1={10,20,30}

>>> s2={10,20,30}

>>> s1.update(s2)

>>> print(s1)-------------{20, 10, 30}

**>>> s={1,2,3}**

**>>> s1={2,3,8}**

**>>> p=s.update(s1)**

**>>> print(p)--------------None**

================================================x=================================

**Most Special Points---MNC Question--By using Set Functions**

================================================x=================================

=>Let Consider the following Problem

set of cricket players={"Sumanth","Swapnil","Satyam"}

set of tennis players={"Swapnil","Rossum","Travis"}

-----------------------------

**Questions**

-----------------------------

Q1) Find all the Players who are playing all the games--cricket and tennis

Q2) Find all the Players who are playing Both Cricket and Tennis

Q3) Find all the Players who are playing Only Cricket But not Tennis

Q4) Find all the Players who are playing Only Tennis But not Cricket

Q5) Find all the Players who are Exclusively playing Tennis and Cricket

---------------------------------------

**Answers**

---------------------------------------

**Q1) Find all the Players who are playing all the games--cricket and tennis**

>>> cp={"Sumanth","Swapnil","Satyam"}

>>> tp={"Swapnil","Rossum","Travis"}

>>> print(cp,type(cp))------------- {'Sumanth', 'Swapnil', 'Satyam'} <class 'set'>

>>> print(tp,type(tp))--------------- {'Rossum', 'Travis', 'Swapnil'} <class 'set'>

>>> cptp=cp.union(tp)

>>> print(cptp,type(cptp))------ {'Rossum', 'Travis', 'Sumanth', 'Swapnil', 'Satyam'} <class 'set'>

**Q2) Find all the Players who are playing Both Cricket and Tennis**

>>> cp={"Sumanth","Swapnil","Satyam"}

>>> tp={"Swapnil","Rossum","Travis"}

>>> print(cp,type(cp))------------- {'Sumanth', 'Swapnil', 'Satyam'} <class 'set'>

>>> print(tp,type(tp))--------------- {'Rossum', 'Travis', 'Swapnil'} <class 'set'>

>>> bothcptp=cp.intersection(tp)

>>> print(bothcptp,type(bothcptp))-----{'Swapnil'} <class 'set'>

**Q3) Find all the Players who are playing Only Cricket But not Tennis**

>>> cp={"Sumanth","Swapnil","Satyam"}

>>> tp={"Swapnil","Rossum","Travis"}

>>> print(cp,type(cp))------------- {'Sumanth', 'Swapnil', 'Satyam'} <class 'set'>

>>> print(tp,type(tp))--------------- {'Rossum', 'Travis', 'Swapnil'} <class 'set'>

>>> onlycp=cp.difference(tp)

>>> print(onlycp,type(onlycp))-------'Sumanth', 'Satyam'} <class 'set'>

**Q4) Find all the Players who are playing Only Tennis But not Cricket**

>>> cp={"Sumanth","Swapnil","Satyam"}

>>> tp={"Swapnil","Rossum","Travis"}

>>> print(cp,type(cp))------------- {'Sumanth', 'Swapnil', 'Satyam'} <class 'set'>

>>> print(tp,type(tp))--------------- {'Rossum', 'Travis', 'Swapnil'} <class 'set'>

>>> onlytp=tp.difference(cp)

>>> print(onlytp,type(onlytp))----{'Rossum', 'Travis'} <class 'set'>

**Q5) Find all the Players who are Exclusively playing Tennis and Cricket**

>>> cp={"Suman

th","Swapnil","Satyam"}

>>> tp={"Swapnil","Rossum","Travis"}

>>> print(cp,type(cp))------------- {'Sumanth', 'Swapnil', 'Satyam'} <class 'set'>

>>> print(tp,type(tp))--------------- {'Rossum', 'Travis', 'Swapnil'} <class 'set'>

>>> exclcptp=cp.symmetric\_difference(tp)

>>> print(exclcptp,type(exclcptp))-----{'Rossum', 'Travis', 'Sumanth', 'Satyam'} <class 'set'>

===============================================x==================================

**Most Most Special Points---MNC Question--Don't Set Functions--Use Bitwise Operators**

================================================x=================================

**Q1) Find all the Players who are playing all the games--cricket and tennis**

>>> cp={"Sumanth","Swapnil","Satyam"}

>>> tp={"Swapnil","Rossum","Travis"}

>>> print(cp,type(cp))------------- {'Sumanth', 'Swapnil', 'Satyam'} <class 'set'>

>>> print(tp,type(tp))--------------- {'Rossum', 'Travis', 'Swapnil'} <class 'set'>

>>> cptp=cp|tp # Bitwise OR Operator ( | )

>>> print(cptp,type(cptp))------ {'Rossum', 'Travis', 'Sumanth', 'Swapnil', 'Satyam'} <class 'set'>

**Q2) Find all the Players who are playing Both Cricket and Tennis**

>>> cp={"Sumanth","Swapnil","Satyam"}

>>> tp={"Swapnil","Rossum","Travis"}

>>> print(cp,type(cp))------------- {'Sumanth', 'Swapnil', 'Satyam'} <class 'set'>

>>> print(tp,type(tp))--------------- {'Rossum', 'Travis', 'Swapnil'} <class 'set'>

>>> bothcptp=cp&tp # Bitwise AND Operator ( & )

>>> print(bothcptp,type(bothcptp))-----{'Swapnil'} <class 'set'>

**Q3) Find all the Players who are playing Only Cricket But not Tennis**

>>> cp={"Sumanth","Swapnil","Satyam"}

>>> tp={"Swapnil","Rossum","Travis"}

>>> print(cp,type(cp))------------- {'Sumanth', 'Swapnil', 'Satyam'} <class 'set'>

>>> print(tp,type(tp))--------------- {'Rossum', 'Travis', 'Swapnil'} <class 'set'>

>>> onlycp=cp-tp # here - is called Substraction Operator

>>> print(onlycp,type(onlycp))-------'Sumanth', 'Satyam'} <class 'set'>

**Q4) Find all the Players who are playing Only Tennis But not Cricket**

>>> cp={"Sumanth","Swapnil","Satyam"}

>>> tp={"Swapnil","Rossum","Travis"}

>>> print(cp,type(cp))------------- {'Sumanth', 'Swapnil', 'Satyam'} <class 'set'>

>>> print(tp,type(tp))--------------- {'Rossum', 'Travis', 'Swapnil'} <class 'set'>

>>> onlytp=tp-cp # here - is called Substraction Operator

>>> print(onlytp,type(onlytp))----{'Rossum', 'Travis'} <class 'set'>

**Q5) Find all the Players who are Exclusively playing Tennis and Cricket**

>>> cp={"Sumanth","Swapnil","Satyam"}

>>> tp={"Swapnil","Rossum","Travis"}

>>> print(cp,type(cp))------------- {'Sumanth', 'Swapnil', 'Satyam'} <class 'set'>

>>> print(tp,type(tp))--------------- {'Rossum', 'Travis', 'Swapnil'} <class 'set'>

>>> exclcptp=cp^tp # Bitwise XOR Operator ( ^ )

>>> print(exclcptp,type(exclcptp))-----{'Rossum', 'Travis', 'Sumanth', 'Satyam'} <class 'set'>

================================================x=================================

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

**Combination set with set,list and tuple OR Nested Sets**

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

**Case-1 :** It is Not Possible to Define One Set in Another Set bcoz sets are unhashable type(Not Possible

to apply yo Indexing + Not Possible to Modify)

--------------------------------------------------------------------------------------------------------------------------------

**Examples:**

----------------

>>> s1={10,"Rossum",{16,19,18},{77,76,75},"OUCET"}---------TypeError: unhashable type: 'set'

--------------------------------------------------------------------------------------------------------------------------------

**Case-2:**  It is Not Possible to Define One List in Another Set bcoz sets are unhashable type(Not Possible

to apply yo Indexing + Not Possible to Modify)

----------------

**Examples**

----------------

>>> s1={10,"Rossum",[16,19,18],[77,76,75],"OUCET"}-----------TypeError: unhashable type: 'list'

--------------------------------------------------------------------------------------------------------------------------------

Case-3: It is Possible to Define One Tuple in Another Set bcoz tuples are Immutable

----------------------------

**Examples**

----------------------------

>>> s1={10,"Rossum",(16,19,18),(77,76,75),"OUCET"}

>>> print(s1,type(s1))-------{(77, 76, 75), (16, 19, 18), 'OUCET', 'Rossum', 10} <class 'set'>

>>> for val in s1:

... print(val,type(val))

...

(77, 76, 75) <class 'tuple'>

(16, 19, 18) <class 'tuple'>

OUCET <class 'str'>

Rossum <class 'str'>

10 <class 'int'>

--------------------------------------------------------------------------------------------------------------------------------

**Case-4:** It is Possible to Define One Set in Another List bcoz List are mutable and allows us to locate set

objects by using Indices.

------------------------

**Examples**

------------------------

>>> lst=[10,"Rossum",{16,19,18},{77,76,75},"OUCET"]

>>> print(lst,type(lst))----------[10, 'Rossum', {16, 18, 19}, {75, 76, 77}, 'OUCET'] <class 'list'>

>>> print(lst[2],type(lst[2]))---------{16, 18, 19} <class 'set'>

>>> lst[2].add(15)

>>> print(lst)---------[10, 'Rossum', {16, 18, 19, 15}, {75, 76, 77}, 'OUCET']

>>> lst[-2].add(66)

>>> print(lst)--------[10, 'Rossum', {16, 18, 19, 15}, {66, 75, 76, 77}, 'OUCET']

--------------------------------------------------------------------------------------------------------------------------------

**Case-5**: It is Possible to Define One Set in Another tuple bcoz **Tuples are Immutable and allows us to**

**locate set objects by using Indices.**

------------------------

**Examples**

------------------------

>>> tpl=(10,"Rossum",{16,19,18},{77,76,75},"OUCET")

>>> print(tpl,type(tpl))-------------(10, 'Rossum', {16, 18, 19}, {75, 76, 77}, 'OUCET') <class 'tuple'>

>>> print(tpl[2],type(tpl[2]))---------{16, 18, 19} <class 'set'>

>>> tpl[2].add(15)

>>> print(tpl,type(tpl))--------(10, 'Rossum', {16, 18, 19, 15}, {75, 76, 77}, 'OUCET') <class 'tuple'>

>>> tpl[-2].remove(76)

>>> print(tpl,type(tpl))--------(10, 'Rossum', {16, 18, 19, 15}, {75, 77}, 'OUCET') <class 'tuple'>

================================x================================================

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

**frozenset**

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

=>'frozenset' is one of the pre-defined class and treated as set data type.

=>The purpose of frozenset data type is that "To store Multiple Values either Simiar Type or Different

Type or Both the Types in Single Object with Unique Values".

=>The elements of frozenset must be obtained from different objects like set , tuple and list.

**Syntax: frozensetobj=frozenset(set/list/tuple)**

=>An Object of frozenset never maintains Insertion Order bcoz PVM can display any one of the

possibility of elements of frozenset object.

=>On the object of frozenset, we can't perform Indexing and Slicing Operations bcoz frozenset object

never maintains Insertion Order.

=>An object of frozenset belongs to Immutable bcoz frozenset' object does not support item assignment and not possible to modify / Change / add.

=>we can create two types of frozenset objects. They are

a) Empty frozenset

b) Non-Empty frozenset

----------------------------------

**a) Empty frozenset:**

----------------------------------

=>An Empty frozenset is one, which does not contain any elements and whose length is 0

=>Syntax: frozensetobj=frozenset()

--------------------------------------

**b) Non-Empty frozenset:**

-------------------------------------

=>A Non-Empty frozenset is one, which contains elements and 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 ] )

------------------------------------------------------------------------------------------------------------------------------------------

**NOTE: The Functionality of frozenset is exactly similar to set But an object of set belongs to both Mutable(add,remove, pop, discard...etc) and also Immutable in the case of Item assigment Whereas frozenset object belongs to Immutable bcoz neither Possible to perform add,remove, pop, discard...etc nor possible to perform Item assigment**.

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

**Examples**

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

>>> l1={10,20,30,10,20,60,70}

>>> print(l1,type(l1))-----------------{20, 70, 10, 60, 30} <class 'set'>

>>> fs1=frozenset(l1)

>>> print(fs1,type(fs1))--------------frozenset({20, 70, 10, 60, 30}) <class 'frozenset'>

>>> s1={10,"RS",33.33,True}

>>> print(s1,type(s1))---------------{33.33, 10, True, 'RS'} <class 'set'>

>>> fs2=frozenset(s1)

>>> print(fs2,type(fs2))------------frozenset({33.33, 10, True, 'RS'}) <class 'frozenset'>

>>> len(fs2)---------------------------4

>>> fs3=frozenset()

>>> print(fs3,type(fs3))----------frozenset() <class 'frozenset'>

>>> len(fs3)-------------------------0

-------------------------

>>> s1={10,"RS",33.33,True}

>>> print(s1,type(s1))-----------{33.33, 10, True, 'RS'} <class 'set'>

>>> fs2=frozenset(s1)

>>> print(fs2,type(fs2))---------frozenset({33.33, 10, True, 'RS'}) <class 'frozenset'>

>>> fs2[0]-------------------------TypeError: 'frozenset' object is not subscriptable

>>> **fs2[0:3]----------------------TypeError: 'frozenset' object is not subscriptable**

**>>> fs2[0]=23--------------------TypeError: 'frozenset' object does not support item assignment**

---------------------------

>>> del fs2[0]-------------------TypeError: 'frozenset' object doesn't support item deletion

>>> del fs2[0:2]-----------------TypeError: 'frozenset' object does not support item deletion

>>> del fs2 # Possible

>>>print(fs2)-----------------NameError: name 'fs2' is not found

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

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

**Pre-Defined Functions in frozenset**

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

=>frozenset contains the following Functions

a) copy()

b) isdisjoint()

c) issuperset()

d) issubset()

e) union()

f) intersection()

g) difference()

h) symmertic\_difference()

**NOTE:**

-------------

>>> fs1=frozenset({10,20,30,409})

>>> print(fs1,type(fs1),id(fs1))--------frozenset({409, 10, 20, 30}) <class 'frozenset'> 2068835340960

>>> fs2=fs1.copy() # Deep Copy

>>> print(fs2,type(fs2),id(fs2))-----frozenset({409, 10, 20, 30}) <class 'frozenset'> 2068835340960

=>In General, Immutable Object content is Not Possible to copy( in the case of tuple). Where as in the case of frozenset, we are able to copy its content to another frozenset object. Here Original frozenset object and copied frozenset object contains Same Memory Address and Not at all possible to Modify / Change their content.

=>frozenset does not contain the following Functions

a) clear()

b) add()

c) remove()

d) discard()

e) pop()

f) update()

h) symmertic\_difference\_update()

---------------------------------------------------------------------------------------------------

**Examples:**

----------------------------------------------------------------------------------------------------

>>> print(fs1,type(fs1),id(fs1))-------frozenset({50, 20, 70, 40, 10, 60, 30}) <class 'frozenset'> 1558323909504

>>> fs2=fs1.copy()

>>> print(fs2,type(fs2),id(fs2))------frozenset({50, 20, 70, 40, 10, 60, 30}) <class 'frozenset'> 1558323909504

>>> print(fs1)----------frozenset({50, 20, 70, 40, 10, 60, 30})

-------------------

>>> fs1=frozenset({10,20,30,40,50,60,70})

>>> fs2=frozenset((10,20,30))

>>> fs1.issuperset(fs2)---------True

>>> fs2.issuperset(fs1)--------False

>>> fs2.issubset(fs1)----------True

>>> fs1.issubset(fs2)------------False

-----------------------------

>>> fs1=frozenset({10,20,30,40,50,60,70})

>>> fs2=frozenset((100,200,300))

>>> fs3=frozenset((10,2,3))

>>> fs1.isdisjoint(fs2)---------True

>>> fs1.isdisjoint(fs3)----------False

>>> print(fs1)-------------frozenset({50, 20, 70, 40, 10, 60, 30})

>>> print(fs2)-----------frozenset({200, 100, 300})

>>> fs1.union(fs2)----------frozenset({100, 70, 40, 200, 10, 300, 50, 20, 60, 30})

>>> fs1.intersection(fs2)---------frozenset()

>>> fs1.difference(fs2)-----------frozenset({70, 40, 10, 50, 20, 60, 30})

>>> fs2.difference(fs1)--------------frozenset({200, 100, 300})

>>> frozenset({10,20,30,40}).symmetric\_difference(frozenset([10,20,50,60]) )

frozenset({40, 50, 60, 30})

>>> fs1|fs2--------------frozenset({100, 70, 40, 200, 10, 300, 50, 20, 60, 30})

=============================================x====================================

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

**Dict Category Data Type (Collections Data Type)**  ===================================================================

=>"dict" is one of the Pre-Defined Class and treated Dict Category

=>The purpose of dict data type is that "To store (Key,Value) ".

=>In (Key,Value), The values of Key are called Unique where as values of Value may or may not Unique.

=>To store the (Key,value) in the object of dict, we use Curly Braces {} and (Key,Value) separated by comma.

=>Syntax: dictobj={Key1:Val1,Key2:Val2,.......,Key-n:Val-n}

Here Key1,Key2,....Key-n Represents Values of Key

Here Val1,Val2,....Val-n Represents Values of Value

=>An obejct of dict maintains Inserion Order

=>On the object of dict , we can't perform Indexing and Slicing Operations bcoz Values of Key Itself act as Indices.

**=>An object of dict belongs to MUTABLE. But in the dict object, Values of Value always belongs to MUTABLE where as Values of Key always belongs to IMMUTABLE.**

=>In Python Programming, we can create Two Types of dict objects. They are

1) Empty Dict

2) Non-Empty Dict

------------------------------------

**1) Empty Dict**

------------------------------------

=>An Empty Dict is one which does not contain any Elements and whose length is 0

**=>Syntax: varname={ }**

**(OR)**

**varname=dict()**

------------------------------------

**2) Non-Empty Dict**

------------------------------------

=>A Non-Empty Dict is one which contains Elements and whose length is >0

=>Syntax: varname={Key1:Val1,Key2:Val2,.......,Key-n:Val-n}

-------------------------------------------------------------------------

Syntax for adding the (Key,value) to dict object

-------------------------------------------------------------------------

dictobj[Key1]=Val1

dictobj[Key2]=Val2

-------------------------------

-------------------------------

dictobj[Key-n]=Val-n

Here Key1,Key2,....Key-n Represents Values of Key

Here Val1,Val2,....Val-n Represents Values of Value

------------------------------------------------------------------------------------------------------------------------------------------

Examples

------------------------------------------------------------------------------------------------------------------------------------------

>>> d1={10:1.2,20:2.3,30:3.4,40:1.2}

>>> print(d1,type(d1))-------------{10: 1.2, 20: 2.3, 30: 3.4, 40: 1.2} <class 'dict'>

>>> d2={"Python":"RS","C":"DR","Java":"Golsing"}

>>> print(d2,type(d2))-----------------{'Python': 'RS', 'C': 'DR', 'Java': 'Golsing'} <class 'dict'>

>>> d3={"Python":1,"C":2,"Java":3,"C++":4}

>>> print(d3,type(d3))-----------{'Python': 1, 'C': 2, 'Java': 3, 'C++': 4} <class 'dict'>

------------------------------

>>> d1={10:1.2,20:2.3,30:3.4,40:1.2}

>>> print(d1,type(d1))--------{10: 1.2, 20: 2.3, 30: 3.4, 40: 1.2} <class 'dict'>

>>> d1[10]-----------------------1.2

>>> d1[20]-----------------------2.3

>>> d1[30]-----------------------3.4

>>> d1[0]------------------------KeyError: 0

--------------------------------------------------------

>>> d1={10:1.2,10:2.3,10:3.4,10:0.5}

>>> print(d1)-----------{10: 0.5}

--------------------------------------------------------

>>> d1={10:1.2,20:2.3,30:3.4,40:1.2}

>>> print(d1,type(d1))-------------{10: 1.2, 20: 2.3, 30: 3.4, 40: 1.2} <class 'dict'>

>>> len(d1)----------------------------4

>>> d2={}

>>> print(d2,type(d2))------------{} <class 'dict'>

>>> len(d2)---------------------------0

>>> d3=dict()

>>> print(d3,type(d3))------------{} <class 'dict'>

>>> len(d3)---------------------------0

---------------------------------------------------------------------------------------------------------------------

>>> d1={}

>>> print(d1,type(d1),id(d1))-------------{} <class 'dict'> 2382612136768

>>> len(d1)------------------------------------0

>>> d1[10]="PYTHON" # Inserted Entry

>>> d1[20]="C" # Inserted Entry

>>> print(d1,type(d1),id(d1))------------{10: 'PYTHON', 20: 'C'} <class 'dict'> 2382612136768

>>> len(d1)------------------------------------2

>>> d1[30]="Java" # Inserted Entry

>>> d1[40]="HTML" # Inserted Entry

>>> print(d1,type(d1),id(d1))-----------{10: 'PYTHON', 20: 'C', 30: 'Java', 40: 'HTML'} <class 'dict'> 2382612136768

>>> d1[30]="JS" # Modified Entry

>>> print(d1,type(d1),id(d1))-----------{10: 'PYTHON', 20: 'C', 30: 'JS', 40: 'HTML'} <class 'dict'> 2382612136768

==========================================x=======================================

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

**Pre-Defined Functions in dict**

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

=>On the object of dict, we can perform Various Operations by using the pre-defined functions present in dict obejct.

=>The Pre-Defined Functions of dict are given bellow.

------------------------------------------------------------------------------------------------------------------------------------------

**1) clear()**

------------------------------------------------------------------------------------------------------------------------------------------

**=>Syntax: dictobj.clear()**

=>This Functrion is used for Removing all the Elements from dict object

=>When we call this Function on empty dict then we get None or Space

--------------------

Examples

-------------------

>>> d1={10:1.2,20:2.3,30:3.4,40:1.2}

>>> print(d1,type(d1),id(d1))-----------{10: 1.2, 20: 2.3, 30: 3.4, 40: 1.2} <class 'dict'> 2382611955968

>>> len(d1)-----------------------------------4

>>> d1.clear()

>>> print(d1,type(d1),id(d1))------------{} <class 'dict'> 2382611955968

>>> len(d1)-----------------------------------0

>>> print(d1.clear())----------None

------------------------------------------------------------------------------------------------------------------------------------------

**2) pop()**

------------------------------------------------------------------------------------------------------------------------------------------

**=>Syntax: dictobj.pop(Key)**

=>This Function is used for Removing (Key,value) from non-empty dict object

=>If the specified Value of Key does not exist in dict object then we get KeyError

=>When we call this Function on empty dict then we get KeyError

--------------------------------

**Examples**

--------------------------------

>>> d1={10:1.2,20:2.3,30:3.4,40:1.2}

>>> print(d1,type(d1),id(d1))-------------{10: 1.2, 20: 2.3, 30: 3.4, 40: 1.2} <class 'dict'> 2382612136768

>>> d1.pop(30)-------------------------------3.4

>>> print(d1,type(d1),id(d1))------------{10: 1.2, 20: 2.3, 40: 1.2} <class 'dict'> 2382612136768

>>> d1.pop(20)-------------------------------2.3

>>> print(d1,type(d1),id(d1))-------------{10: 1.2, 40: 1.2} <class 'dict'> 2382612136768

>>> **d1.pop(60)-------------------------------KeyError: 60**

>>> {}.pop(100)------------------------------KeyError: 100

>>> dict().pop(10)--------------------------KeyError: 10

------------------------------------------------------------------------------------------------------------------------------------------

**3) popitem()**

------------------------------------------------------------------------------------------------------------------------------------------

**Syntax: dictobj.popitem()**

=>This Function is used for Removing last (Key,value) from non-empty dict object

=>When we call this Function on empty dict then we get KeyError

----------------------

**Examples**

----------------------

>>> d1={10:1.2,20:2.3,30:3.4,40:1.2}

>>> print(d1,type(d1),id(d1))-------------{10: 1.2, 20: 2.3, 30: 3.4, 40: 1.2} <class 'dict'> 2382612086528

>>> d1.popitem()----------------------------(40, 1.2)

>>> print(d1,type(d1),id(d1))-------------{10: 1.2, 20: 2.3, 30: 3.4} <class 'dict'> 2382612086528

>>> d1.popitem()----------------------------(30, 3.4)

>>> print(d1,type(d1),id(d1))-------------{10: 1.2, 20: 2.3} <class 'dict'> 2382612086528

>>> d1.popitem()----------------------------(20, 2.3)

>>> print(d1,type(d1),id(d1))-------------{10: 1.2} <class 'dict'> 2382612086528

>>> d1.popitem()----------------------------(10, 1.2)

>>> print(d1,type(d1),id(d1))--------------{} <class 'dict'> 2382612086528

>>> d1.popitem()----------------------------KeyError: 'popitem(): dictionary is empty'

>>> dict().popitem()-----------------------KeyError: 'popitem(): dictionary is empty'

------------------------------------------------------------------------------------------------------------------------------------------

**4) copy()----Implements Shallow Copy**

------------------------------------------------------------------------------------------------------------------------------------------

Syntax: dictobj2=dictobj1.copy()

=>This Functrion is used for Copying the content of One dict object to another dict object (i.e.dictobj1 into dictobj2)

---------------------------

=>Examples

----------------------------

>>> d2={10:"Python",20:"C"}

>>> d1=d2.copy()

>>> print(d2,id(d2))-----------{10: 'Python', 20: 'C'} 2382612136128

>>> print(d1,id(d1))-----------{10: 'Python', 20: 'C'} 2382614518272

>>> d2[30]="C++"

>>> d1[40]="HTML"

>>> print(d2,id(d2))-------------{10: 'Python', 20: 'C', 30: 'C++'} 2382612136128

>>> print(d1,id(d1))-------------{10: 'Python', 20: 'C', 40: 'HTML'} 2382614518272

------------------------------------------------------------------------------------------------------------------------------------------

**5) get( )--Most Imp**

------------------------------------------------------------------------------------------------------------------------------------------

**Syntax: variablename=dictobj.get(Key)**

**=>This Function is used for Obtaining Value of Value by passing Value of Key**

**=>if the Value of the key does not exist then we get None as a Result**

(OR)

**Syntax: dictobj[Key]**

**=>This Syntax is used for Obtaining Value of Value by passing Value of Key**

**=>if the Value of the key does not exist then we get KeyError**

------------------------------------------------------------------------------------------------------------------------------------------

**Examples**

--------------------------

>>> d1={10:1.2,20:2.3,30:3.4,40:1.2}

>>> print(d1,type(d1),id(d1))--------{10: 1.2, 20: 2.3, 30: 3.4, 40: 1.2} <class 'dict'> 2382612086528

>>> d1.get(10)---------------------------1.2

>>> val=d1.get(10)

>>> print(val)-----------------------------1.2

>>> val=d1.get(20)

>>> print(val)------------------------------2.3

>>> val=d1.get(40)

>>> print(val)-----------------------------1.2

>>> val=d1.get(80)

>>> print(val)-----------------------------None

OR

>>> d1[10]-----------------------------------1.2

>>> d1[80]-----------------------------------KeyError: 80

------------------------------------------------------------------------------------------------------------------------------------------

**6) keys()**

------------------------------------------------------------------------------------------------------------------------------------------

**Syntax: varname=dictobject.keys()**

=>This Function is used for obtaining Values of Key and Placed them in LHS Varname and whose type is <class,dict\_keys>

-------------------------

**Examples**

-------------------------

>>> d1={10:1.2,20:2.3,30:3.4,40:1.2}

>>> print(d1)-----------{10: 1.2, 20: 2.3, 30: 3.4, 40: 1.2}

>>> ks=d1.keys()

>>> print(ks,type(ks))-----------dict\_keys([10, 20, 30, 40]) <class 'dict\_keys'>

>>> for k in ks:

... print(k)

...

10

20

30

40

>>> for k in d1.keys():

... print(k)

...

10

20

30

40

------------------------------------------------------------------------------------------------------------------------------------------

**7) values()**

------------------------------------------------------------------------------------------------------------------------------------------

Syntax: varname=dictobject.values()

=>This Function is used for obtaining Values of Value and Placed them in LHS Varname and whose type is <class,dict\_values>

----------------------------

**Examples**

----------------------------

>>> d1={10:1.2,20:2.3,30:3.4,40:1.2}

>>> print(d1)--------------{10: 1.2, 20: 2.3, 30: 3.4, 40: 1.2}

>>> vs=d1.values()

>>> print(vs,type(vs))------------------dict\_values([1.2, 2.3, 3.4, 1.2]) <class 'dict\_values'>

>>> for val in vs:

... print(val)

...

1.2

2.3

3.4

1.2

>>> for val in d1.values():

... print(val)

...

1.2

2.3

3.4

1.2

------------------------------------------------------------------------------------------------------------------------------------------

**8) items()**

------------------------------------------------------------------------------------------------------------------------------------------

Syntax: varname=dictobject.items()

=>This Function is used for obtaining all (Key,Value) and Placed them in LHS Varname and whose type is <class,dict\_items>

-----------------------

Examples:

-----------------------

>>> d1={10:1.2,20:2.3,30:3.4,40:1.2}

>>> print(d1)-------------{10: 1.2, 20: 2.3, 30: 3.4, 40: 1.2}

>>> its=d1.items()

>>> print(its,type(its))--------dict\_items([(10, 1.2), (20, 2.3), (30, 3.4), (40, 1.2)]) <class 'dict\_items'>

>>> for x in its:

... print(x)

...

(10, 1.2)

(20, 2.3)

(30, 3.4)

(40, 1.2)

>>> for x in d1.items():

... print(x)

...

(10, 1.2)

(20, 2.3)

(30, 3.4)

(40, 1.2)

3

------------------------------------------------------------------------------------------------------------------------------------------

**9) update()**

------------------------------------------------------------------------------------------------------------------------------------------------------------------

Syntax: dictobj1.update(dictobj2)

=>This Function is used for updating / adding / Merging of dictobj2 (key,value) to dictobj1

-----------------------

Examples

-----------------------

>>> d1={10:"Python",20:"Java"}

>>> d2={1:1.2,2:3.4}

>>> print(d1)------------{10: 'Python', 20: 'Java'}

>>> print(d2)------------{1: 1.2, 2: 3.4}

>>> d1.update(d2)

>>> print(d1)---------{10: 'Python', 20: 'Java', 1: 1.2, 2: 3.4}

----------------------

>>> d1={10:"Python",20:"Java"}

>>> d2={1:1.2,20:"C"}

>>> print(d1)---------{10: 'Python', 20: 'Java'}

>>> print(d2)---------{1: 1.2, 20: 'C'}

>>> d1.update(d2)

>>> print(d1)-------{10: 'Python', 20: 'C', 1: 1.2}

----------------------------

>>> d1={10:"Python",20:"Java"}

>>> d2={10:"C",20:"JS"}

>>> print(d1)--------{10: 'Python', 20: 'Java'}

>>> print(d2)--------{10: 'C', 20: 'JS'}

>>> d1.update(d2)

>>> print(d1)---------{10: 'C', 20: 'JS'}

------------------------------------------------------------------------------------------------------------------------------------------

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

Combination of dict with dict,set,tuple and list

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

**Case1: dict in dict --Nested Dict Possible to Define**

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

Examples

------------------

>>> d1={"sno":10,"sname":"RS","IntMarks":{"cm":17,"cpp":16,"os":19},"ExtMarks":{"cm":78,"cpp":77,"os":79},"cname":"CUCET"}

>>> print(d1,type(d1))

{'sno': 10, 'sname': 'RS', 'IntMarks': {'cm': 17, 'cpp': 16, 'os': 19}, 'ExtMarks': {'cm': 78, 'cpp': 77, 'os': 79}, 'cname': 'CUCET'} <class 'dict'>

>>> for its in d1.items():

... print(its)

...

('sno', 10)

('sname', 'RS')

('IntMarks', {'cm': 17, 'cpp': 16, 'os': 19})

('ExtMarks', {'cm': 78, 'cpp': 77, 'os': 79})

('cname', 'CUCET')

>>> for k,v in d1.items():

... print(k,"--->",v)

...

sno ---> 10

sname ---> RS

IntMarks ---> {'cm': 17, 'cpp': 16, 'os': 19}

ExtMarks ---> {'cm': 78, 'cpp': 77, 'os': 79}

cname ---> CUCET

>>> for k,v in d1.items():

... print(k,"--->",v,"-->",type(v),type(d1))

...

sno ---> 10 --><class 'int'><class 'dict'>

sname ---> RS --><class 'str'><class 'dict'>

IntMarks ---> {'cm': 17, 'cpp': 16, 'os': 19} --><class 'dict'><class 'dict'>

ExtMarks ---> {'cm': 78, 'cpp': 77, 'os': 79} --><class 'dict'><class 'dict'>

cname ---> CUCET --><class 'str'><class 'dict'>

>>> d1["IntMarks"]------------------{'cm': 17, 'cpp': 16, 'os': 19}

>>> d1["IntMarks"]["DBMS"]=16

>>> d1["ExtMarks"]["DBMS"]=74

>>> print(d1,type(d1))

{'sno': 10, 'sname': 'RS', 'IntMarks': {'cm': 17, 'cpp': 16, 'os': 19, 'DBMS': 16}, 'ExtMarks': {'cm': 78, 'cpp': 77, 'os': 79, 'DBMS': 74}, 'cname': 'CUCET'} <class 'dict'>

>>> d1["IntMarks"].pop("cm")-------17

>>> d1["ExtMarks"].pop("os")------79

>>> print(d1,type(d1))

{'sno': 10, 'sname': 'RS', 'IntMarks': {'cpp': 16, 'os': 19, 'DBMS': 16}, 'ExtMarks': {'cm': 78, 'cpp': 77, 'DBMS': 74}, 'cname': 'CUCET'} <class 'dict'>

==================================================================================**Case2: set in dict--Poissible**

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

>>> d1={"sno":10,"sname":"RS","IntMarks":{17,19,16},"ExtMarks":{67,77,78},"cname":"CUCET"}

>>> print(d1,type(d1))---{'sno': 10, 'sname': 'RS', 'IntMarks': {16, 17, 19}, 'ExtMarks': {67, 77, 78}, 'cname': 'CUCET'} <class 'dict'>

>>> for k,v in d1.items():

... print(k,"--->",v,"--->",type(v),type(d1))

...

sno ---> 10 ---><class 'int'><class 'dict'>

sname ---> RS ---><class 'str'><class 'dict'>

IntMarks ---> {16, 17, 19} ---><class 'set'><class 'dict'>

ExtMarks ---> {67, 77, 78} ---><class 'set'><class 'dict'>

cname ---> CUCET ---><class 'str'><class 'dict'>

>>> d1["IntMarks"]-------------{16, 17, 19}

>>> d1["IntMarks"].add(15)

>>> d1["ExtMarks"].add(64)

>>> print(d1,type(d1))----{'sno': 10, 'sname': 'RS', 'IntMarks': {16, 17, 19, 15}, 'ExtMarks': {64, 67, 77, 78}, 'cname': 'CUCET'} <class 'dict'>

>>> d1.pop("IntMarks")-------{16, 17, 19, 15}

>>> print(d1,type(d1))--------{'sno': 10, 'sname': 'RS', 'ExtMarks': {64, 67, 77, 78}, 'cname': 'CUCET'} <class 'dict'>

>>> d1["ExtMarks"].clear()------->>> print(d1,type(d1))

{'sno': 10, 'sname': 'RS', 'ExtMarks': set(), 'cname': 'CUCET'} <class 'dict'>

>>> d1.pop("ExtMarks")---------set()

>>> print(d1,type(d1))---{'sno': 10, 'sname': 'RS', 'cname': 'CUCET'} <class 'dict'>

>>> d1["IntMarks"]={17,16,15}

>>> print(d1,type(d1))-------{'sno': 10, 'sname': 'RS', 'cname': 'CUCET', 'IntMarks': {16, 17, 15}} <class 'dict'>

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

**Case3: tuple in dict--Possible**

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

>>> d1={"sno":10,"sname":"RS","IntMarks":(17,19,16),"ExtMarks":(67,77,78),"cname":"CUCET"}

>>> print(d1,type(d1))----{'sno': 10, 'sname': 'RS', 'IntMarks': (17, 19, 16), 'ExtMarks': (67, 77, 78), 'cname': 'CUCET'} <class 'dict'>

>>> for k,v in d1.items():

... print(k,"--->",v,"--->",type(v),type(d1))

...

sno ---> 10 ---><class 'int'><class 'dict'>

sname ---> RS ---><class 'str'><class 'dict'>

IntMarks ---> (17, 19, 16) ---><class 'tuple'><class 'dict'>

ExtMarks ---> (67, 77, 78) ---><class 'tuple'><class 'dict'>

cname ---> CUCET ---><class 'str'><class 'dict'>

>>> d1["IntMarks"]-----------(17, 19, 16)

>>> d1["IntMarks"][::2]------(17, 16)

>>> d1["ExtMarks"]----------(67, 77, 78)

>>> d1["ExtMarks"]=tuple(sorted(d1["ExtMarks"])[::-1])

>>> print(d1,type(d1))----{'sno': 10, 'sname': 'RS', 'IntMarks': (17, 19, 16), 'ExtMarks': (78, 77, 67), 'cname': 'CUCET'} <class 'dict'>

==================================================================================**Case4: list in dict--Possible**

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

>>> d1={"sno":10,"sname":"RS","IntMarks":[17,19,16],"ExtMarks":[67,77,78],"cname":"CUCET"}

>>> print(d1,type(d1))---{'sno': 10, 'sname': 'RS', 'IntMarks': [17, 19, 16], 'ExtMarks': [67, 77, 78], 'cname': 'CUCET'} <class 'dict'>

>>> for k,v in d1.items():

... print(k,"--->",v,"--->",type(v),type(d1))

...

sno ---> 10 ---><class 'int'><class 'dict'>

sname ---> RS ---><class 'str'><class 'dict'>

IntMarks ---> [17, 19, 16] ---><class 'list'><class 'dict'>

ExtMarks ---> [67, 77, 78] ---><class 'list'><class 'dict'>

cname ---> CUCET ---><class 'str'><class 'dict'>

>>> d1["IntMarks"].insert(1,15)

>>> d1["ExtMarks"].insert(-1,55)

>>> print(d1,type(d1))---{'sno': 10, 'sname': 'RS', 'IntMarks': [17, 15, 19, 16], 'ExtMarks': [67, 77, 55, 78], 'cname': 'CUCET'} <class 'dict'>

>>> d1["IntMarks"].sort()

>>> d1["ExtMarks"].sort(reverse=True)

>>> print(d1,type(d1))----{'sno': 10, 'sname': 'RS', 'IntMarks': [15, 16, 17, 19], 'ExtMarks': [78, 77, 67, 55], 'cname': 'CUCET'} <class 'dict'>

>>> d1["ExtMarks"].insert(-1,[1.2,2.3])

>>> print(d1,type(d1))--{'sno': 10, 'sname': 'RS', 'IntMarks': [15, 16, 17, 19], 'ExtMarks': [78, 77, 67, [1.2, 2.3], 55], 'cname': 'CUCET'} <class 'dict'>

>>> d1["IntMarks"].insert(2,(1.2,2.3))

>>> print(d1,type(d1))---{'sno': 10, 'sname': 'RS', 'IntMarks': [15, 16, (1.2, 2.3), 17, 19], 'ExtMarks': [78, 77, 67, [1.2, 2.3], 55], 'cname': 'CUCET'} <class 'dict'>

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

**Case5: dict in set--------Not Possible**

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

>>> s1={10,"RS",{"C":15,"CPP":14,"OS":17},"OUCET"}----------TypeError: unhashable type: 'dict'

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

**Case6: dict in tuple--Possible**

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

>>> tpl=(10,"RS",{"C":15,"CPP":14,"OS":17},"OUCET")

>>> print(tpl,type(tpl))-----(10, 'RS', {'C': 15, 'CPP': 14, 'OS': 17}, 'OUCET') <class 'tuple'>

>>> for val in tpl:

... print(val,"--->",type(val),type(tpl))

...

10 ---><class 'int'><class 'tuple'>

RS ---><class 'str'><class 'tuple'>

{'C': 15, 'CPP': 14, 'OS': 17} ---><class 'dict'><class 'tuple'>

OUCET ---><class 'str'><class 'tuple'>

>>> tpl[2]["DBMS"]=16

>>> print(tpl,type(tpl))-----(10, 'RS', {'C': 15, 'CPP': 14, 'OS': 17, 'DBMS': 16}, 'OUCET') <class 'tuple'>

>>> for k,v in tpl[2].items():

... print(k,"-->",v)

...

C --> 15

CPP --> 14

OS --> 17

DBMS --> 16

>>> del tpl[2]-----------TypeError: 'tuple' object doesn't support item deletion

>>> tpl[2].pop("OS")------17

>>> print(tpl,type(tpl))----(10, 'RS', {'C': 15, 'CPP': 14, 'DBMS': 16}, 'OUCET') <class 'tuple'>

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

**Case7: dict in list----Possible**

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

>>> lst=[10,"RS",{"C":15,"CPP":14,"OS":17},"OUCET"]

>>> print(lst,type(lst))-------------[10, 'RS', {'C': 15, 'CPP': 14, 'OS': 17}, 'OUCET'] <class 'list'>

>>> for val in lst:

... print(val,"--->",type(val),type(lst))

...

10 ---><class 'int'><class 'list'>

RS ---><class 'str'><class 'list'>

{'C': 15, 'CPP': 14, 'OS': 17} ---><class 'dict'><class 'list'>

OUCET ---><class 'str'><class 'list'>

>>> lst.pop(2)---------{'C': 15, 'CPP': 14, 'OS': 17}

>>> print(lst,type(lst))------[10, 'RS', 'OUCET'] <class 'list'>

>>> lst.insert(-1,{"C":15,"CPP":14})

>>> print(lst,type(lst))----[10, 'RS', {'C': 15, 'CPP': 14}, 'OUCET'] <class 'list'>

>>> lst[2]["OS"]=15

>>> print(lst,type(lst))-----[10, 'RS', {'C': 15, 'CPP': 14, 'OS': 15}, 'OUCET'] <class 'list'>

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

**Special Point1**

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

>>> d1={10:1.2,20:1.3,30:4.5,40:1.2}

>>> print(d1,type(d1))

{10: 1.2, 20: 1.3, 30: 4.5, 40: 1.2} <class 'dict'>

>>> for k in d1.keys():

... print(k)

...

10

20

30

40

>>> for k in d1.values():

... print(k)

...

1.2

1.3

4.5

1.2

>>> for k in d1.items():

... print(k)

...

(10, 1.2)

(20, 1.3)

(30, 4.5)

(40, 1.2)

>>> for k in d1: # Most Imp

... print(k)

...

10

20

30

40

>>> for k in d1: # Most Imp

... print(k,"--->",d1[k])

...

10 ---> 1.2

20 ---> 1.3

30 ---> 4.5

40 ---> 1.2

>>> for k in d1:

... print(k,"--->",d1.get(k))

...

10 ---> 1.2

20 ---> 1.3

30 ---> 4.5

40 ---> 1.2

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

**Special Point2**

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

>>> lst=[(10, 1.2), (20, 1.3), (30, 4.5), (40, 1.2)]

>>> print(lst,type(lst))--------------[(10, 1.2), (20, 1.3), (30, 4.5), (40, 1.2)] <class 'list'>

>>> d=dict(lst)

>>> print(d,type(d1))---------------{10: 1.2, 20: 1.3, 30: 4.5, 40: 1.2} <class 'dict'>

-----------------

>>> lst=[[10, 1.2], [20, 1.3], [30, 4.5], [40, 1.2]]

>>> print(lst,type(lst))---[[10, 1.2], [20, 1.3], [30, 4.5], [40, 1.2]] <class 'list'>

>>> d=dict(lst)

>>> print(d,type(d1))----{10: 1.2, 20: 1.3, 30: 4.5, 40: 1.2} <class 'dict'>

-----------------

>>> lst=((10, 1.2), (20, 1.3), (30, 4.5), (40, 1.2))

>>> print(lst,type(lst))------((10, 1.2), (20, 1.3), (30, 4.5), (40, 1.2)) <class 'tuple'>

>>> d=dict(lst)

>>> print(d,type(d1))---------{10: 1.2, 20: 1.3, 30: 4.5, 40: 1.2} <class 'dict'>

-----------------

>>> lst=[{10, 1.2}, (20, 1.3), [30, 4.5], [40, 1.2]]

>>> print(lst,type(lst))-------[{1.2, 10}, (20, 1.3), [30, 4.5], [40, 1.2]] <class 'list'>

>>> d=dict(lst)

>>> print(d,type(d1))----------{1.2: 10, 20: 1.3, 30: 4.5, 40: 1.2} <class 'dict'>

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

**Special Point3**

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

>>> l1=["a1","b2","c3"]

>>> print(l1,type(l1))--------['a1', 'b2', 'c3'] <class 'list'>

>>> d1=dict(l1)

>>> print(d1,type(d1))------{'a': '1', 'b': '2', 'c': '3'} <class 'dict'>

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

**Special Point4**

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

>>> l1=["a1","a2","a3"]

>>> print(l1,type(l1))-----------------['a1', 'a2', 'a3'] <class 'list'>

>>> d1=dict(l1)

>>> print(d1,type(d1))---------------{'a': '3'} <class 'dict'>

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

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

**NoneType Category Data Type**

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

=>" NoneType" is one of the pre-defined class

=>None is the Keyword and Treated as Value of NoneType Data Type

=>None is not False, Space, 0 and None is Nothing

=>We can't create an object of NoneType bcoz It contains Single Value

--------------------------------------------------------

Examples

--------------------------------------------------------

>>> a=None

>>> print(a,type(a))--------------None <class 'NoneType'>

>>> None==0----------------------False

>>> None==""--------------------False

>>> None==False----------------False

>>> None==None----------------True

>>> **n=NoneType()------------------------NameError: name 'NoneType' is not defined**

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

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

**Number of Approaches to develop the Programs in Python**  ====================================================================

---------------------------------

**Definition of Program**

---------------------------------

**=>Set of Optimized Instructions meant for solving a real Time Task is called Program.**

**=>The Meaning of Optimized Instructions is That they takes Less Memory space (Space Complexity) and Less Execution Time (Time Complexity)**

=>The purpose of writing Program is that "To Solve OR Implement any Real Time Task".

=>When we write set of Optimized Instructions by using Python Language and those statements must saved on some file name with an extension .py (FileName.py)

------------------------------------------------------------------------------------------------------------------------------------------

=>In Python Environment, we can develop the program in two ways. They are

1. By using Interactive Approach

2. By using Batch Mode Approach

------------------------------------------------------------------------------------------------------------------------------------------

**1. By using Interactive Approach**

------------------------------------------------------------------------------------------------------------------------------------------

=>In this Mode of develop, Python Programmer can Issue One statement at a time and gets One Output at a time.

=>Interactive Approach is most useful to test One Instruction at a time But not useful to write group of instructions bcoz this environment is not allowing us to re-use the code in part of program and not allowing to save the code.

------------------------------------

**=>Examples Softwares:**

------------------------------------

1) Python Command Prompt

2) Python IDLE Shell (Installation of Python Software---www.python.org)

Examples:

--------------------

>>> a=10

>>> b=20

>>> c=a+b

>>> print(a)------10

>>> print(b)------20

>>> print(c)-----30

------------------------------------------------------------------------------------------------------------------------------------------

**2. By using Batch Mode Approach**

------------------------------------------------------------------------------------------------------------------------------------------

=>In this mode of development, Python Programmer writes or Develops Group of Optimized Instructions (batch) and Save that group of Optimized Instructions on some file name with an extension .py (FileName.py).

=>Hence All Program must be developed with Batch Mode Approach only.

------------------------------------

=**>Examples Softwares:**

------------------------------------

1) Python IDLE Shell (Installation of Python Software---www.python.org)

--------------------------

Third Pary IDES

--------------------------

2) PyCharm

3) Jupiter Note Book

4) VS Code

5) Spider

6) Google Clab

7) Sublime Text

8) Edit Plus...etc

NOTE: To run any Python Program from Windows Command Prompt, we use a Tool called "python"(Present in C:\Users\KVR\AppData\Local\Programs\Python\Python312 folder).

Syntax: python FileName.py

(OR)

Syntax: py FileName.py

Examples:

--------------- e:\KVR-PYTHON-11am\batch mode> python sumex3.py

(OR)

e:\KVR-PYTHON-11am\batch mode> py sumex3.py

------------------------------------------------------------------------------------------------------------------------------------------

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

**Displaying the Result of Python Program on the console (Monitor)**

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

=>To display the result of python program on the console, we use a pre-defined function called print().

=>In otherwords, print() is used for displying the result of python program on the console.

=>print() can be used in 5 ways. They are

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Syntax-1: print(Val1)

(OR)

print(Val1,Val2,...Val-n)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*=>This Syntax is used for displaying Either Single Value or Multiple Values.

-----------------

**Examples**

------------------

>>> a=10

>>> print(a)----------10

------------------------

>>> a=10

>>> b=20

>>> c=a+b

>>> print(a)---------10

>>> print(b)---------20

>>> print(c)---------30

>>> print(a,b,c)-----10 20 30

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***Syntax-2:** print(Msg1)

(OR)

print(Msg1,Msg2,.....,Msg-n)

(OR)

print(Msg1+Msg2+....+Msg-n)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*=>Here Msg1,Msg2,....Msg-n Represents str data.

=>This Syntax is used for displying Str Data on the console

**Examples:**

-----------------

>>> print("HelloPython")-----------------HelloPython

>>> print('HelloPython')------------------HelloPython

>>> print("""HelloPython""")-----------HelloPython

>>> print('''HelloPython''')---------------HelloPython

----------------------

**>>> print("Hello","Python")----------Hello Python**

>>> print("Hello","Python","World")-----Hello Python World

>>> print("Hello","""Python""",'World')------Hello Python World

------------------------

>>> print("Hello"+"Python"+"World")---------HelloPythonWorld

>>> print("Hello"+""+"Python"+' '+"World")---Hello Python World

**>>> print("Python"+10)--------------TypeError: can only concatenate str (not "int") to str**

>>> print("Python"+str(10) )------Python10

----------------------

>>> print("Python"+3.12)----------------TypeError: can only concatenate str (not "float") to str

>>> print("Python"+str(3.12))---------Python3.12

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***Syntax-3:**  print(Message Cum Value)

(OR)

print(Value Cum Message)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*=>This Syntax display Value cum message OR Message Cum Value.

-------------------------------

**Examples**

-------------------------------

>>> a=10

>>> print("Value of a=",a)----------Value of a= 10

>>> print(a," is the value of a")----10 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**

>>> print("Sum of "+str(a)+" and "+str(b)+"="+str(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

>>> print("Sum of ",a,",",b," and ",c,"=",d)-----Sum of 10 , 20 and 30 = 60

>>> print("Sum of "+str(a)+","+str(b)+" and "+str(c)+"="+str(d))----Sum of 10,20 and 30=60

>>> print("Sum of "+str(a)+str(",")+str(b)+" and "+str(c)+"="+str(d))-----Sum of 10,20 and 30=60

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Syntax-4: print(Message Cum Value with format())**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

=>This Syntax display Value cum message OR Message Cum Value with format()

--------------------------------

**Examples**

--------------------------------

>>> a=10

>>> print("Val of a={}".format(a))-------Val of a=10

>>> print("{} is the value of a".format(a))-----10 is the value of a

-------------

>>> a=10

>>> b=20

>>> c=a+b

>>> print("Sum of {} and {}={}".format(a,b,c))--------Sum of 10 and 20=30

------------

>>> a=10

>>> b=20

>>> c=30

>>> d=a+b+c

>>> print("sum of {},{} and {}={}".format(a,b,c,d))----sum of 10,20 and 30=60

>>> print("sum({},{},{})={}".format(a,b,c,d))----sum(10,20,30)=60

>>> print("{}+{}+{}={}".format(a,b,c,d))---10+20+30=60

-------------------------

>>> a=10

>>> b=2

>>> print("{}+{}={}, {}-{}={}, {}\*{}={}".format(a,b,a+b,a,b,a-b,a,b,a\*b))----10+2=12, 10-2=8, 10\*2=20

--------------------------

>>> sno=10

>>> sname="Rossum"

>>> print("My Number is {} and Name is {}".format(sno,sname))-----My Number is 10 and Name is Rossum

-------------------

>>> print("My Number is {} and Name is '{}' ".format(sno,sname))----My Number is 10 and Name is 'Rossum'

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***Syntax-5:** print(Message Cum Value with format specifiers)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*=>This Syntax display Value cum message OR Message Cum Value with format specifiers.

=>In Python Programming, we have the following Format Specifiers.

**%d is used for Displying Integer Data**

**%f is used for Displying float Data**

**%s is used for Displying str Data**

=>**NOTE:** If we come across the Otherthan int,float and str, we convert Other Type of Values into str type by using str(object)

and use %s format specifier.

-------------------------------

**Examples**

-------------------------------

>>> a=10

>>> print("Val of a=%d" %a)---------Val of a=10

>>> 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(%d,%d)=%d" %(a,b,c))----sum(10,20)=30

>>> sno=10

>>> sname="Rossum"

>>> marks=23.45

>>> print("my Number is %d and Name is '%s' and Marks=%f" %(sno,sname,marks))---my Number is 10 and Name is 'Rossum'

and Marks=23.450000

------------------------------------

>>> a=10

>>> b=2.3

>>> c=a+b

>>> print("Sum(%d,%f)=%f" %(a,b,c))------Sum(10,2.300000)=12.300000

>>> print("Sum(%d,%0.2f)=%f" %(a,b,c))----Sum(10,2.30)=12.300000

>>> print("Sum(%d,%0.1f)=%f" %(a,b,c))------Sum(10,2.3)=12.300000

>>> print("Sum(%f,%0.1f)=%f" %(a,b,c))--------Sum(10.000000,2.3)=12.300000

>>> print("Sum(%0.2f,%0.2f)=%0.2f" %(a,b,c)-----Sum(10.00,2.30)=12.30

----------------------------

>>> lst=[10,"RS",1.2,True]

>>> print("content of list=",lst)---------------content of list= [10, 'RS', 1.2, True]

>>> print("content of list={}".format(lst))--------content of list=[10, 'RS', 1.2, True]

>>> print("content of list=%d"%lst)--------TypeError: %d format: a real number is required, not list

>>> print("content of list=%s"%str(lst))------content of list=[10, 'RS', 1.2, True]

>>> d1={10:1.2,20:2.3,30:1.2}

>>> print("content of dict=%s"%str(d1))-----content of dict={10: 1.2, 20: 2.3, 30: 1.2}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Syntax-6: print(Value,end=" Delimeter")**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

=>This Syntax displays list of values in same line (Horizontally)

-----------------------------

**Examples**

-----------------------------

>>> r=range(10,16)

>>> for val in r:

... print(val)

...

10

11

12

13

14

15

>>> for val in r:

... print(val,end="")-----------10 11 12 13 14 15

>>> for val in r:

... print(val,end="-->") ------------- 10-->11-->12-->13-->14-->15-->

>>> for val in r:

... print(val,end=",")-------10,11,12,13,14,15,

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

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

**Reading the Data OR Input Dynamically from the Key Board**

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

=>In Python Programming, for Reading the Data OR Input Dynamically from the Key Board, we have 2 Pre-Defined functions. They are

1. input()

2. input(Message)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**1. input()**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Syntax: varname=input()

=>This Function is used for Reading any type of Value from Key Board in the form of str and placed in LHS Varname and whose type is <class, 'str'>

=>Programatically, to convert str type value into Other Type values, we use Type Casting Functions.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**2. input(Message)**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Syntax: varname=input(Message)

=>Here Message is one of the str type and it is treated as User-Prompting Message

=>This Function is used for Reading any type of Value from Key Board in the form of str and placed in LHS Varname and whose

type is <class, 'str'> and additionally this function gives User-Prompting Message

=>Programatically, to convert str type value into Other Type values, we use Type Casting Functions.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**#DataReadEx1.py**

print("Enter Any Value:")

a=input()

print(a,type(a))

b=float(a)

print(b,type(b))

**#DataReadEx2.py**

a=input("Enter Any Value: ")

print(a,type(a))

b=float(a)

print(b,type(b))

**#Program for accepting two numerical and find their product**

**#DataReadEx3.py**

x=input("Enter First Value:")

y=input("Enter Second Value:")

#Convert str type into float OR int

a=float(x)

b=float(y)

c=a\*b

print("product({},{})={}".format(a,b,c))

**#Program for accepting two numerical and find their product**

**#DataReadEx4.py**

a=float(input("Enter First Value:"))

b=float(input("Enter Second Value:"))

c=a\*b

print("product({},{})={}".format(a,b,c))

**#Program for accepting two numerical and find their product**

**#DataReadEx5.py**

print("Enter Two Values:")

x=float(input())

y=float(input())

print("product({},{})={}".format(x,y,x\*y))

print("-----------OR--------------------------")

print("product({},{})={}".format(x,y,round(x\*y,2)))

print("-----------OR--------------------------")

print("product(%0.2f,%0.2f)=%0.2f" %(x,y,x\*y))

**#DataReadEx6.py**

print("Enter Two Values:")

print("Product={}".format(float(input())\*float(input())))

**#DataReadEx7.py**

print("Product={}".format(float(input("Enter First Value:"))\*float(input("Enter Second Value:"))))

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

**Operators and Expressions in Python**

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

=>An Operator is a Symbol which is used to perform Some Operations on the Data.

=>If any Operator Connects with Two Or More Values OR Objects OR Operands OR Variables then it is Called Expression.

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

1. Arithmetic Operators

2. Assigment Operator

3. Relational Operators( Comparision Operators )

4. Logical Operators ( Comparision Operators )

5. Bitwise Operators--Most Imp

6. Membership Operators

a) in operator

b) not in operator

7. Identity Operators

a) is operator

b) is not operator

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

**NOTE-1:** Python does not Support Unary Operators Like ++ and - -

**NOTE-2:** Python does not Support Ternary Operator of C, C++, Java, and C#.net ( ? : )

------------------------------------

**NOTE-3:** Python Supports Short Hand Operators

**NOTE-4:** Python Has Its Own Ternary Operator ------- if... else Operator

-------------------------------------

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

**1. Arithmetic Operators**

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

=>The purpose of Arithmetic Operators is that "To perform Arithmetic Operations such as Addition, Substraction,

Multiplication..etc".

=>If any Arithmetic Operator Connects with Two Or More Values OR Objects OR Operands OR Variables then it is Called Arithmetic Expression.

=>In Python Programming, we have 7 Types of Arithmetic Operators. They are given in the following Table.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

SLNO SYMBOL MEANING EXAMPLES

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 1. .1 + Addition print(10+20)-----30

print("10"+"20")---1020

2. - Substraction print(10-20)----- -10

print("10"-"20")----- TypeError

3. \* Multiplication print(10\*20)---------200

print(""PYTHON"\*2)---- PYTHONPYTHON"--here \* is called Repetation Operator

4. / Division print(10/3)--------- 3.3333333333333335

(Float Quoteint)

5 // Floor Division print(10//3)--------3

(Integer Quoteint)

6. % Modulo Division print(10%3)------- 1

(Remainder)

7. \*\* Exponentiation print(10\*\*2)-------100

(Power )

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***NOTE1:** >>> print(10/3)---------3.3333333333333335

>>> print(10//3)--------3

**NOTE2:** >>> print(10.0/3.0)----3.3333333333333335

>>> print(10.0//3.0)----3.0

**NOTE3:** >>> print(10.0/3)-------3.3333333333333335

>>> print(10.0//3)------3.0

**NOTE4:** >>> print(10/3.0)-------3.3333333333333335

>>> print(10//3.0)------3.0

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

**#program for Demonstrating arithmetic Operators**

**#ArithmeticOpEx1.py**

a=int(input("Enter Value of a:"))

b=int(input("Enter Value of b:"))

print("\*"\*50)

print("Arithmetic Operators")

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("\tFloor Div({},{})={}".format(a,b,a//b))

print("\tModulo Div({},{})={}".format(a,b,a%b))

print("\tExpo({},{})={}".format(a,b,a\*\*b))

print("\*"\*50)

**#Program for cal square root of a given number**

**#ArithmeticOpEx1.py**

n=float(input("Enter any Number for cal square root:"))

res=n\*\*0.5

print("SquareRoot({})={}".format(n,res))

print("---------------OR--------------------------------")

print("SquareRoot(%0.3f)=%0.3f" %(n,res))

print("---------------OR--------------------------------")

print("SquareRoot({})={}".format(n,round(res,3)))

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

**2. Assingment Operator**

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

=>The purpose of assignment operator is that " To assign or transfer Right Hand Side (RHS) Value / Expression Value to the Left Hand Side (LHS) Variable "

=>The Symbol for Assigment Operator is single equal to ( = ).

=>In Python Programming,we can use Assigment Operator in two ways.

1. Single Line Assigment

2. Multi Line Assigment

**1. Single Line Assigment:**

----------------------------------------

=>Syntax: LHS Varname= RHS Value

LHS Varname= RHS Expression

=>With Single Line Assigment at a time we can assign one RHS Value / Expression to the single LHS Variable Name.

------------------

**Examples:**

-----------------

>>> a=10

>>> b=20

>>> c=a+b

>>> print(a,b,c)------------10 20 30

-----------------------------------------------------------------------------------

**2. Multi Line Assigment:**

----------------------------------------

=>Syntax: Var1,Var2.....Var-n= Val1,Val2....Val-n

Var1,Var2.....Var-n= Expr1,Expr2...Expr-n

Here The values of Val1, Val2...Val-n are assigned to Var1,Var2...Var-n Respectively.

Here The values of Expr1, Expr2...Expr-n are assigned to Var1,Var2...Var-n Respectively.

Examples:

-------------------

>>> a,b=10,20

>>> print(a,b)------------10 20

>>> c,d,e=a+b,a-b,a\*b

>>> print(c,d,e)-------------30 -10 200

>>> sno,sname,marks=10,"Rossum",34.56

>>> print(sno,sname,marks)---------10 Rossum 34.56

------------------------------------------

>>> a,b=10,20

>>> print(a,b)-------------------10 20

>>> a,b=b,a # Swapping Logic

>>> print(a,b)-----------20 10

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

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

**3. Relational Operators( Comparision Operators )**

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

=>The purpose of Relational Operators is that "To compare Two Values".

=>**If Two Values Connects with Relational Operators then It is called Relational Expression.**

=>The Result of Relational Expression is either True or False .

=>The Relational Expression is called Test Condition and whose result can be either True or False.

=>In Python Programming, we have 6 types of Relational Operators and They are given in the following Table:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**SLNO SYMBOL MEANING EXAMPLES**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*1. > greater than print(10>2)------True

print(10>20)-----False

2. < less than print(10<20)-----True

print(10<3)-------False

3. == Equality print(10==10)----True

(Double Equal to) print(10==20)-----False

4. != Not Equal to print(10!=20)------True

print(10!=10)------False

5. >= greater than print(10>=2)-------True

or equal to print(10>=20)-----False

6. <= Less than print(10<=2)-------False

or equal to print(10<=20)-----True

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**#Program for Demonstarting Relational Operators**

**#RelationalOpEx1.py**

a=int(input("Enter Value of a:"))

b=int(input("Enter Value of b:"))

print("="\*50)

print("\t\tResult of Relational Operators")

print("="\*50)

print("\t\t\t{} > {}={}".format(a,b,a>b))

print("\t\t\t{} < {}={}".format(a,b,a<b))

print("\t\t\t{} == {}={}".format(a,b,a==b))

print("\t\t\t{} != {}={}".format(a,b,a!=b))

print("\t\t\t{} >= {}={}".format(a,b,a>=b))

print("\t\t\t{} <= {}={}".format(a,b,a<=b))

print("="\*50)

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

**4. Logical Operators ( Comparision Operators )**

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

=>**The purpose of Logical Operators is that "To Combine Two or More Relational Expressions".**

**=>If Two Or More Relational Expressions Connects with Logical Operators then It is called Logical Expression.**

=>The result of Logical Expression is either True or False

=>The Logical Expression is called Compound Test Condition and whose result can be either True or False.

=>In Python Programming, we have 3 Types of Logical Operators. They are given in the following table.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**SLNO SYMBOL MEANING**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

1. and Physical ANDing

2. or Physical ORing

3. not --------------------------

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*==================================================================================

**1. and Operator**

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

=>Syntax: RelExpr1 and RelExpr2

=>The functionality of "and" operator is expressed with the following Truth tables

-----------------------------------------------------------------------------

RelExpr1 RelExpr2 RelExpr1 and RelExpr2

-----------------------------------------------------------------------------

True True True

False False False

True False False

False True False

-----------------------------------------------------------------------------

**Example-1**

--------------------

>>> True and True---------True

>>> False and False-------False

>>> True and False--------False

>>> False and True--------False

----------------------------

**Examples-2**

----------------------------

>>> (10<2) and (20>30)---------False--Short circuit Evalualtion

>>> (10>20) and (20>3)--------False---Short circuit Evalualtion

>>> (10>20) and (10>40) and (10>2)---False---Short circuit Evalualtion

>>>(10>2) and (10>20) and (20>2) and (30>15)---False---Short circuit Evalualtion

>>> (10>2) and (100>20) and (200>10)---True---Full length Evalualtion

----------------------------------------------------------------------------

**Short Circuit Evalualtion in the case "and" Operator:**

----------------------------------------------------------------------------

=>if Two or More Relational Expressions Connected with "and" operator (Called Logical Expression) and if Initial Relational Expression is evaluated as False, then PVM will not evaulate rest of the Relational Expressions and Result of entire Logical Expression is False. This process of Evaulation is called Short Circuit Evaluation.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*==================================================================================

**2. or Operator**

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

=>Syntax: RelExpr1 or RelExpr2

=>The functionality of "or" operator is expressed with the following Truth tables

-----------------------------------------------------------------------------

**RelExpr1 RelExpr2 RelExpr1 or RelExpr2**

-----------------------------------------------------------------------------

True True True

False False False

True False True

False True True

-----------------------------------------------------------------------------

**Examples-1**

--------------------

>>> True or True----------True

>>> False or True---------True

>>> True or False---------True

>>> False or False--------False

-------------------------

**Examples-2**

-------------------------

>>> (10>20) or (10>2)--------True-------Short circuit Evalualtion

>>> (10>2) or (10>20)--------True-------Short circuit Evalualtion

>>> (10>2) or (10>20) or (10>30)----True------Short circuit Evalualtion

>>> (10>20) or (10>2) or (10>30) or (10>40)---True----Short circuit Evalualtion

>>> (10>20) or (10>200) or (10>30) or (10>40)----False--Full length Evalualtion

----------------------------------------------------------------------------

**Short Circuit Evalualtion in the case "or" Operator:**

----------------------------------------------------------------------------

=>if Two or More Relational Expressions Connected with "or" operator (Called Logical Expression) and if Initial Relational Expression is evaluated as True, then PVM will not evaulate rest of the Relational Expressions and Result of entire Logical Expression is True. This process of Evaulation is called Short Circuit Evaluation.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**3. not Operator**

==================================================================================**Syntax: not Relational Expression**

=>The functionality of "not" operator is expressed with the following Truth tables

-----------------------------------------------------------------------------

**RelExpr1 not RelExpr1**

-----------------------------------------------------------------------------

True False

False True

-----------------------------------------------------------------------------

Examples

--------------------

>>> not True-----------False

>>> not False---------True

--------------------------

Examples

---------------------------

>>> 10>20------------False

>>> not (10>20)----True

>>> (100!=100)------False

>>> not(100!=100)---True

--------------------------

Examples

---------------------------

>>> not 100-----False

>>> not -100----False

>>> not 0-------True

-----------------------------------------

>>> not "HYD"-------------False

>>> not "False"-----------False

>>> not 2-2----------------True

>>> not ""-----------------True

>>> not "int(2-2)"--------False

>>> not "not"------------False

>>> not not "not"-------True

>>> **not not---------------SyntaxError: invalid syntax**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

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

**1. Bitwise Left Shift Operator ( << )**

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

Syntax: varname = Given Number << No. of Bits

--------------------

Explanation:

--------------------

The Execution Process of Bitwise LeftShift Operator ( << ) is that "It Moves Number of Bits Towards Left Side By,

Adding Number of Zeros (Number of Zeros=Depending No. Of bits we Flipped-off) at Right Side.

-------------------------------

Examples

-------------------------------

>>> a=10

>>> b=a<<3

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

>>> print(4<<3)-------32

>>> print(9<<2)-------36

>>> print(10<<0)-----10

>>> print(10.3<<2)-----------TypeError: unsupported operand type(s) for <<: 'float' and 'int'

========================================x=========================================

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

**2. Bitwise RightShift Operator ( >> )**

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

Syntax: varname=Given Data >> No. of Bits

----------------

Explanation:

------------------

The Execution Process of Bitwise Right Shift Operator ( >> ) is that "It Moves Number of Bits Towards Right Side By,

Adding Number of Zeros (Number of Zeros=Depending No. Of bits we Flipped-off) at Left Side.

-------------------------------

Examples

-------------------------------

>>> a=10

>>> b=a>>3

>>> print(b)---------1

>>> print(16>>2)---4

>>> print(32>>3)---4

>>> print(32>>2)---8

>>> print(32>>0)---32

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

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

**3. Bitwise OR Operator ( | )**

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

=>Syntax: varname = Value1 | Value2

=>The Functionality of Bitwise OR Operator ( | ) is expressed with the following Truth Table.

---------------------------------------------------------------------------------------

**Value1 Value2 Value1 | Value2**

---------------------------------------------------------------------------------------

0 0 0

1 1 1

1 0 1

0 1 1

---------------------------------------------------------------------------------------

-----------------------------

**Examples-1**

-----------------------------

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

>>> 1|1----------------1

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

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

-----------------------------

**Examples-2**

-----------------------------

>>>a=10--------------------------- 1010

>>>b=5----------------------------- 0101

---------

c=a|b -------------------------------1111

>>>print(c)-------15

-----------------------------

>>> print(10|10)------10

------------------------------------

Note:

>>>"Apple" or "Mango"------------'Apple'

>>>"Apple" | "Mango"--------------TypeError: unsupported operand type(s) for |: 'str' and 'str'

------------------------------------------------------------------

**Special Point--Most Imp**

------------------------------------------------------------------

>>>s1={10,20,30}

>>>s2={20,40,50}

>>>s3=s1.union(s2)

>>>print(s3,type(s3))

----------OR----------------------

>>> s1={10,20,30}

>>> s2={20,40,50}

>>> s3=s1.union(s2)

>>> print(s3,type(s3))------{50, 20, 40, 10, 30} <class 'set'>

>>> s1={10,20,30}

>>> s2={20,40,50}

>>> s3=s1|s2 # Bitwise OR Operator (|)

>>> print(s3,type(s3))--------{50, 20, 40, 10, 30} <class 'set'>

---------------------------------------------------------

>>> s1={1.2,2.3,4.5}

>>> s2={7.2,2.3,8.5}

>>> s3=s1|s2 # Bitwise OR (|) Operator

>>> print(s3,type(s3))----------{1.2, 2.3, 4.5, 7.2, 8.5} <class 'set'>

>>> **print(1.2|2.3)----------------TypeError: unsupported operand type(s) for |: 'float' and 'float'**

---------------------------------------------------------

>>> s1={"Apple","Mango","Kiwi"}

>>> s2={"Sberry","Mango","Guava"}

>>> s3=s1|s2 # Bitwise OR (|) Operator

>>> print(s3,type(s3))-----{'Guava', 'Sberry', 'Kiwi', 'Apple', 'Mango'} <class 'set'>

---------------------------------------------------------

>>>"Apple" | "Mango"---------TypeError: unsupported operand type(s) for |: 'str' and 'str'

=======================================x==========================================

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

**4. Bitwise AND Operator ( & )**

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

=>Syntax: varname = Value1 & Value2

=>The Functionality of Bitwise AND Operator ( & ) is expressed with the following Truth Table.

---------------------------------------------------------------------------------------

Value1 Value2 Value1& Value2

---------------------------------------------------------------------------------------

0 0 0

1 1 1

1 0 0

0 1 0

---------------------------------------------------------------------------------------

-------------------------------

**Examples**

-------------------------------

>>> 0&0----------------0

>>> 1&1---------------1

>>> 1&0---------------0

>>> 0&1---------------0

--------------------------------

**Examples**

--------------------------------

>>> a=10--------------------- 1010

>>> b=5---------------------- 0101

--------------------

>>> c=a&b 0000

--------------------

>>> print(c)------0

>>> 10 and 5-------5

>>> 5 and 10--------10

>>> print(7&3)-------3

------------------------------------------------------------------

**Special Points--Most Imp**

------------------------------------------------------------------

>>> s1={10,20,30}

>>> s2={40,20,50}

>>> s3=s1.intersection(s2)

>>> print(s3,type(s3))-------------{20} <class 'set'>

-------------------

>>> s1={10,20,30}

>>> s2={40,20,50}

>>> s3=s1&s2 # Bitwise AND Operator (& )

>>> print(s3,type(s3))---------{20} <class 'set'>

------------

>>> s1={10,20,30}

>>> s2={40,60,50}

>>> s3=s1&s2

>>> print(s3,type(s3))----set() <class 'set'>

**>>>"Apple"&"Mango"-------TypeError: unsupported operand type(s) for &: 'str' and 'str'**

>>>"Apple" and "Mango"------'Mango'

-------------------------------------------------------

>>> s1={1.2,2.3,4.5}

>>> s2={7.2,2.3,8.5}

>>> s3=s1&s2

>>> print(s3,type(s3))------{2.3} <class 'set'>

>>> 1.2 & 2.3--------------TypeError: unsupported operand type(s) for &: 'float' and 'float'

>>>"$" and "#"-----------'#'

>>>"$"&"#" ----------------TypeError: unsupported operand type(s) for &: 'str' and 'str'

=========================================x========================================

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

**5. Bitwise Complement Operator ( ~ )**

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

=>Syntax: varname= ~Given Number

=>The execution process of Bitwise Complement Operator ( ~ ) is that " It Inverts the given bits".

=>Inverting the bits is nothing but 1 becomes 0 and 0 becomes 1

=>The formula for Bitwise Complement Operator ( ~ ) is given bellow

~Given Number = - ( Given Number + 1)

----------------------------------------------------------------------------------------------------------------------------------------------------------------

Example1:

--------------------

>>> a=10

>>> print(~a)-------------- -11

>>> a=100

>>> print(~a)----------------- -101

-----------------

>>> a=-123

>>> print(~a)----------------- 122

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

**Q) Prove that ~10 is -11**

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

Proof : Given ~10 = -11

Here -11 is the Opposite counter part of 11

=>Given Number 10 and Whose Binary Part is 1010

=> ~10 ----------------- ~(1010) = 0101 (Which is the Binary form of -11 which is the 2's complement of 11)

------------------------------------

=>Here -11 is the Opposite counter part of 11 (2's complement of 11 )

=>All Negative Number stored in Main Memory in the form 2's Complement (2's complement= 1's complement+1)

-------------------------------------------------------

=>Here we Take 11 and whose Binary form is 1 0 1 1

(1's Complement of any Number= 1 becomes 0 and 0 becomes 1)

=>1's Complement of 11 is----------------------------0 1 0 0

=>2's Complement of 11 is----------------------------1's Complement of 11 + 1

=> 0100

=>0001 Binary Addition (0+0=0, 1+0=1, 0+1=1 , 1+1= 0 with carry 1 )

--------------

0101----which is 2's Complement of 11 --result is -11

------------------

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

**Q) Prove that ~16 is -17**

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

Proof : Given ~16 = -17

Here -17 is the Opposite counter part of 17

=>Given Number 16 and Whose Binary Part is 0001 0000

=> ~16 ----------------- ~(0001 0000) = 1110 1111 (Which is the Binary form of -17)

---------------------------------------------------------

=>Here -17 is the Opposite counter part of 17

=>All Negative Number stored in Main Memory in the form 2's Complement (2's complement= 1's complement+1)

-------------------------------------------------------

=>Here we Take 17 and whose Binary form is 0001 0001

(1's Complement of any Number= 1 becomes 0 and 0 becomes 1)

=>1's Complement of 17 is----------------------------1110 1110

=>2's Complement of 17 is----------------------------1's Complement of 17 + 1

=> 1110 1110

0000 0001 Binary Addition ( (0+0=0, 1+0=1, 0+1=1 , 1+1= 0 with carry 1 )

------------------

1110 1111 (Which is the Binary form of -17)

=============================================x==========================================================

**Q) Prove that ~15 is -16**

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

Proof : Given ~15 = -16

Here -16 is the Opposite counter part of 16

=>Given Number 15 and Whose Binary Part is 0000 1111

=> ~15 ----------------- ~(0000 1111) = 1111 0000 (Which is the Binary form of -16)

---------------------------------------------------------

=>Here -16 is the Opposite counter part of 16

=>All Negative Number stored in Main Memory in the form 2's Complement (2's complement= 1's complement+1)

-------------------------------------------------------

=>Here we Take 16 and whose Binary form is 0001 0000

(1's Complement of any Number= 1 becomes 0 and 0 becomes 1)

=>1's Complement of 16 is----------------------------1110 1111

=>2's Complement of 16 is----------------------------1's Complement of 16 + 1

=> 1110 1111

0000 0001 Binary Addition ( (0+0=0, 1+0=1, 0+1=1 , 1+1= 0 with carry 1 )

------------------

1111 0000 (Which is the Binary form of -16)

=============================================x====================================

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

**Bitwise XOR Operator ( ^ )**

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

=>Syntax: Varname= Value1 ^ Value2

=>The Functionality of Bitwise XOR Operator ( ^ ) is expressed with the following Truth Table

-----------------------------------------------------------------------------------------------

Value1 Value2 Value1^Value2

-----------------------------------------------------------------------------------------------

1 0 1

0 1 1

1 1 0

0 0 0

-----------------------------------------------------------------------------------------------

**Examples1**

----------------------

>>> 1^0----------------1

>>> 0^1----------------1

>>> 1^1----------------0

>>> 0^0----------------0

---------------------------------

**Examples2**

---------------------------------

>>> print(2^3)--------1

>>> print(10^15)-----5

---------------------------------

**Special Points**

---------------------------------

>>> s1={10,20,30}

>>> s2={15,20,25}

>>> s3=s1.symmetric\_difference(s2)

>>> print(s3,type(s3))----------------{10, 15, 25, 30} <class 'set'>

--------------

>>> s1={10,20,30}

>>> s2={15,20,25}

>>> s3=s1^s2

>>> print(s3,type(s3))------------{10, 15, 25, 30} <class 'set'>

>>> s1={"apple","mango","kiwi"}

>>> s2={"Sberry","mango","guava"}

>>> s3=s1^s2

>>> print(s3,type(s3))--------{'guava', 'apple', 'kiwi', 'Sberry'} <class 'set'>

>>> {1.2,2.3,3.4}^{1.2,2.3,4.5}--------{3.4, 4.5}

----------------------

>>> 1.2^2.3-----------------------TypeError: unsupported operand type(s) for ^: 'float' and 'float'

>>>"apple"^1.2----------------TypeError: unsupported operand type(s) for ^: 'str' and 'float'

---------------------------------------------------

**Imp Logic----Swapping of Two Integer values**

---------------------------------------------------

>>> a=3

>>> b=4

>>> print(a,b)---------3 4

>>> a=a^b

>>> b=a^b

>>> a=a^b

>>> print(a,b)--------4 3

================================================x=================================

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

**Special Points about Logical Operators**

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

**on "and" Operator**

---------------------------------------------------------------

>>> 100 and 200------------------200

>>> 100 and 0---------------------0

>>> -123 and -124----------------124

>>> 0 and 123---------------------0

>>> 100 and 200 and 300-----300

>>> 100 and 0 and 300--------0

>>>"True" and "False"-------'False'

>>>"True" and False---------False

>>> 0 and False-----------------0

>>>"0" and False and 0----False

----------------------------------------------

**on "or" Operator**

----------------------------------------------

>>> 100 or 200-------------------100

>>> 0 or 200----------------------200

>>> 0 or -120----------------------120

>>> True or 123-----------------True

>>> 100 or 0 or 200------------100

>>>"Python" or "Django"----'Python'

>>>"Python" or "Django" or False----'Python'

>>> False or "True" or False------------'True'

-----------------------------------------------

**on "and ""or" Operator**

-----------------------------------------------

>>> 100 or 200 and 300-------100

>>> False or True and False or True and False----False

>>> 0 or 200 and 0 and 200 or 100-----100

>>>"Python" or "Java" or "HTML" and "Java" and "C" or "DS"----'Python'

>>> not "Python" or "Java" or "HTML" and "Java" and "C" or "DS"----'Java'

===============================================x==================================

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

**6. Membership Operators--Most Imp**

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

=>The purpose of Membership Operators is that "To Check the Whether the Specified Values Present in Iterable object or Not".

=>An Iterable object is one, which contains More than One Value (Examples: Sequence, List,set,dict )

=>A Non-Iterable object is one, which contains One Value only (Examples: int, float,bool,complex, NoneType)

=>In Python Programming, we have Two Types of Membership Operators. They are

1. in

2. not in

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***1. in**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*=>Syntax: Value in Iterable-Object

=>Here "in" operator Returns True Provided "Value" Present in Iterable-Object.

=>Here "in" operator Returns False Provided "Value" not Present in Iterable-Object.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

2. not in

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

=>Syntax: Value not in Iterable-Object

=>Here "not in" operator Returns True Provided "Value" not Present in Iterable-Object.

=>Here "not in" operator Returns False Provided "Value" Present in Iterable-Object.

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

**Examples**

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

>>> s="PYTHON"

>>> print(s)--------------------PYTHON

>>>"P" in s-------------------True

>>>"P" not in s-------------False

>>>"O" not in s------------False

>>>"o" in s-----------------False

>>>"N" not in s------------False

-----------------------------------------------

>>> s="PYTHON"

>>> print(s)

PYTHON

>>>"PYTH" in s

True

>>>"HON" in s

True

>>>"PYO" in s

False

>>>"NOH" not in s

True

>>>"NOH" in s

False

>>> s in s[::-1]

False

>>> s in s[::]

True

>>> s[::2] not in s[::2][::-1]

True

>>> s[::2] not in s[::2][::-1][::-1]

False

>>> s[::2] in s[::2][::-1][::-1]

True

---------------------------------------------------

>>> lst=[10,"Rossum",23.45,2+3j]

>>> print(lst)

[10, 'Rossum', 23.45, (2+3j)]

>>> 10 in lst

True

>>>"ssum" in lst

False

>>>"ssum" in lst[1]

True

>>> **2+3j in lst[-1]-------------TypeError: argument of type 'complex' is not iterable**

--------------------------------------------------------------

>>> lst=[10,"Rossum",23.45,2+3j]

>>> print(lst)------------[10, 'Rossum', 23.45, (2+3j)]

>>> 2.0 in lst[-1].real--------------TypeError: argument of type 'float' is not iterable

----------------------------------------------

>>> lst=[10,"Rossum",23.45,2+3j]

>>> print(lst)----------------------[10, 'Rossum', 23.45, (2+3j)]

>>> lst[1][::-1] in lst[1:2][::-1]--------False

>>> print(lst[1:2])-----------['Rossum']

>>> print(lst[1:2][::-1])-----['Rossum']

>>> lst[1][::-1] in lst[1][::-1]----True

>>> type(lst[1:2])----------------<class 'list'>

>>> type(lst[1])-------------------<class 'str'>

-------------------------------------------------------------------

>>> d1={10:"Apple",20:"Mango",30:"Kiwi"}

>>> print(d1)----------------{10: 'Apple', 20: 'Mango', 30: 'Kiwi'}

>>>"Apple" in d1

False

>>>"Kiwi" not in d1

True

>>> 10 in d1

True

>>> 100 in d1

False

>>> **(10:"Apple") in d1-----------------SyntaxError: invalid syntax**

>>> (10:"Apple") in d1.items()-----------SyntaxError: invalid syntax

>>> d1.items()-----------------dict\_items([(10, 'Apple'), (20, 'Mango'), (30, 'Kiwi')])

>>> (10,"Apple") in d1.items()

True

>>> (10,"Python") in d1.items()

False

>>>"Apple" in d1[10]

True

>>>"Apple"[::-2] not in d1[10][::-1]

True

>>> **d1 in d1------------TypeError: unhashable type: 'dict'**

====================================x==============================================

>>>"LIRIL"[0:len("LIRIL")][::2] in "LIRIL"[::-1][::2]

True

>>> str(complex("2+3j").real) not in str("2+3j")[0]

True

>>> str(complex("2+3j").real)[0] not in str("2+3j")[0]

False

=================================================x================================

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

**Ternary Operator in Python ( if...else Operator )**

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

=>The ternary operator in Python is "if..else " Operator

=>Syntax: varname= Expr1 if Test Cond else Expr2

-----------------------

**Explanation:**

-----------------------

=>Here Test Cond can be either **Relational Expression OR Logical Expression** and whose result can be either True OR False

=>If the Test Cond is True then PVM executes Expr1 and whose result Placed in LHS Varname.

=>If the Test Cond is False then PVM executes Expr2 and whose result Placed in LHS Varname.

==========================================x=======================================

**#program for Palindrome**

**#TernaryOpEx1.py**

value=input("Enter any Value:")

res= "Palindrome" if value==value[::-1] else "Not Palindorme"

print("'{}' is {}".format(value,res))

**#program for finding Biggest of Two Numbers and Check for Equality**

**#TernaryOpEx3.py**

a=float(input("Enter Value of a:"))#a=10

b=float(input("Enter Value of b:"))# b=10

res=a if a>b else b if b>a else "Both Values are Equal"

print("Big({},{})={}".format(a,b,res))

**#program for chekcing the number is +Ve or -Ve or Zero**

**#TernaryOpEx4.py**

n=float(input("Enter Any Number:")) # n=-10

res="+VE" if n>0 else "-VE" if n<0 else "ZERO"

print("{} is {}".format(n,res))

**#program for chekcing biggest of Three Numbers and Check for Equality**

**#TernaryOpEx5.py**

a=int(input("Enter Value of a:")) # a=10

b=int(input("Enter Value of b:")) # b=100

c=int(input("Enter Value of c:"))# c=100

#Code for Finding a big----- a>b and a>c

#Code for finding b big--------b>a and b>c

#Code for Finding c big----- c>a and c>b

bigv=a if (a>=b) and (a>c) else b if b>a and b>=c else c if c>a and c>=b else " All are Equal"

print("Big({},{},{})={}".format(a,b,c,bigv))

**#program for chekcing biggest of Three Numbers and Check for Equality**

**#TernaryOpEx6.py**

a=int(input("Enter Value of a:")) # a=10

b=int(input("Enter Value of b:")) # b=100

c=int(input("Enter Value of c:"))# c=100

#Code for Finding a big----- a>b and a>c

#Code for finding b big--------b>a and b>c

#Code for Finding c big----- c>a and c>b

bigv=a if b<=a>c else b if a<b>=c else c if a<=c>b else " All are Equal"

print("Big({},{},{})={}".format(a,b,c,bigv))

**#TernaryOpEx7.py**

st=int(input("Enter Ur Talent in terms of Digit(0-9):"))

scap="Bellow Average" if st in range(4) else "Average" if st in range(4,7) else "Above Average" if st in range(7,9) else "Good Student" if st==9 else "Extra Candidate"

print("{} range is {}".format(st,scap))

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

**Identity Operators----Python Command Prompt**

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

=>The purpose of Identity Operators is that "To Compare the memory address of Two Objects".

=>In Python Programming, we have Two Types of Identity Operators. They are

1. is

2. is not

------------------------------------------------------------------------------------------------------------------------------------------

**1. is**

------------------------------------------------------------------------------------------------------------------------------------------

Syntax: Object1 is Object2

=>Here "is" Operator Returns True provided Memory Address of Object1 and Object2 are Same

=>Here "is" Operator Returns False provided Memory Address of Object1 and Object2 are Different

------------------------------------------------------------------------------------------------------------------------------------------

**2. is not**

------------------------------------------------------------------------------------------------------------------------------------------

Syntax: Object1 is not Object2

=>Here "is not " Operator Returns True provided Memory Address of Object1 and Object2 are Different

=>Here "is not " Operator Returns False provided Memory Address of Object1 and Object2 are Same

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

**Examples**

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

NOTE1: "is" Operator Returns True Provided Object1 and Object2 are in Deep Copy

NOTE2: "is" Operator Returns False Provided Object1 and Object2 are in Shallow Copy

NOTE3: "is not " Operator Returns True Provided Object1 and Object2 are in Shallow Copy

NOTE4: "is not " Operator Returns False Provided Object1 and Object2 are in Deep Copy

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

**Examples**

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

**>>> a=None**

**>>> b=None**

**>>> print(a,id(a))--------------------None 140733786138112**

**>>> print(b,id(b))--------------------None 140733786138112**

**>>> a is b--------------------True**

**>>> a is not b--------------------False**

-----------------------------------------------------

>>> d1={10:"Apple",20:"Mango"}

>>> d2={10:"Apple",20:"Mango"}

>>> print(d1,id(d1))--------------------{10: 'Apple', 20: 'Mango'} 2204488840320

>>> print(d2,id(d2))--------------------{10: 'Apple', 20: 'Mango'} 2204488840000

>>> d1 is d2--------------------False

>>> d1 is not d2--------------------True

-------------------------------------------------------------

>>> s1={10,20,30,40}

>>> s2={10,20,30,40}

>>> print(s1,id(s1))--------------------{40, 10, 20, 30} 2204488776416

>>> print(s2,id(s2))--------------------{40, 10, 20, 30} 2204488776640

>>> s1 is s2--------------------False

>>> s1 is not s2--------------------True

>>> fs1=frozenset(s1)

>>> fs2=frozenset(s2)

>>> print(fs1,id(fs1))--------------------frozenset({40, 10, 20, 30}) 2204488774176

>>> print(fs2,id(fs2))--------------------frozenset({40, 10, 20, 30}) 2204489480928

>>> fs1 is fs2--------------------False

>>> fs1 is not fs2--------------------True

----------------------------------------------------------

>>> l1=[10,20,30,40,10]

>>> l2=[10,20,30,40,10]

>>> print(l1,id(l1))--------------------[10, 20, 30, 40, 10] 2204491272896

>>> print(l2,id(l2))--------------------[10, 20, 30, 40, 10] 2204491269312

>>> l1 is l2--------------------False

>>> l1 is not l2--------------------True

>>>

>>> t1=(10,20,30,40,10)

>>> t2=(10,20,30,40,10)

>>> print(t1,id(t1))--------------------(10, 20, 30, 40, 10) 2204488792032

>>> print(t2,id(t2))--------------------(10, 20, 30, 40, 10) 2204488942752

>>> t1 is t2--------------------False

>>> t1 is not t2--------------------True

-----------------------------------------------------------

>>> r1=range(10,20,2)

>>> r2=range(10,20,2)

>>> print(r1,id(r1))--------------------range(10, 20, 2) 2204488737984

>>> print(r2,id(r2))--------------------range(10, 20, 2) 2204491246224

>>> r1 is r2--------------------False

>>> r1 is not r2--------------------True

>>> ba1=bytearray([10,20,30,40])

>>> ba2=bytearray([10,20,30,40])

>>> print(ba1,id(ba1))--------------------bytearray(b'\n\x14\x1e(') 2204488909936

>>> print(ba2,id(ba2))--------------------bytearray(b'\n\x14\x1e(') 2204488842800

>>> ba1 is ba2--------------------False

>>> ba1 is not ba2--------------------True

>>> b=bytes([10,20,30])

>>> c=bytes([10,20,30])

>>> print(b,id(b))--------------------b'\n\x14\x1e' 2204491245936

>>> print(c,id(c))--------------------b'\n\x14\x1e' 2204491245744

>>> b is c--------------------False

>>> b is not c--------------------True

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**MOST IMP str Data**

**>>> s1="PYTHON"**

**>>> s2="PYTHON"**

>>> print(s1,id(s1))--------------------PYTHON 2204491246128

>>> print(s2,id(s2))--------------------PYTHON 2204491246128

>>> s1 is s2--------------------True

>>> s1 is not s2--------------------False

----------------

>>> s1="PYTHON"

>>> s2="PYTHOn"

>>> print(s1,id(s1))--------------------PYTHON 2204491246128

>>> print(s2,id(s2))--------------------PYTHOn 2204491245888

>>> s1 is s2--------------------False

>>> s1 is not s2--------------------True

----------------

>>> s1="INDIA"

>>> s2="INDAI"

>>> s1 is s2--------------------False

>>> s1 is not s2--------------------True

----------------

>>> s1="JUST"

>>> s2="JSUT"

>>> s1 is s2--------------------False

>>> s1 is not s2--------------------True

>>> s1="THIS"

>>> s2="THSI"

>>> s1 is s2--------------------False

>>> s1 is not s2--------------------True

--------------------------

>>>**"INDIA" is "india"------- SyntaxWarning: "i**s" with 'str' literal. Did you mean "=="?

False

>>>"INDIA"=="india"-------False

>>>"INDIA"=="INDIA"-------True

-----------------------------------------------------------------------------

>>> a=2+2j

>>> b=2+2j

>>> print(a,id(a))--------------------(2+2j) 2204488858480

>>> print(b,id(b))--------------------(2+2j) 2204488843824

>>> a is b--------------------False

>>> a is not b--------------------True

--------------------------

>>> a=True

>>> b=True

>>> print(a,id(a))--------------------True 140733786138048

>>> print(b,id(b))--------------------True 140733786138048

>>> a is b--------------------True

>>> a is not b----------------False

>>> a=True

>>> b=False

>>> a is b--------------------False

>>> a is not b----------------True

-------------------------------------

>>> a=1.2

>>> b=1.2

>>> print(a,id(a))--------------------1.2 2204486381168

>>> print(b,id(b))--------------------1.2 2204488843984

>>> a is b--------------------False

>>> a is not b--------------------True

-----------------------------------------------------------------------------------------------------------------

**MOST MOST IMP----0 to 256 and -1 to -5 Int Data in multiple objects contains Same Address otherwise we have Different Address.**

-----------------------------------------------------------------------------------------------------------------

>>> a=300

>>> b=300

>>> print(a,id(a))--------------------300 2204488858480

>>> print(b,id(b))--------------------300 2204488858512

>>> a is b--------------------False

>>> a is not b--------------------True

>>> a=10

>>> b=10

>>> print(a,id(a))--------------------10 140733786933976

>>> print(b,id(b))--------------------10 140733786933976

>>> a is b--------------------True

>>> a is not b--------------------False

>>> a=256

>>> b=256

>>> print(a,id(a))--------------------256 140733786941848

>>> print(b,id(b))--------------------256 140733786941848

>>> a is b--------------------True

>>> a is not b--------------------False

>>> a=0

>>> b=0

>>> print(a,id(a))--------------------0 140733786933656

>>> print(b,id(b))--------------------0 140733786933656

>>> a is b--------------------True

>>> a is not b--------------------False

>>> a=257

>>> b=257

>>> print(a,id(a))--------------------257 2204488858480

>>> print(b,id(b))--------------------257 2204488843824

>>> a is b--------------------False

>>> a is not b--------------------True

>>> a=-1

>>> b=-1

>>> print(a,id(a))---------------------1 140733786933624

>>> print(b,id(b))---------------------1 140733786933624

>>> a is b--------------------True

>>> a is not b--------------------False

>>> a=-5

>>> b=-5

>>> print(a,id(a))---------------------5 140733786933496

>>> print(b,id(b))---------------------5 140733786933496

>>> a is b--------------------True

>>> a is not b--------------------False

>>> a=-6

>>> b=-6

>>> print(a,id(a))---------------------6 2204486381392

>>> print(b,id(b))---------------------6 2204488858480

>>> a is b--------------------False

>>> a is not b--------------------True

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

**Most Special Points--Multi Line Assigment**

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

>>> a,b=356,356

>>> print(a,id(a))--------------------356 2204488858480

>>> print(b,id(b))--------------------356 2204488858480

>>> a is b--------------------True

>>> a is not b--------------------False

>>>

>>> a,b=1.2,1.2

>>> print(a,id(a))--------------------1.2 2204486381168

>>> print(b,id(b))--------------------1.2 2204486381168

>>> a is b--------------------True

>>> a is not b--------------------False

>>> a,b=2+2.5j,2+2.5j

>>> print(a,id(a))--------------------(2+2.5j) 2204486381392

>>> print(b,id(b))--------------------(2+2.5j) 2204486381392

>>> a is b--------------------True

>>> a is not b--------------------False

--------------------------

>>> l1,l2=[10,20,30],[10,20,30]

>>> print(l1,id(l1))--------------------[10, 20, 30] 2204491271936

>>> print(l2,id(l2))--------------------[10, 20, 30] 2204486980032

>>> l1 is l2--------------------False

>>> l1 is not l2--------------------True

>>> d1={10:"Apple"}

>>> d2={10:"Apple"}

>>> d1,d2={10:"Apple"},{10:"Apple"}

>>> print(d1,id(d1))--------------------{10: 'Apple'} 2204488840320

>>> print(d2,id(d2))--------------------{10: 'Apple'} 2204491278848

>>> d1 is d2--------------------False

>>> d1 is not d2--------------------True

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

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

**String Handling Part-2**

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

=>We know that, on str data we can perform Both Indexing and Slicing Operations.

=>By using Indexing Concept, we can get Single Value from str object.

=>By using Slicing Concept, we can get Range of Values from str object.

=>Along with Indexing and Slicing Operations, we can perform Various Operations on str data by using Pre-defined Functions

present in str object.

--------------------------------------------------------------------------------------------------------------------------------

**Pre-defined Functions in str object**

--------------------------------------------------------------------------------------------------------------------------------

**1) capitalize()**

--------------------------------------------------------------------------------------------------------------------------------

=>This Function is used for capitalizing the first letter First word of a given Sentence only.

=>Syntax: strobj.capitalize()

(OR)

strobj=strobj.capitalize()

-----------------

**Examples:**

-----------------

>>> s="python"

>>> print(s,type(s))-------------------python <class 'str'>

>>> s.capitalize()--------------------'Python'

>>> s="python is an oop lang"

>>> print(s,type(s))-------------------------python is an oop lang <class 'str'>

>>> s.capitalize()-----------------------------'Python is an oop lang'

-------------------------------------

>>> s="python"

>>> print(s,type(s))--------------------python <class 'str'>

>>> s.capitalize()--------------------'Python'

>>> print(s,type(s))----------------python <class 'str'>

>>> s=s.capitalize()

>>> print(s,type(s))-----------------Python <class 'str'>

>>> s="python"

>>> s.capitalize()-------------------------'Python'

>>> s="python is an oop lang"

>>> s.capitalize()-------------------------'Python is an oop lang'

>>> s="python is an oop lang.python is also fun lang"

>>> s.capitalize()-------------------------'Python is an oop lang.python is also fun lang'

>>> s="PYTHON"

>>> s.capitalize()-------------------------'Python'

--------------------------------------------------------------------------------------------------------------------------------

**2) title():**

--------------------------------------------------------------------------------------------------------------------------------

=>This is used for obtaining Title Case of a Given Sentence (OR) Making all words First

Letters are capital.

Syntax: s.title()

(OR)

s=s.title()

------------------

**Examples:**

------------------

>>> s="python"

>>> print(s,type(s))-------------------python <class 'str'>

>>> s.capitalize()---------------------'Python'

>>> s.title()-----------------------------'Python'

----------------------------------------------------------

>>> s="python is an oop lang"

>>> print(s,type(s))------------------python is an oop lang <class 'str'>

>>> s.capitalize()--------------------'Python is an oop lang'

>>> s.title()----------------------------'Python Is An Oop Lang'

>>> print(s)----------------------------python is an oop lang

>>> s=s.title()

>>> print(s)--------------------------Python Is An Oop Lang

--------------------------------------------------------------------------------------------------------------------------------

**3) index()**

--------------------------------------------------------------------------------------------------------------------------------

=>This Function obtains Index of the specified Value

=>If the specified value does not exist then we get ValueError

=>Syntax: strobj.index(Value)

=>Syntax: indexvalue=strobj.index(value)

**Examples:**

-----------------

>>> s="python"

>>> s.index("p")------------------0

>>> s.index("y")------------------1

>>> s.index("o")------------------4

>>> s.index("n")------------------5

>>> s.index("K")----------------ValueError: substring not found

>>> s="python is an oop lang"

>>> s.index('is')------------------------7

>>> s.index('o')-------------------------4

>>> s.index('an')------------------------10

>>> s.index('10')------------------------ValueError: substring not found

>>> s.index("""an""")--------------------10

>>> s="python"

>>> s.index('thon')------------------------2

--------------------------------------------------------------------------------------------------------------------------------

**4) upper()**

--------------------------------------------------------------------------------------------------------------------------------

=>It is used for converting any type of Str Data into Upper Case.

=>Syntax:- strobj.upper()

OR

strobj=strobj.upper()

-----------------

**Examples:**

=---------------

>>> s="python"

>>> print(s)------------------------------python

>>> s.upper()-----------------------'PYTHON'

>>> s="python is an oop lang"

>>> print(s)---------------------------------python is an oop lang

>>> s.upper()--------------------------------'PYTHON IS AN OOP LANG'

>>> s="Python IS an OOP lang"

>>> print(s)-------------------------------Python IS an OOP lang

>>> s.upper()--------------------------'PYTHON IS AN OOP LANG'

>>> s="AbCdEf"

>>> print(s)------------------------AbCdEf

>>> s.upper()----------------------'ABCDEF'

>>> s="PYTHON"

>>> print(s)--------------------PYTHON

>>> s.upper()-----------------'PYTHON'

>>> s="123"

>>> print(s)------------------123

>>> s.upper()----------------'123'

--------------------------------------------------------------------------------------------------------------------------------

**5) lower()**

--------------------------------------------------------------------------------------------------------------------------------

=>It is used for converting any type of Str Data into lower Case.

=>Syntax:- strobj.lower()

OR

strobj=strobj.lower()

**Examples:**

-----------------

>>> s="Data Science"

>>> print(s)--------------Data Science

>>> s.lower()------------'data science'

>>> s="python"

>>> print(s)-------------python

>>> s.lower()-----------'python'

>>> s="PYTHON"

>>> print(s)-------------PYTHON

>>> s.lower()------------'python'

>>> s="PYThon"

>>> print(s)----------PYThon

>>> s.lower()---------'python'

--------------------------------------------------------------------------------------------------------------------------------

**6) isupper()**

--------------------------------------------------------------------------------------------------------------------------------

=>This Function returns True provided the given str object data is purely Upper Case otherwise it returns False.

Syntax: strobj.isupper()

**Examples:**

-----------------

>>> s="PYTHON"

>>> s.isupper()-----------True

>>> s="python"

>>> s.isupper()----------False

>>> s="Python"

>>> s.isupper()----------False

>>> s="PYThon"

>>> s.isupper()----------False

>>> s="123"

>>> s.isupper()------------False

>>> s="%$#^&@"

>>> s.isupper()-----------False

--------------------------------------------------------------------------------------------------------------------------------

**7)islower()**

--------------------------------------------------------------------------------------------------------------------------------

=>This Function returns True provided the given str object data is purely lower Case otherwise it returns False.

Syntax: strobj.islower()

-----------------

**Examples:**

-----------------

>>> s="pythopn"

>>> s.islower()------------True

>>> s="pythOn"

>>> s.islower()------------False

>>> s="PYTHON"

>>> s.islower()-----------False

>>> s="123"

>>> s.islower()----------False

--------------------------------------------------------------------------------------------------------------------------------

**8) isalpha()**

--------------------------------------------------------------------------------------------------------------------------------

=>This Function returns True provided str object contains Purely Alphabets otherwise returns False.

Syntax: strobj.isalpha()

-------------------

**Examples:**

-------------------

>>> s="Ambition"

>>> s.isalpha()--------------------True

>>> s="Ambition123"

>>> s.isalpha()-------------------False

>>> s="1234"

>>> s.isalpha()------------------False

>>> s=""

>>> s.isalpha()------------------False

>>> s="#$%^@"

>>> s.isalpha()-----------------False

>>> s="AaBbZz"

>>> s.isalpha()----------------True

>>> s="Apple"

>>> s.isalpha()--------------------True

>>> s="Apple123"

>>> s.isalpha()--------------------False

>>> s="Ap ple"

>>> s.isalpha()--------------------False

>>> s="123"

>>> s.isalpha()--------------------False

>>> s="Apple"

>>> s.isalpha()------------------------True

>>> s="Apple123"

>>> s.isalpha()------------------------False

>>> s="Ap ple"

>>> s.isalpha()------------------------False

>>> s="123"

>>> s.isalpha()------------------------False

>>> s="Pyth$on"

>>> s.isalpha()------------------------False

>>> s="PYTHON311"

--------------------------------------------------------------------------------------------------------------------------------

**9) isdigit()**

--------------------------------------------------------------------------------------------------------------------------------

=>This Function returns True provided given str object contains purely digits otherwise returns False

**Examples:**

--------------------

>>> s="python"

>>> s.isdigit()------------------False

>>> s="python123"

>>> s.isdigit()----------------False

>>> s="123"

>>> s.isdigit()-----------------True

>>> s="123 456"

>>> s.isdigit()---------------False

>>> s="1\_2\_3"

>>> s.isdigit()---------------False

>>> s="123KV"

>>> s.isdigit()-------------False

--------------------------------------------------------------------------------------------------------------------------------

**10) isalnum()**

--------------------------------------------------------------------------------------------------------------------------------

=>This Function returns True provided str object contains either Alpabets OR Numerics or Alpha-Numerics only otherwise It returns False.

=>Syntax: strobj. isalphanum()

---------------------------

=>**Examples:**

---------------------------

>>> s="python310"

>>> s.isalnum()-----------------True

>>> s="python"

>>> s.isalnum()-----------------True

>>> s="310"

>>> s.isalnum()-----------------True

>>> s="$python310"

>>> s.isalnum()-----------------False

>>> s="python 310"

>>> s.isalnum()----------------False

>>> s="$python3.10"

>>> s.isalnum()----------------False

>>> s="python3.10"

>>> s.isalnum()-------------False

>>> s.isalnum()------------------------True

>>> s="PYTHON"

>>> s.isalnum()------------------------True

>>> s="311"

>>> s.isalnum()------------------------True

>>> s="PYT HON"

>>> s.isalnum()------------------------False

>>> s="PYTHON3.11"

>>> s.isalnum()------------------------False

>>> s="PYTH$on"

>>> s.isalnum()------------------------False

>>> s="123.56"

>>> s.isalnum()------------------------False

>>> s="123"

>>> s="Pyth$on"

>>> s.isalnum()--------------------True

>>> s="PYTHON"

>>> s.isalnum()--------------------True

>>> s="311"

>>> s.isalnum()--------------------True

>>> s="PYT HON"

>>> s.isalnum()--------------------False

>>> s="PYTHON3.11"

>>> s.isalnum()--------------------False

>>> s="PYTH$on"

>>> s.isalnum()--------------------False

>>> s="123.56"

>>> s.isalnum()--------------------False

>>> s="123"

-------------------------------------------------------------------------------------------------------------------------------

**11) isspace()**

-------------------------------------------------------------------------------------------------------------------------------

=>This Function returns True provided str obj contains purely space otherwise it returns False.

=>Syntax: strobj.isspace()

------------------------

**Examples:**

----------------------

>>> s=""

>>> s.isspace()-----------True

>>> s=""

>>> s.isspace()--------------False

>>> s="python Prog"

>>> s.isspace()-------------False

>>> s="Prasana Laxmi"

>>> s.isspace()--------------False

>>> s.isalpha()-----------False

>>> s.isalpha() or s.isspace()-----------False

-------------------------------------------------------------------------------------------------------------------------------

**12) split()**

-------------------------------------------------------------------------------------------------------------------------------

=>This Function is used for splitting the given str object data into different words base specified delimter ( - \_ # % ^ ^ , ; ....etc)

=>The dafeult deleimter is space

=>The Function returns Splitting data in the form of list object

=>Syntax: strobj.split("Delimter")

(OR)

strobj.split()

(OR)

listobj= strobj.split("Delimter")

(OR)

listobj=strobj.split()

----------------

**Examples:**

----------------

>>> s="Python is an oop lang"

>>> print(s)----------------Python is an oop lang

>>> s.split()----------------['Python', 'is', 'an', 'oop', 'lang']

>>> len(s.split())-----------5

>>> x=s.split()

>>> print(x,type(x))---------['Python', 'is', 'an', 'oop', 'lang'] <class 'list'>

>>> len(x)---------------5

>>> s="12-09-2022"

>>> print(s)-------------12-09-2022

>>> s.split("-")----------['12', '09', '2022']

>>> s="12-09-2022"

>>> dob=s.split("-")

>>> print(dob,type(dob))------------['12', '09', '2022'] <class 'list'>

>>> print("Day",dob[0])----------Day 12

>>> print("Month ",dob[1])---------Month 09

>>> print("Year ",dob[2])----------Year 2022

---------------------------------------------------------

>>> s="Apple#Banana#kiwi/Guava"

>>> words=s.split("#")

>>> print(words)-----------['Apple', 'Banana', 'kiwi/Guava']

>>> words=s.split("/")

>>> print(words)------------------['Apple#Banana#kiwi', 'Guava']

>>> s="Apple is in red"

>>> s.split()--------------------['Apple', 'is', 'in', 'red']

>>> x=s.split()

>>> print(x,type(x))--------------------['Apple', 'is', 'in', 'red'] <class 'list'>

>>> len(x)--------------------4

>>> s="08-07-2023"

>>> print(s)--------------------08-07-2023

>>> x=s.split("-")

>>> print(x)--------------------['08', '07', '2023']

>>> print("Day=",x[0])--------------------Day= 08

>>> print("Month=",x[1])--------------------Month= 07

>>> print("Year=",x[2])--------------------Year= 2023

>>> s="Apple#Mango#kiwi-Banana"

>>> print(s)--------------------Apple#Mango#kiwi-Banana

>>> x=s.split("#")

>>> print(x)--------------------['Apple', 'Mango', 'kiwi-Banana']

>>> y=s.split("-")

>>> print(y)--------------------['Apple#Mango#kiwi', 'Banana']

>>> y[0]--------------------'Apple#Mango#kiwi'

>>>

>>> y[0].split("#")--------------------['Apple', 'Mango', 'kiwi']

>>> y[0:1]=y[0].split("#")

>>> print(y)--------------------['Apple', 'Mango', 'kiwi', 'Banana']

>>>

>>> s="123$456$678$156$"

>>> print(s)--------------------123$456$678$156$

>>> s.split("$")--------------------['123', '456', '678', '156', '']

>>> s="123$456$678$156"

>>> s.split("$")--------------------['123', '456', '678', '156']

>>>

-------------------------------------------------------------------------------

**13) join():**

-------------------------------------------------------------------------------

=>This Function is used for combining or joining list of values from any Iterable object

=>Syntax: strobj.join(Iterableobject)

Examples:

------------------------------

>>> lst=["HYD","BANG","AP","DELHI"]

>>> print(lst,type(lst))------------------['HYD', 'BANG', 'AP', 'DELHI'] <class 'list'>

>>> s=""

>>> s.join(lst)---------------'HYDBANGAPDELHI'

>>> s=""

>>> s.join(lst)------------------'HYD BANG AP DELHI'

-------------------------------------------------------------------

>>> t=("Rossum","is", "Father""of" ,"Python")

>>> print(t,type(t))

('Rossum', 'is', 'Fatherof', 'Python') <class 'tuple'>

>>> k=""

>>> k.join(t)

'Rossum is Fatherof Python'

>>> t=("Rossum","is", "Father", "of" ,"Python")

>>> k=""

>>> k.join(t)

'Rossum is Father of Python'

lst=["apple","mango","kiwi","guava"]

>>> print(lst,type(lst))------------------------['apple', 'mango', 'kiwi', 'guava'] <class 'list'>

>>> k=""

>>> k.join(lst)------------------------'applemangokiwiguava'

>>> print(k)

>>> k------------------------''

>>> lst=["apple","mango","kiwi","guava"]

>>> print(lst,type(lst))------------------------['apple', 'mango', 'kiwi', 'guava'] <class 'list'>

>>> k=""

>>> k=k.join(lst)

>>> print(k)------------------------applemangokiwiguava

>>> k------------------------'applemangokiwiguava'

>>>

>>> lst=["apple","mango","kiwi","guava"]

>>> print(lst,type(lst))------------------------['apple', 'mango', 'kiwi', 'guava'] <class 'list'>

>>> k=""

>>> k=k.join(lst)

>>> print(k)------------------------apple mango kiwi guava

>>> lst=["Python","is","an","oop","lang"]

>>> k=""

>>> k=k.join(lst)

>>> print(k)------------------------Python is an oop lang

>>> print(k,type(k))------------------------Python is an oop lang <class 'str'>

>>> k.split()------------------------['Python', 'is', 'an', 'oop', 'lang']

>>> s=""

>>> s.isnull()------------------------AttributeError: 'str' object has no attribute 'isnull'

---------------------------------------------------------------------------------------------------------------------

**14) startswith():**

---------------------------------------------------------------------------------------------------------------------

=>The startswith() Function returns True if the string starts with the specified value, otherwise False.

Examples:

-------------------

>>>s="Python is an oop lang"

>>>s.startswith("Python")------------True

>>>s.startswith("python")------------False

>>> s="Python is an oop lang"

>>> print(s)------------------------Python is an oop lang

>>> s.startswith("Python")------------------------True

>>> s.startswith("Pyt")------------------------True

>>> s.startswith("p")------------------------False

>>> s.startswith("p".upper())------------------------True

>>> s.startswith("lang")------------------------False

>>> s="Python is an oop lang"

>>> print(s)------------------------Python is an oop lang

---------------------------------------------------------------------------------------------------------------------

**15) endswith():**

---------------------------------------------------------------------------------------------------------------------

=>The endswith() Function returns True if the string ends with the specified value, otherwise False.

Examples:

-------------------

>>>s="Python is an oop lang"

>>>s.endswith("Python")------------False

>>>s.endswith("lang")------------True

>>> s.endswith("Python")------------------------False

>>> s.endswith("lang")------------------------True

>>> s.endswith("la")------------------------False

>>> s.endswith("ng")------------------------True

>>> s.endswith("n")------------------------False

>>> s.endswith("g")------------------------True

>>> s.endswith("lang".upper())------------------------False

>>> s.endswith("Python")--------------------False

>>> s.endswith("lang")--------------------True

>>> s.endswith("la")--------------------False

>>> s.endswith("ng")--------------------True

>>> s.endswith("n")--------------------False

>>> s.endswith("g")--------------------True

>>> s.endswith("lang".upper())--------------------False

---------------------------------------------------------------------------------------------------------------------

**16) swapcase()**

---------------------------------------------------------------------------------------------------------------------

=>Make the lower case letters upper case and the upper case letters lower case:

Examples:

>>>s="PyThOn"

>>>s.swapcase()--------pYtHoN

>>> s="PyThOn"

>>> s.swapcase()--------------------'pYtHoN'

>>> s="PYThon"

>>> s.swapcase()--------------------'pytHON'

>>> s="PYTHON"

>>> s.swapcase()--------------------'python'

>>> s="python"

>>> s.swapcase()--------------------'PYTHON'

>>> s="12345"

>>> s.swapcase()--------------------'12345'

>>> s="Python3.11"

>>> s.swapcase()--------------------'pYTHON3.11'

>>> s="$%^&\*()"

>>> s.swapcase()--------------------'$%^&\*()'

>>> s="PYTHON"

>>> s.lower()--------------------'python'

>>> s="PYTHON"

>>> s.swapcase()--------------------'python'

>>> s="PYThon"

>>> s.swapcase()--------------------'pytHON'

>>> s.lower()--------------------'python'

==========================================x=======================================

Flow Control Statements in Python

(OR)

Control Structures in Python-----8 Days

==========================================x=======================================

**Index**

------------

=>Purpose of Flow Control Statements in Python

=>Types of Flow Control Statements in Python

I. Conditional OR Selection OR Branching Statements

i) simple if statement

ii) if..else statement

iii) if..elif..else statement

iv) match case statement

=>Programming Exampoles

II. Looping or Iterating or Repetative Statements

i) while loop OR while ...else Loop

ii) for loop or for..else loop

=>Programming Examples

III. Transfer Flow Statements

i) break

ii) continue

iii) pass

iv) return

=>Programming Examples

=>Combined Programming Examples on Conditional , Looping and Transfer Flow Statements

=>Nested OR Inner Loops

i) while loop in while loop

ii) for loop in for loop

iii) while loop in for loop

iv) for loop in while loop

=>Programming Examples

============================================x===================================

==========================================x=======================================

**Introduction to Flow Control Statements in Python**

==========================================x=======================================

=>The purpose of Flow Control Statements in Python is that " To Perform Certain Operation(X-Operation-True OR Y-Operation-False) Only Once OR Perform Certain Operation Iteratively OR Repeatedly for Finite Number of Times until Test Condition Becomes False".

=>In Python Programming, we have **3** Types of Flow Control Statements. They are

**1. Conditional OR Selection OR Branching Statements**

**2. Looping OR Iterative OR Repetative Statements**

**3. Transfer Flow Statements**

==========================================x=======================================

==========================================x=======================================

**1. Conditional OR Selection OR Branching Statements**

==========================================x=======================================

=>The purpose of Conditional OR Selection OR Branching Statements is that "To perform X -Operation in the case of True OR Y-Operation in the case of False Only Once".

=>In Python Programming, we have 4 Types of Conditional OR Selection OR Branching Statements. They are

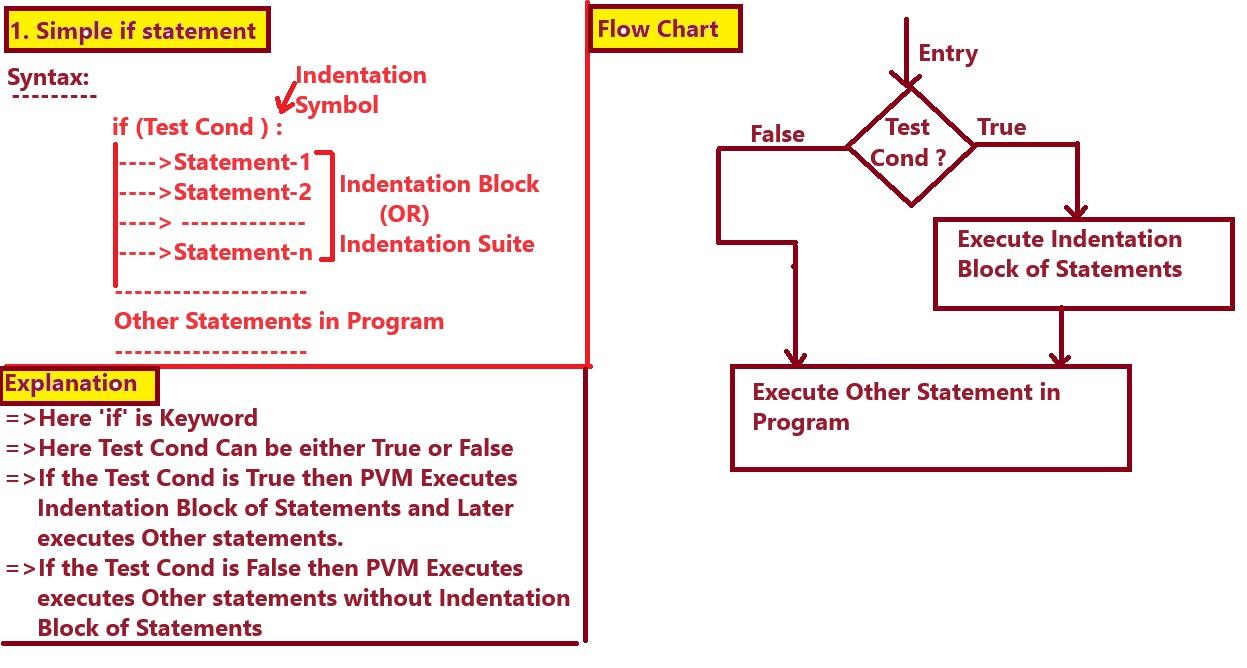
**1. Simple if statement**

**2. if..else statement**

**3. if..elif..else statement**

**4. match case statement.**

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



#program finding Biggest of Two Numbers

#SimplfIfStmtEx1.py

a=float(input("Enter Value of a:")) # a=20

b=float(input("Enter Value of b:")) # b=50

if(a>b):

print("Big(a={},b={})={}".format(a,b,a))

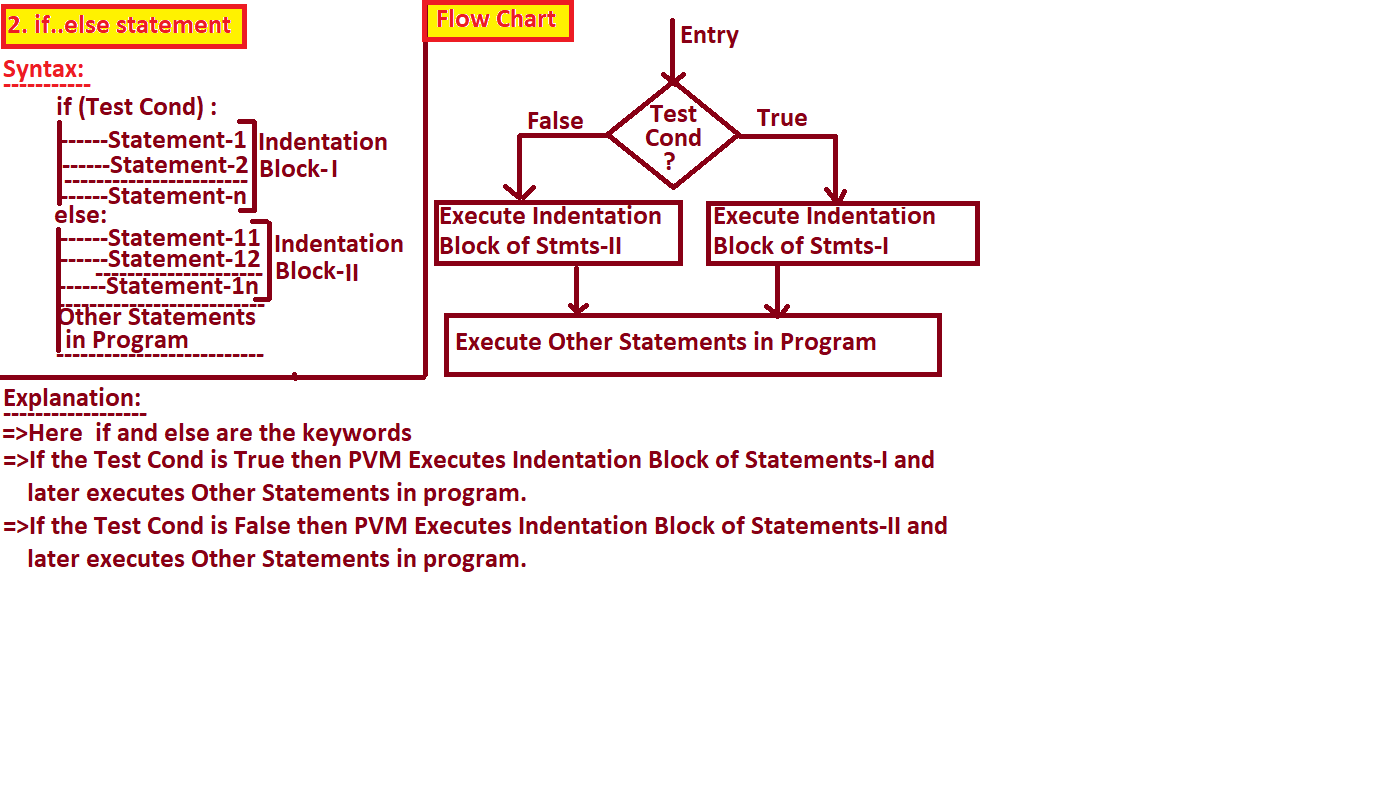
if(b>a):

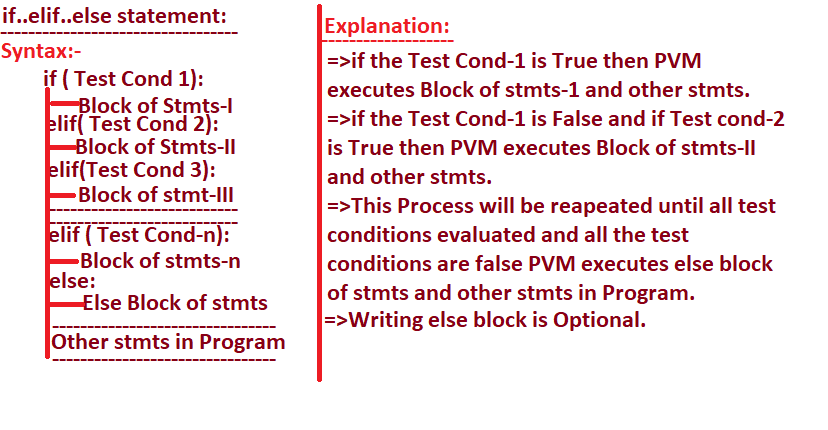
print("Big(a={},b={})={}".format(a,b,b))

if(a==b):

print("Both Values are Equal")

print("Program execution Completed")



****

**#Given number is odd or even**

n=int(input("Enter a Number:"))

if(n%2==0):

print("{} is Even".format(n))

else:

print("{} is Odd".format(n))

print("Program execution completed")

**#Given number is -ve or +ve or zero**

n=int(input("Enter a Number:")) # -100

if(n>0):

print("{} is +VE".format(n))

else:

if(n<0):

print("{} is -VE".format(n))

else:

print("{} is Zero".format(n))

print("i am inner if..else stmt")

print("i am in outer if..else stmt")

**#Given word is palindrome or not**

word=input("Enter a Word:")

if(word==word[::-1]):

print("{} is Palindrome".format(word))

else:

print("{} is not a palindrome".format(word))

**#Printing the given digit in alpha**

d=int(input("Enter a Digit:")) # d= 0 1 2 3 4 5 6 7 8 9

if(d==0):

print("{} is ZERO".format(d))

else:

if(d==1):

print("{} is ONE".format(d))

else:

if(d==2):

print("{} is TWO".format(d))

else:

if(d==3):

print("{} is THREE".format(d))

else:

if(d==4):

print("{} is FOUR".format(d))

else:

if(d==5):

print("{} is FIVE".format(d))

else:

if(d==6):

print("{} is SIX".format(d))

else:

if(d==7):

print("{} is SEVEN".format(d))

else:

if(d==8):

print("{} is EIGHT".format(d))

else:

if(d==9):

print("{} is NINE".format(d))

else:

if (d ==-1):

print("{} is -ONE".format(d))

else:

if (d == -2):

print("{} is -TWO".format(d))

else:

if (d == -3):

print("{} is -THREE".format(d))

else:

if (d == -4):

print("{} is -FOUR".format(d))

else:

if (d == -5):

print("{} is -FIVE".format(d))

else:

if (d == -6):

print("{} is -SIX".format(d))

else:

if (d == -7):

print("{} is -SEVEN".format(d))

else:

if (d == -8):

print("{} is -EIGHT".format(d))

else:

if (d == -9):

print("{} is -NINE".format(d))

else:

if(d<0):

print("{} is -VE Number".format(d))

else:

print("{} is +VE Number".format(d))

print("Program execution Completed")

**#IfElifElseStmtEx3.py**

d={0:"ZERO",1:"ONE",2:"TWO",3:"THREE",4:"FOUR",5:"FIVE",6:"SIX",7:"SEVEN",8:"EIGHT",9:"NINE",-1:"-ONE",-2:"-TWO",-3:"-THREE",-4:"-FOUR",-5:"-FIVE",-6:"-SIX",-7:"-SEVEN",-8:"-EIGHT",-9:"-NINE"}

dig=int(input("Enter a Digit:")) # d= 0 1 2 3 4 5 6 7 8 9

res= d.get(dig) if d.get(dig)!=None else "+Ve Number" if dig>0 else "-VE Number"

print("{} is {}".format(dig,res))

**#IfElifElseStmtEx2.py**

d=int(input("Enter a Digit:")) # d= 0 1 2 3 4 5 6 7 8 9

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==5):

print("{} is FIVE".format(d))

elif(d==6):

print("{} is SIX".format(d))

elif(d==7):

print("{} is SEVEN".format(d))

elif(d==8):

print("{} is EIGHT".format(d))

elif(d==9):

print("{} is NINE".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==-5):

print("{} is -FIVE".format(d))

elif(d==-6):

print("{} is -SIX".format(d))

elif(d==-7):

print("{} is -SEVEN".format(d))

elif(d==-8):

print("{} is -EIGHT".format(d))

elif(d==-9):

print("{} is -NINE".format(d))

elif(d<0):

print("{} is -VE Number".format(d))

elif(d>0):

print("{} is a +VE Number".format(d))

print("Program execution completed")

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

**match case statement (Python3.10 Version onwards--PEP)**

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

=>match case statement is one of conditional statement available from Python3.10 Version onwards.

=>The purpose of match case statement is that "To Deal with Pre-Designed Conditions OR Menu Driven Applications".

-----------------------------

**Syntax:**

-------------------------------

match(Choice Expr):

case Choice Label1:

Block of Stements-1

case Choice Label2:

Block of Stements-2

case Choice Label3:

Block of Stements-3

----------------------------

case Choice Label-n:

Block of Stements-n

case \_: # Default Case Block

default Block of Statements

-------------------------------------------------------

Other Statements in Program

-------------------------------------------------------

----------------------

**Explanation:**

----------------------

=>here "match" and "case" are the keywords

=>"Choice Expr" represents either int or str or bool

=>If "Choice Expr" is matching with "case label1" then PVM executes Block of Statements-1 and later executes Other statements in program.

=>If "Choice Expr" is matching with "case label2" then PVM executes Block of Statements-2 and later executes Other statements in program.

=>In General "Choice Expr" is trying to match with case label-1, case label-2,....case label-n then PVM executes corresponding block of statements and later executes Other statements in program.

=>If "Choice Expr" is not matching with any of the specified case labels then PVM executes Default Block of Statements which are written under default case block(case \_ ) and later executes Other statements in program.

=>Writing default case block is optional and If we write then it must be written at last (Otherwise we get SyntaxError)

=>When we represent multiple case labels under one case then those case labels must be combined with Bitwise OR Operator ( | ) .

------------------------------------------------------------------------------------------------------------------------------------------

**Examples**

---------------------

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Arithmetic Operations

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

1. Addition

2. Substraction

3. Multiplication

4. Division

5. Modulo Division

6. Exponentiation

7. Exit

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Enter Ur Choice:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

------------------------------------------------------------------------------------------------------------------------------------------

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Area of Different Figures

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

R. Rectangle

S. Square

C. Circle

T. Triangle

E. Exit

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Enter Ur Choice:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Home Work---student must do

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Temprature Conversion Calculator

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

1. F to C

2. F to K

3. C to F

4. C to K

5. K to F

6. K to C

7. Exit

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Enter Ur Choice: 1

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Fahrenheit to Celcius: C = (F-32) (5/9)---------------------case-1

Fahrenheit to Kelvin: K = (F-32) (5/9) + 273.15---------case-2

Celsius to Fahrenheit: F = C(9/5) + 32---------------------case-3

Celsius to Kelvin: K = C + 273.15----------------------------case-4

Kelvin to Celcius: C = K - 273.15----------------------------case-5

Kelvin to Fahrenheit: F = (K-273.15) (9/5) + 32---------case-6

**#Program for Implementing Arithmetic Operations by using Menu driven with match case statement**

**#MatchCaseEx1.py**

print("\*"\*50)

print("\tArithmetic Operations")

print("\*"\*50)

print("\t1.Addition")

print("\t2.Substraction")

print("\t3.Multiplication")

print("\t4.Division")

print("\t5.Modulo Division")

print("\t6.Exponentiation")

print("\t7.Exit")

print("\*"\*50)

ch=int(input("Enter Ur Choice:"))

match(ch):

case 1:

print("Enter Two Values for Addition Operation:")

a,b=int(input()),int(input())

print("\tsum({},{})={}".format(a,b,a+b))

case 2:

print("Enter Two Values for Substraction Operation:")

a,b=int(input()),int(input())

print("\tsub({},{})={}".format(a,b,a-b))

case 3:

print("Enter Two Values for Multiplication Operation:")

a,b=int(input()),int(input())

print("\tmul({},{})={}".format(a,b,a\*b))

case 4:

print("Enter Two Values for DivisionOperation:")

a,b=int(input()),int(input())

print("\tNormal Div({},{})={}".format(a,b,a/b))

print("\tFloor Div({},{})={}".format(a,b,a//b))

case 5:

print("Enter Two Values for Modulo Div Operation:")

k,v=int(input()),int(input())

print("\tmodl({},{})={}".format(k,v,k%v))

case 6:

a,b=int(input("Enter Base: ")),int(input("Enter Power:" ))

print("\tPow({},{})={}".format(a,b,a\*\*b))

case 7:

print("Thx for Using Arithmetic Cal:")

case \_:

print("Ur Selection of Operation is Wrong!!!--try again")

print("Program Execution Completed")

**#MatchCaseEx2.py**

menu="""\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Temprature Conversion Calculator

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

1. F to C

2. F to K

3. C to F

4. C to K

5. K to C

6. K to F

7. Exit

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"""

print(menu)

ch=int(input("Enter Ur Choice:"))

match(ch):

case 1:

F=float(input("Enter Temp in F to Convert into C:"))

C = (F-32)\*(5/9)

print("Temp in C:{}".format(round(C,2)))

case 2:

F=float(input("Enter Temp in F to Convert into K:"))

K = (F-32)\*(5/9) + 273.15

print("Temp in K:{}".format(K))

case 3:

C=float(input("Enter Temp in C to Convert into F:"))

F = C\*(9/5) + 32

print("Temp in F:{}".format(F))

case 4:

C=float(input("Enter Temp in C to Convert into K:"))

K= C + 273.15

print("Temp in K:{}".format(K))

case 5:

K=float(input("Enter Temp in K to Convert into C:"))

C = K - 273.15

print("Temp in C:{}".format(C))

case 6:

K=float(input("Enter Temp in K to Convert into F:"))

F = (K-273.15)\*(9/5) + 32

print("Temp in F:{}".format(F))

case 7:

print("Thx for using this program")

case \_:

print("Ur Selection of Operation Wrrong--try again")

**#MatchCaseEx2.py**

wkd=input("Enter a week name:")

match(wkd.upper()):

case "MONDAY":

print("{} is Working Day".format(wkd))

case "TUESDAY":

print("{} is Working Day".format(wkd))

case "WEDNESDAY":

print("{} is Working Day".format(wkd))

case "THURSDAY":

print("{} is Working Day".format(wkd))

case "FRIDAY":

print("{} is Working Day".format(wkd))

case "SATURDAY":

print("{} is Week End--Prepared UGAC Plans".format(wkd))

case "SUNDAY":

print("{} is Holy Day".format(wkd))

case \_:

print("{} is Not a Week Day".format(wkd))

**#MatchCaseEx4.py**

wkd=input("Enter a week name:")

match(wkd.upper()):

case "MONDAY"|"TUESDAY"|"WEDNESDAY"|"THURSDAY"|"FRIDAY":

print("{} is Working Day".format(wkd))

case "SATURDAY":

print("{} is Week End--Prepared UGAC Plans".format(wkd))

case "SUNDAY":

print("{} is Holy Day".format(wkd))

case \_:

print("{} is Not a Week Day".format(wkd))

**#MatchCaseEx5.py**

wkd=input("Enter a week name:")

match(wkd.upper()[0:3]):

case "MON"|"TUE"|"WED"|"THU"|"FRI":

print("{} is Working Day".format(wkd))

case "SAT":

print("{} is Week End--Prepared UGAC Plans".format(wkd))

case "SUN":

print("{} is Holy Day".format(wkd))

case \_:

print("{} is Not a Week Day".format(wkd))

**#MatchCaseEx6.py**

wkd=input("Enter a week name:")

if(wkd.upper() in ["MONDAY","TUESDAY","WEDNESDAY","THURSDAY","FRIDAY","SATURDAY","SUNDAY","MON","TUE","WED","THU","FRI","SAT","SUN"]):

match(wkd.upper()[0:3]):

case "MON"|"TUE"|"WED"|"THU"|"FRI":

print("{} is Working Day".format(wkd))

case "SAT":

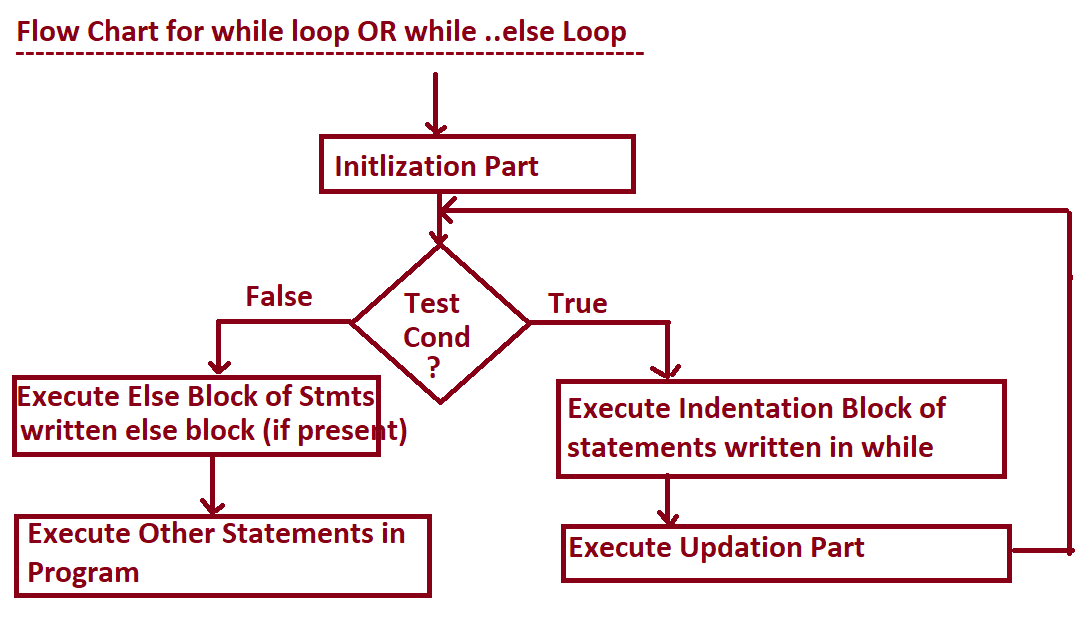
print("{} is Week End--Prepared UGAC Plans".format(wkd))

case "SUN":

print("{} is Holy Day".format(wkd))

else:

print("{} is Not a Week Day".format(wkd))



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

**Looping OR Iterative OR Repetative Statements**

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

=>The purpose of Looping OR Iterative OR Repetative Statements is that "To Perform Certain Operation Iteratively OR Repeatedly for Finite Number of Times until Test Condition Becomes False".

=>In Python Programming, we have 2 types of Looping OR Iterative OR Repetative Statements. They are

1. while loop OR while..else loop

2. for loop OR for..else loop

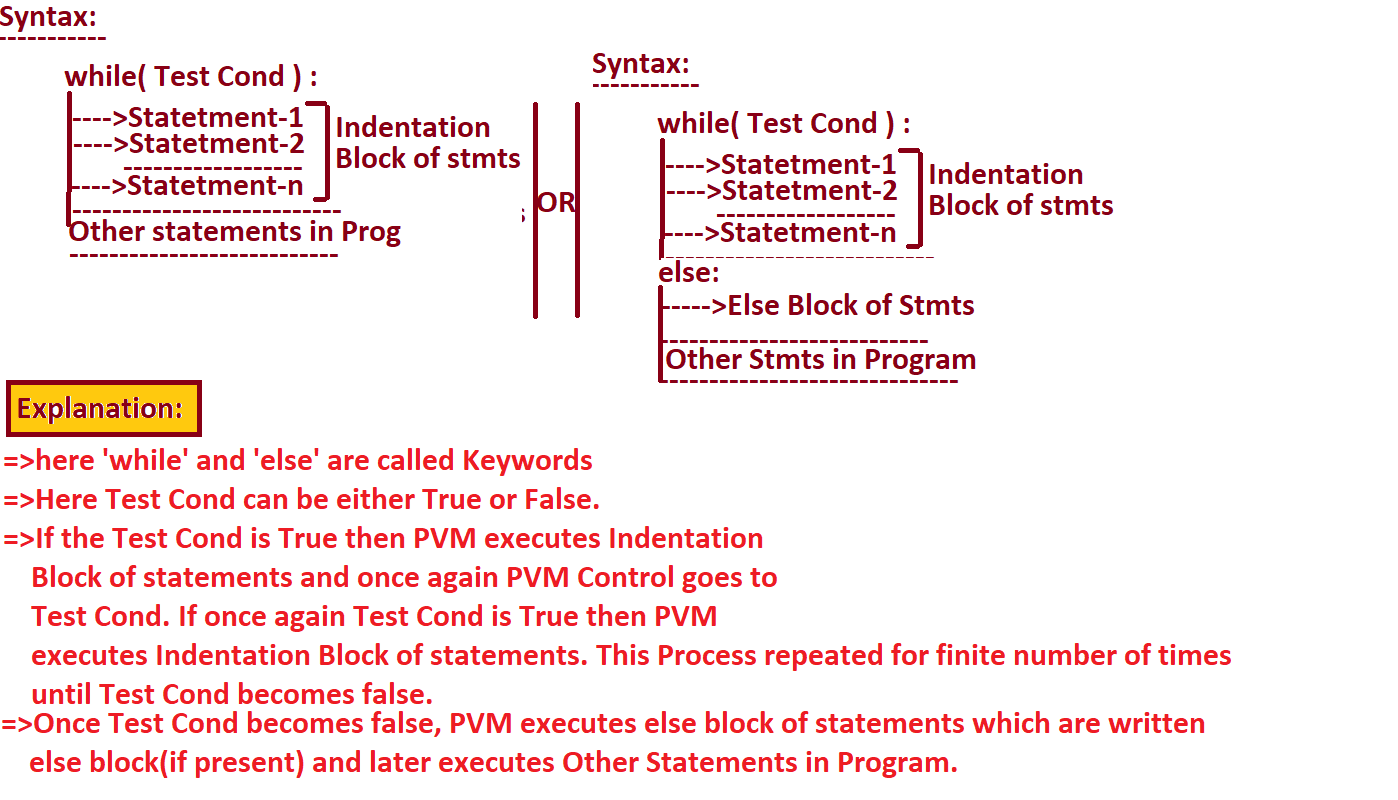
=>At the time writing Looping Level Programs, we must ensure 3 Points. They are

1. Initlization Part ( From Where to Start )

2. Condition Part ( Where to Stop)

3. Updation Part (Incr OR Decr )

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



**#program for generating 1 to n where n is +ve**

**#WhileLoopEx1.py**

n=int(input("Enter How Many Numbers u want to generate:"))

if(n<=0):

print("{} is Invalid Input".format(n))

else:

i=1 # Initlization Part

print("----------------------------------------")

print("Numbers within:{}".format(n))

print("----------------------------------------")

while(i<=n):

print("\t\t{}".format(i))

i+=1

else:

print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

print("Other Statement")

**#program for generating n to 1 where n is +ve**

**#WhileLoopEx2.py**

n=int(input("Enter How Many Numbers to want to generate:"))

if(n<=0):

print("{} is Invalid Input".format(n))

else:

print("\*"\*50)

print("Numbers from {} to 1".format(n))

print("\*" \* 50)

while(n>=1):

print("\t\t{}".format(n))

n-=1

print("\*" \* 50)

**#program for generating odd numbers from n to 1 where n is +ve**

**#WhileLoopEx3.py**

n=int(input("Enter How Many Odd Numbers to want to generate:"))

if(n<=0):

print("{} is invalid Input".format(n))

else:

if(n%2==0):

n=n-1

while(n>=1):

print("\t\t{}".format(n))

n=n-2

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

**2. for loop or for ...else loop**

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

**Syntax1:-**

-----------

for varname in Iterable\_object:

----------------------------------------

Indentation block of stmts

----------------------------------------

---------------------------------------------------

Other statements in Program

---------------------------------------------------

---------------

**Syntax2:**

---------------

for varname in Iterable\_object:

----------------------------------------

Indentation block of stmts

----------------------------------------

else:

----------------------------------------

else block of statements

----------------------------------------

---------------------------------------------------

Other statements in Program

---------------------------------------------------

**Explanation:**

-----------------------

=>Here 'for' , "in" and 'else' are keywords

=>Here Iterable\_object can be Sequence(bytes,bytearray,range,str),

list(list,tuple),set(set,frozenset) and dict.

=>The execution process of for loop is that " Each of Element of Iterable\_object selected , placed in varname and executes Indentation block of statements".This Process will be repeated until all elements of Iterable\_object completed.

=> when all the elements of Iterable Object completed then PVM comes out of for loop and executes else block of statements which are written under else block

=>Writing else block is optional.

======================================x====================================

**#ForLoopEx1.py**

s=input("Enter a Line of Text:")

print("By using While Loop")

i=0 # Initlization

while(i<len(s)): # Test Cond

print("\t{}".format(s[i]))

i=i+1 # Updation

print("----------------OR----------------------")

print("By using for loop")

for ch in s:

print("\t{}".format(ch))

else:

print("else part of for loop")

#ForLoopEx2.py

lst=[10,"Rossum",34.56,True,2+3j]

print("By using while loop")

i=0

while(i<len(lst)):

print("\t\t{}".format(lst[i]))

i+=1

else:

print("else part of while loop")

print("+++++++++++OR+++++++++++++++")

print("By using for loop")

for value in lst:

print("\t\t{}".format(value))

**#ForLoopEx3.py**

lst=[10,"Rossum",34.56,True,2+3j]

print("By using while loop")

l=len(lst)-1

while(l>=0):

print("\t\t{}".format(lst[l]))

l=l-1

print("+++++++++++OR+++++++++++++++")

print("By using for loop with Sliicing")

for value in lst[::-1]:

print("\t\t{}".format(value))

print("+++++++++++OR+++++++++++++++")

print("By using for loop without Sliicing")

for index in range(len(lst)-1, -1 , -1):

print("\t\t{}".format(lst[index]))

**#Program for generating 1 to n numbers by using for loop**

**#ForLoopEx4.py**

n=int(input("Enter How Many Numbers u want to generate:"))

if(n<=0):

print("{} is Invalid".format(n))

else:

print("-"\*50)

print("Numbers within {}".format(n))

print("-"\*50)

for i in range(1,n+1):

print("\t\t{}".format(i))

else:

print("-"\*50)

**#Program for generating n to 1 numbers by using for loop**

#ForLoopEx5.py

n=int(input("Enter How Many Numbers u want to generate:"))

if(n<=0):

print("{} is Invalid".format(n))

else:

print("-"\*50)

print("Numbers from {} to 1".format(n))

print("-"\*50)

for i in range(n,0,-1):

print("\t\t{}".format(i))

else:

print("-" \* 50)

**#Program for generating all odd numbers 1 to n numbers by using for loop**

**#ForLoopEx6.py**

n=int(input("Enter How Many Range of Odd Numbers u want to generate:"))

if(n<=0):

print("{} is Invalid".format(n))

else:

print("-"\*50)

print("Odd Numbers within ".format(n))

print("-"\*50)

for i in range(1,n+1,2):

print("\t\t{}".format(i))

else:

print("-" \* 50)

**#Program for generating all even numbers from n to 1 numbers by using for loop**

**#ForLoopEx6.py**

n=int(input("Enter How Many Range of Even Numbers u want to generate:"))

if(n<=0):

print("{} is Invalid".format(n))

else:

print("-" \* 50)

print("Even Numbers {} from 2 ".format(n))

print("-" \* 50)

if(n%2!=0):

n=n-1

for i in range(n,1,-2):

print("\t\t{}".format(i))

else:

print("-" \* 50)

**#Program for generating Mul Table for a Given +Ve Numbers**

**#ForLoopEx8.py**

n=int(input("Enter a Number for generating mul table:"))

if(n<=0):

print("{} is Invalid".format(n))

else:

print("\*"\*50)

print("\t\tMul table for {}".format(n))

print("\*" \* 50)

for i in range(1,11):

print("\t\t {} x {}={}".format(n,i,n\*i))

else:

print("\*" \* 50)

**#Program for finding sum of n natural numbers**

**#ForLoopEx9.py**

n=int(input("Enter How Many Natural Numbers sum u want:"))

if(n<=0):

print("{} is invalid input".format(n))

else:

print("="\*50)

print("Sum of {} Natural Numbers".format(n))

print("=" \* 50)

s=0 # Initial Value for Acuumutaing Natural nums sum

for i in range(1,n+1):

print("\t\t{}".format(i))

s=s+i

else:

print("=" \* 50)

print("\t\tsum={}".format(s))

print("=" \* 50)

#program obtraining length of words of line of text

#CountWordLengthEx1.py

line=input("Enter Line of Text:")

words=line.split()

print("----------------------------------------------")

print("Given Line:{}".format(line)) # Apple is in red

print("Given Words={}".format(words)) # words=["Apple","is","in","red"]

for word in words:

print("\t\t{}--->{}".format(word,len(word)))

print("----------------------------------------------")

**#Program for finding sum of digits of Given Number**

#DigitsSumEx1.py

num=int(input("Enter Any Number:"))

if(num<=0):

print("{} is Invalid Input".format(num))

else:

snum=str(num)

s=0

for dig in snum:

s=s+int(dig)

else:

print("Sum of Digits({})={}".format(num,s) )

**#Program for finding sum of digits of Given Numb**er

#DigitsSumEx1.py

num=int(input("Enter Any Number:"))

if(num<=0):

print("{} is Invalid Input".format(num))

else:

tnum=num# we are preserving the original number into temp Var

s=0

while(num>0):

d=num%10

s=s+d

num=num//10

else:

print("sum of digits({})={}".format(tnum,s))

**#Program for finding Product of n natural numbers**

#NatNumsProduct.py

n=int(input("Enter How Many Natural Numbers Product u want:"))

if(n<=0):

print("{} is invalid input".format(n))

else:

print("="\*50)

print("{} Natural Numbers Product ".format(n))

print("=" \* 50)

s=1 # Here 1 is Multiplicate Identity

for i in range(1,n+1):

print("\t\t{}".format(i))

s=s\*i # Accumulating the product First 'n' natural Numbers

else:

print("Product={}".format(s))

print("=" \* 50)

**#Program for finding sum of Squares of n natural numbers**

#NatNumsSquaresSum.py

n=int(input("Enter How Many Natural Numbers Squares sum u want:"))

if(n<=0):

print("{} is invalid input".format(n))

else:

print("="\*50)

print("{} Natural Numbers Sum of Squares and Cubes ".format(n))

print("=" \* 50)

print("NatNums\t\tNatNumsSquares\t\tNatNumsCubes")

print("-"\*60)

s,ss,cs=0,0,0

for i in range(1,n+1):

print("\t{}\t\t\t\t{}\t\t\t\t\t\t\t\t{}".format(i,i\*\*2,i\*\*3))

s=s+i

ss=ss+i\*\*2

cs=cs+i\*\*3

else:

print("-" \* 60)

print("\t{}\t\t\t{}\t\t\t\t\t\t\t\t{}".format(s,ss,cs))

print("-" \* 60)

**#program obtaining Reverse of Given Value without using Slicing and reverse()**

#ValueReverseEx1.py

value=input("Enter Any Value:") # Python

print("Slicing---Reverse Value({})={}".format(value,value[::-1]))

print("------------------OR-------------------------------------")

#Convert value into list

l1=list(value)

l1.reverse()

rv=""

rv=rv.join(l1)

print("reverse()---Reverse Value({})={}".format(value,rv))

print("------------------By using Logic-------------------------------------")

#S=PYTHON

rv=""

for i in range(len(value)-1, -1,-1):

rv=rv+value[i]

print("By logic-for loop---Reverse Value({})={}".format(value, rv))

print("------------------By using Logic-------------------------------------")

#S=PYTHON

rvv=""

i=len(value)-1

while(i>=0):

rvv+=value[i]

i-=1

print("By logic-while loop---Reverse Value({})={}".format(value, rvv))

**#Program for Cal Factorial of a Number**

# Logic-1 : n! = 1 x 2 x 3 x 4.....x n

#FactEx1.py

n=int(input("Enter the number for Cal Factorial:"))

if(n<0):

print("{} is Invalid Input--Factorial Cal is Not Possible".format(n))

else:

fact=1

for i in range(1,n+1):

fact\*=i # Here \*= is called Short hand Operator

else:

print("Fact({})={}".format(n,fact))

**#Program for Cal Factorial of a Numbe**r

# Logic-1 : n! = n x n-1 x n-2......1

#Examples: 4! = 4 x 3 x 2 x 1

#FactEx2.py

n=int(input("Enter the number for Cal Factorial:"))

if(n<0):

print("{} is Invalid Input--Factorial Cal is Not Possible".format(n))

else:

fact=1

for i in range(n,0,-1):

fact\*=i

else:

print("\t\t{}!={}".format(n,fact))

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

III. Transfer Flow Statements

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

=>The purpose of Transfer Flow Statements is that "To Change the control of PVM from One Part of the Program to another Part of the Program".

=>In Python Programming, we have 4 types of Transfer Flow Statements. They are

1. break

2. continue

3. pass

4. return

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

1. break statement

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

=>break is a key word

=>break keyword must be used always inside of loops otherwise we get Syntax error

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

=>when break statement takes place inside for loop or while loop then PVM will not execute corresponding else block(bcoz loop is not becoming False) but it executes other statements in the program

=>Syntax1:

-------------------

for varname in Iterable\_object:

------------------------------

if (test cond):

break

------------------------------

------------------------------

------------------

=>Syntax2:

-------------------

while(Test Cond-1):

------------------------------

if (test cond-2):

break

------------------------------

------------------------------

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

-------------------------------------

2. 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 inside of loop then 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

-----------------------------------------

-----------------------------------------

=>Syntax:-

----------------

while (Test Cond):

------------------------------------------

if ( Test Cond):

continue

statement-1 # written after continue stateemnt

statement-2

statement-n

-----------------------------------------

-----------------------------------------

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

#Program for Demonstrating break statement

#BreakStmtEx1.py

s="PYTHON"

print("By using for loop--without break stmt")

for ch in s:

print("\t{}".format(ch))

else:

print("i am from else part of for loop")

print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

#Here My Requirment is to Display Only PYTH without using Indexing and Slicing

print("By using for loop--with break stmt")

for ch in s: # s="PYTHON"

if(ch=="O"):

break

print("\t\t{}".format(ch))

else:

print("i am from else part of for loop")

print("Other Statements in program")

#Program for Demonstrating break statement

#BreakStmtEx2.py

s="PYTHON"

print("By using while loop--without break stmt")

i=0

while(i<len(s)):

print("\t{}".format(s[i]))

i=i+1

else:

print("i am from else part of while loop")

print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

#Here My Requirment is to Display Only PYT without using Indexing and Slicing

print("By using while loop--with break stmt")

i=0

while(i<len(s)): # s="PYTHON"

if(s[i]=="H"):

break

print("\t\t{}".format(s[i]))

i=i+1

else:

print("i am from else part of while loop")

print("Other Statements in program")

#Program for Prime Number

#BreakStmtEx3.py

n=int(input("Enter any Number:")) # 9

if(n<=1):

print("{} is Invalid Number".format(n))

else:

res="Prime"

for i in range(2,n): # n=5

if(n%i==0):

res="Not Prime"

break

if(res=="Prime"):

print("{} is Prime".format(n))

else:

print("{} is Not Prime".format(n))

#Program for Prime Number

#BreakStmtEx4.py

n=int(input("Enter any Number:")) # 9

if(n<=1):

print("{} is Invalid Number".format(n))

else:

res=True

for i in range(2,n): # n=5

if(n%i==0):

res=False

break

if(res):

print("{} is Prime".format(n))

else:

print("{} is Not Prime".format(n))

#Program for Prime Number

#BreakStmtEx5.py

n=int(input("Enter any Number:")) # -7

if(n<=1):

print("{} is Invalid Number".format(n))

else:

res=1

for i in range(2,n):

if(n%i==0):

res=0

break

if(res):

print("{} is Prime".format(n))

else:

print("{} is Not Prime".format(n))

#Program for Validation of Name

#BreakStmtEx6.py

name=input("Enter Ur Name:") # name = Mohan2 Kumar

print("Ur Name:{}".format(name))

#split name into words

namewords=name.split()

if(len(namewords)==0):

print("{} is Invalid Valid Name:".format(name))

else:

valid=True

for word in namewords:

if(not word.isalpha()):

valid=False

break

if(valid):

print("{} is Valid Name:".format(name))

else:

print("{} is Invalid Valid Name:".format(name))

#ContinueStmtEx1.py

s="PYTHON"

print("By using for loop---without using continue stmt")

for ch in s:

print("\t\t{}".format(ch))

else:

print("i am from else part of for loop")

print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

#My Requirment today is "PYHON"

print("By using for loop---without using continue stmt")

for ch in s:

if(ch=="T"):

continue

print("\t\t{}".format(ch))

else:

print("i am from else part of for loop")

#ContinueStmtEx2.py

s="PYTHON"

print("By using for loop---without using continue stmt")

for ch in s:

print("\t\t{}".format(ch))

else:

print("i am from else part of for loop")

print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

#My Requirment today is "PYHN"

print("By using for loop---without using continue stmt")

for ch in s:

if(ch=="T") or (ch=="O"):

continue

print("\t\t{}".format(ch))

else:

print("i am from else part of for loop")

#ContinueStmtEx3.py

s="PYTHON"

print("By using while loop---without using continue stmt")

i=0

while(i<len(s)):

print("\t\t{}".format(s[i]))

i=i+1

else:

print("i am from else part of for loop")

print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

#My Requirment today is "PYHN"

print("By using while loop---without using continue stmt")

i=0

while(i<len(s)): # S="PYTHON"

if(s[i]=="T") or (s[i]=="O"):

i=i+1

continue

print("\t\t{}".format(s[i]))

i=i+1

else:

print("i am from else part of for loop")

#ContinueStmtEx4.py

s="MISSISSIPPI"

for index, char in enumerate(s):

if(index==4):

continue

print("\t\t{}".format(char))

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

Nested OR Inner Loops =========================================================

=>The Process of Defining One Loop inside of another loop is called Inner OR Nested Loop

=>The Execution Process of Nested OR Inner Loops is that "For Every Value of Outer loop, Inner Loop executes for Finite

Numbers until Test Cond becomes False".

=>In Python Programming, we can use Inner loops in 4 ways. They are

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

Syntax1: for loop in for loop

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

for varname in Iterable-Object1: # Outer loop

------------------------------------------

------------------------------------------

for varname in Iter bale-object2: # Inner loop

----------------------------------

----------------------------------

else:

-------------------------------------

-------------------------------------

else:

-----------------------------------------

-----------------------------------------

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

Syntax2: while loop in while loop

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

----------------------------------------------------

while(Test Cond1): # Outer Loop

-------------------------------------------

-------------------------------------------

while(Test Cond2): # Inner Loop

-----------------------------------

------------------------------------

else:

-------------------------------------

-------------------------------------

else:

--------------------------------------------

--------------------------------------------

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

Syntax3: while loop in for loop

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

----------------------------------------------------

for Varname in Iterbale-Object: # Outer Loop

-------------------------------------------

-------------------------------------------

while(Test Cond2): # Inner Loop

-----------------------------------

------------------------------------

else:

-------------------------------------

-------------------------------------

else:

--------------------------------------------

--------------------------------------------

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

Syntax4: for loop in while loop

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

----------------------------------------------------

while(Test Cond1): # Outer Loop

-------------------------------------------

-------------------------------------------

for varname in Iterable-Object: # Inner Loop

-----------------------------------

------------------------------------

else:

-------------------------------------

-------------------------------------

else:

--------------------------------------------

--------------------------------------------

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

#Program for reading List of Values and obtain +Ve values

#ContinueStmtEx5.py

n=int(input("Enter How Many Values:"))

if(n<=0):

print("{} is Invalid Input".format(n))

else:

lst=[] # create an empty list

for i in range(1,n+1):

value=int(input("Enter {} Value: ".format(i)))

lst.append(value)

else:

print("List of Values:{}".format(lst)) # [20, -45, 100, -56, 0, 23, -67, -45, 78, 0]

#get the +ve Values

pslist=list() # create an empty list for placing +ve Values

for value in lst:

if(value<=0):

continue

pslist.append(value)

else:

print("List of +Ve Values={}".format(pslist))

#get the -Ve Values

nglist=list() # create an empty list for placing -ve Values

for value in lst:

if(value>=0):

continue

nglist.append(value)

else:

print("List of -Ve Values={}".format(nglist))

#program for Demonstrating Inner Loops

#InnerLoopEx1.py

for i in range(1,6): # Outer Loop

print("Outer Loop Val of i={}".format(i))

for j in range(1,4): # Inner Loop

print("\t\tInner Loop--Value of j={}".format(j))

else:

print("I am from else part-inner for Loop")

else:

print("I am from Outer-for Loop")

#program for Demonstrating Inner Loops

#InnerLoopEx2.py

i=1

while(i<=5): # Outer Loop

print("Outer Loop Value of i={}".format(i))

j=1

while(j<=3): # Inner Loop

print("\t\tInner Loop--Value of j={}".format(j))

j=j+1

else:

print("\tI am from else part of inner while loop")

i=i+1

else:

print("I am from else of while loop")

#program for Demonstrating Inner Loops

#InnerLoopEx3.py

for i in range(5,0,-1): # Outer Loop

print("Outer Loop Value of i={}".format(i))

j=1

while(j<=3): # Inner Loop

print("\t\tInner Loop--Value of j={}".format(j))

j=j+1

else:

print("\tI am from else part of inner while loop")

else:

print("I am from else of for loop")

#program for Demonstrating Inner Loops

#InnerLoopEx4.py

i=5

while(i>=1): # Outer Loop

print("Outer Loop Val of i={}".format(i))

for j in range(3,0,-1): # Inner Loop

print("\t\tInner Loop--Value of j={}".format(j))

else:

print("I am from else part-inner for Loop")

i=i-1

else:

print("I am from Outer-while Loop")

#Program generating 1 to n mul tables where n is +ve

#InnerLoopEx5.py

n=int(input("Enter How Many Mul tables u want:"))

if(n<=0):

print("{} is invalid input".format(n))

else:

for num in range(1,n+1): # Outer Loop supply the Number for gen Mul table

print("------------------------------------------")

print("Mult Table for {}".format(num))

print("------------------------------------------")

for i in range(1,11): # Inner loop--generates for Mul table for that number supplied by outer loop

print("\t\t{} x {} = {}".format(num,i,num\*i))

else:

print("------------------------------------------")

**#Program generating random numbers mul tables**

#InnerLoopEx6.py

n=int(input("Enter How Many random Mul tables u want:"))

if(n<=0):

print("{} is invalid input".format(n))

else:

lst=[]

for i in range(1,n+1):

value=int(input("Enter {} Value:".format(i)))

lst.append(value)

else:

print("List of Values:{}".format(lst))

#Code for generating mul tables for every value of list

for num in lst: # Outer loop

if(num<=0):

print("{} Is Invalid Input".format(num))

continue

print("-"\*40)

print("Mul Table for {}".format(num))

for i in range(1,11): # Inner loop

print("\t\t{} x {} = {}".format(num,i,num\*i))

else:

print("-" \* 40)

**#program for generating list of primes within the range**

#InnerLoopEx7.py

n=int(input("Enter the range in which u want Prime Numbers:"))

if(n<=1):

print("{} is invalid Range:".format(n))

else:

primelist=[]

for num in range(2,n+1): # Suppy the Numbers

res=True

for i in range(2,num): # Decide the whether supplied number is prime or not

if(num%i==0):

res=False

break

if(res):

primelist.append(num)

else:

print("List of Primes:{}".format(primelist))

#InnerLoopEx7.py

n = int(input("Enter the range in which u want Prime Numbers:"))

if (n <= 1):

print("{} is invalid Range:".format(n))

else:

lst = []

for i in range(1, n + 1):

value = int(input("Enter {} Value:".format(i)))

lst.append(value)

else:

print("List of Values:{}".format(lst))

#Program form generating Student marks Report

#StudentMarksReportEx1.py

#validation of sno

while(True):

sno=input("Enter Student Number:")

if(sno.isalnum()) and (not sno.isdigit()):

print("{} is Invalid Name--Try again".format(sno))

elif(sno.isdigit()):

break

elif(sno.isspace()):

print("{} is Invalid Number--Try again".format(sno))

#validation of name

while(True):

sname=input("Enter Ur Name:") # name = Mohan2 Kumar

#split name into words

namewords=sname.split()

if(len(namewords)==0):

print("{} is Invalid Valid Name:".format(sname))

else:

valid=True

for word in namewords:

if(not word.isalpha()):

valid=False

break

if(valid):

#print("{} is Valid Name:".format(sname))

break

else:

print("{} is Invalid Valid Name:".format(sname))

#Validation of Marks in C---- 0 to 100

while(True):

cm=int(input("Enter Marks in C Lang:"))

if(cm>=0) and (cm<=100):

break

print("{} is invalid Marks in C Lang:".format(cm))

#Validation of Marks in CPP---- 0 to 100

while(True):

cppm=int(input("Enter Marks in C++ Lang:"))

if(cppm>=0) and (cppm<=100):

break

print("{} is invalid Marks in C Lang:".format(cppm))

#Validation of Marks in Python---- 0 to 100

while(True):

pym=int(input("Enter Marks in PYTHON Lang:"))

if pym in range(0,101):

break

print("{} is invalid Marks in Python Lang:".format(pym))

#Calculate totmarks and percenatge

totmarks=cm+cppm+pym

percent=(totmarks/300)\*100

if(cm<40) or (cppm<40) or (pym<40):

grade="FAIL"

else:

if(totmarks>=250) and (totmarks<=300):

grade="DISTINCTION"

elif(200<=totmarks<=249):

grade="FIRST"

elif(totmarks in range(150,200)):

grade="SECOND"

elif(totmarks in range(120,150)):

grade="THIRD"

#Display Student Report

print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

print("\t\tStudent Marks Report")

print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

print("\t\tStudent Number:{}".format(sno))

print("\t\tStudent Name:{}".format(sname))

print("\t\tStudent Marks in C:{}".format(cm))

print("\t\tStudent Marks in C++:{}".format(cppm))

print("\t\tStudent Marks in Python:{}".format(pym))

print("\t\tStudent Total Marks :{}".format(totmarks))

print("\t\tStudent Percentage of Marks:{}".format(round(percent,2)))

print("\t\tSTUDENT GRADE: {}".format(grade))

print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

**#Program form generating Student marks Report**

#StudentMarksReportEx2.py

while(True):

#validation of sno

while(True):

sno=input("Enter Student Number:")

if(sno.isalnum()) and (not sno.isdigit()):

print("{} is Invalid Name--Try again".format(sno))

elif(sno.isdigit()):

break

elif(sno.isspace()):

print("{} is Invalid Number--Try again".format(sno))

#validation of name

while(True):

sname=input("Enter Ur Name:") # name = Mohan2 Kumar

#split name into words

namewords=sname.split()

if(len(namewords)==0):

print("{} is Invalid Valid Name:".format(sname))

else:

valid=True

for word in namewords:

if(not word.isalpha()):

valid=False

break

if(valid):

#print("{} is Valid Name:".format(sname))

break

else:

print("{} is Invalid Valid Name:".format(sname))

#Validation of Marks in C---- 0 to 100

while(True):

cm=int(input("Enter Marks in C Lang:"))

if(cm>=0) and (cm<=100):

break

print("{} is invalid Marks in C Lang:".format(cm))

#Validation of Marks in CPP---- 0 to 100

while(True):

cppm=int(input("Enter Marks in C++ Lang:"))

if(cppm>=0) and (cppm<=100):

break

print("{} is invalid Marks in C Lang:".format(cppm))

#Validation of Marks in Python---- 0 to 100

while(True):

pym=int(input("Enter Marks in PYTHON Lang:"))

if pym in range(0,101):

break

print("{} is invalid Marks in Python Lang:".format(pym))

#Calculate totmarks and percenatge

totmarks=cm+cppm+pym

percent=(totmarks/300)\*100

if(cm<40) or (cppm<40) or (pym<40):

grade="FAIL"

else:

if(totmarks>=250) and (totmarks<=300):

grade="DISTINCTION"

elif(200<=totmarks<=249):

grade="FIRST"

elif(totmarks in range(150,200)):

grade="SECOND"

elif(totmarks in range(120,150)):

grade="THIRD"

#Display Student Report

print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

print("\t\tStudent Marks Report")

print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

print("\t\tStudent Number:{}".format(sno))

print("\t\tStudent Name:{}".format(sname))

print("\t\tStudent Marks in C:{}".format(cm))

print("\t\tStudent Marks in C++:{}".format(cppm))

print("\t\tStudent Marks in Python:{}".format(pym))

print("\t\tStudent Total Marks :{}".format(totmarks))

print("\t\tStudent Percentage of Marks:{}".format(round(percent,2)))

print("\t\tSTUDENT GRADE: {}".format(grade))

print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

ch=input("Do u want to enter another student data(yes/no):")

if(ch.lower()=="no"):

print("thx for using this program")

break

#VoterEx1.py

age=int(input("Enter Age of Citizen:"))

if(age>=18):

print("{} Years Citizen is Eligible to Vote".format(age))

else:

print("{} Years Citizen is not Eligible to Vote".format(age))

#VoterEx2.py

while(True):

age=int(input("Enter Age of Citizen:")) # 16

if(age>=18) and (age<=100):

print("{} Years Citizen is Eligible to Vote".format(age))

break

print("\t{} Years Invalid Age--try again!!".format(age))

**=============================================================**

**Functions in Python---8 Days**

**Functions(8)-------->Modules(2)---->Packages (1)**

**=============================================================**

Index

-------------------------------------------------------------------------------------------------------

=>Types of Languages

a) Un-Structured Programming Languages

b) Structured Programming Languages

=>Purpose of Functions

=>Definition of Function

=>Parts of Functions

=>Phases in Functions

=>Advantages of Functions

=>Types of Functions

a) Pre-Defined Functions

b) Programmer-Defined Functions

=>Syntax for defining Functions

=>Number of approaches to define a Function for Problem Solving

=>Programming Examples

------------------------------------------------------------------------------------------------------

=>Arguments and Parameters

=>Types of Arguments and Parameters

a) Possitional Arguments

b) Default Arguments

c) Keyword Arguments

d) Variable Length Arguments

e) Keyword Variable Length Arguments

=>Programming Examples

-----------------------------------------------------------------------------------------------------

=>Local and Global Variables

=>Programming Examples

**=>global Keyword and Programming Examples**

**=>globals() and Programming Examples**

----------------------------------------------------------------------------------------------------

=>**Anonymous OR Lambda Functions**

**=>Implementation of Anonymous OR Lambda Functions**

**=>Programming Examples**

----------------------------------------------------------------------------------------------------

**=>Comprehension in Python**

**a) List Comprehension**

**b) Set Comprehension**

**c) Dict Comprehension**

**d) tuple Comprehension (Not there---Programming)**

**=>Programming Examples**

----------------------------------------------------------------------------------------------------

=>Special Functions in Python

**a) filter() with Normal and Anonymous Functions**

**b) map() with Normal and Anonymous Functions**

**c) reduce() with Normal and Anonymous Functions**

=>Programming Examples

========================================x==================================

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

Types of Languages

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

=>In IT we have Two types of Programming Languages. They are

1) Un-Structured Programming Languages

2) Structured Programming Languages

-------------------------------------------------------------------------------------------------------------------------------1) Un-Structured Programming Languages

-------------------------------------------------------------------------------------------------------------------------------

=>Un-Structured Programming Languages **does not contain the concept of Functions**. So that the applications OR Programs related to Un-Structured Programming Languages has the following Limitations.

1. Application development time is More

2. Application Memory Space is More

3. Application execution time is More

4. Performance of the application is degraded

5. Redundancy (Duplication or Replication) of the code is More

Examples: GW-BASIC

-------------------------------------------------------------------------------------------------------------------------------

2) Structured Programming Languages

-------------------------------------------------------------------------------------------------------------------------------

=>Structured Programming Languages contains the concept of Functions. So that the applications OR Programs related to Structured Programming Languages has the following Advantages.

1. Application development time is Less

2. Application Memory Space is Less

3. Application execution time is Less

4. Peformance of the application is Enhnaced (Improved)

5. Redundency (Duplication or Replication) of the code is Minimized

Examples: C, C++, Java, PYTHON, C#.net.................etc

-------------------------------------------------------------------------------------------------------------------------------

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

Functions in Python

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

=>The purpose of Functions is that " To Perform Certain Operation /Task and Provides Code Re-Usability".

=>The Advantages of Functions in any languages are

1. Application Development time is Less

2. Application Memory Space is Less

3. Application Execution Time is Less

4. Application Performance is Enhanced

5. Redundancy of the Code is Minimized

-------------------------------------------------------------------------------------------------------------------------------

Definitions of Function

-------------------------------------------------------------------------------------------------------------------------------

=>Sub Program of Main Program is Called Function.

(OR).

=>A Part of main program is Called Function.

-------------------------------------------------------------------------------------------------------------------------------

Parts of Functions

-------------------------------------------------------------------------------------------------------------------------------

=>At the time Developing functions in real time, we must ensure that, there must exist 2 Parts. they are

**1. Function Definition**

**2. Function Calls**

=>Every Function Definition Exists Only Once

**=>Every Function call must contains a Function Definition Otherwise we get NameError.**

=>Function Definition will execute when we call by using function calls OR Without calling the

Function by using Function Calls, PVM will not execute Function Definition.

-------------------------------------------------------------------------------------------------------------------------------

Phases in Functions

-------------------------------------------------------------------------------------------------------------------------------

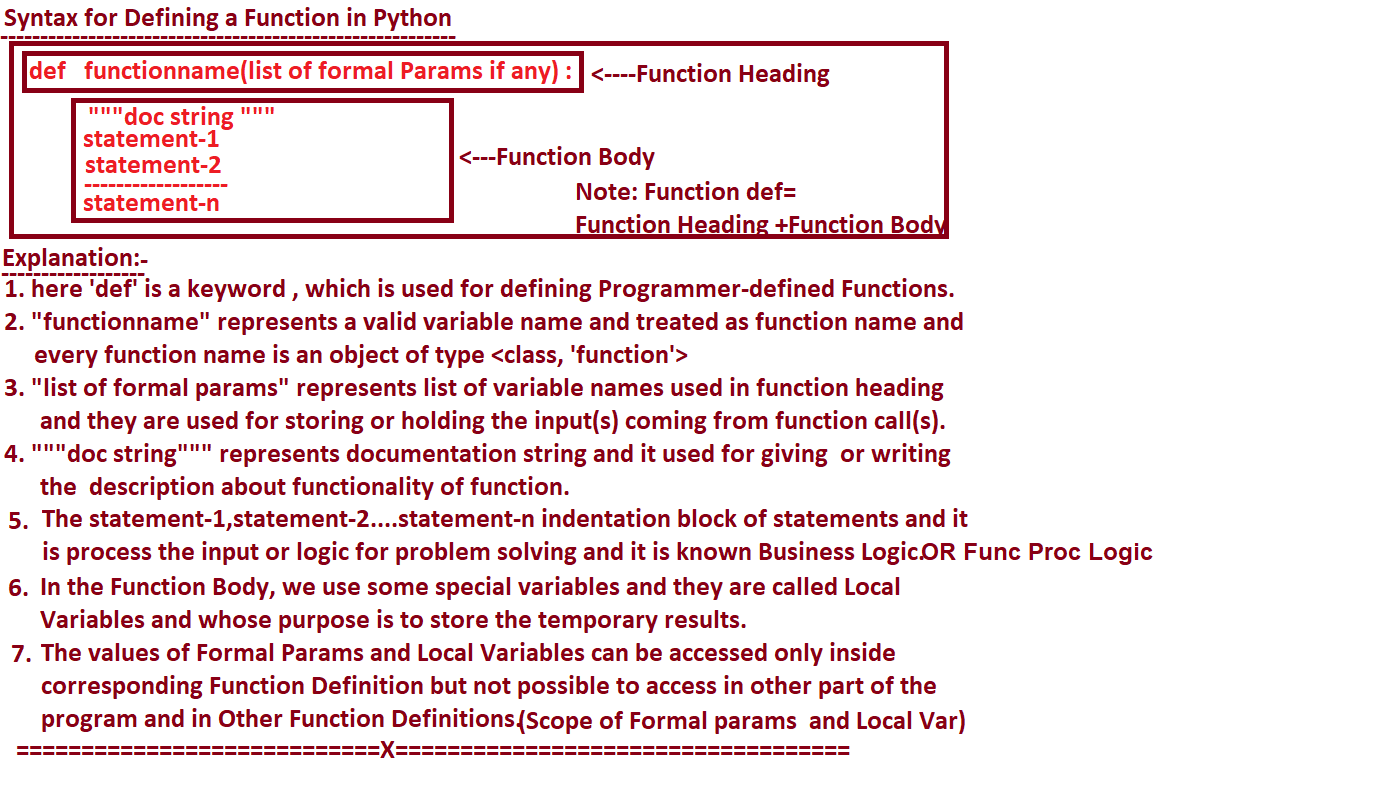
=>At the time Defining the functions, the Programmer must ensure that there must exist the following Phases.

1. Every Function Must take INPUT

2. Every Function Must PROCESS the Input

3. Every Function Must give OUTPUT or RESULT

------------------------------------------------------------------------------------------------------------------------------



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

Number of approaches to define a Function for Problem Solving

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

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Approach-1

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

INPUT : Through Function calls

PROCESS : Inside of Function Body

RESULT : Holding in Function Call

Examples:Approach1.py

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Approach-2

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

INPUT : Inside of Function Body

PROCESS : Inside of Function Body

RESULT : Inside of Function Body

Example: Approach2.py

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Approach-3

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

INPUT : Through Function calls

PROCESS : Inside of Function Body

RESULT : Inside of Function Body

Examples: Approach3.py

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Approach-4

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

INPUT : Inside of Function Body

PROCESS : Inside of Function Body

RESULT : Holding in Function Call

Examples: Approach4.py

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

#Function for adding of Twe Numbers

#Approach1.py

#INPUT : Through Function calls

#PROCESS : Inside of Function Body

#RESULT : Holding in Function Call

def addop(a,b): # Here 'a' and ''b' are called formal Parameters

c=a+b # Here 'c' is called Local Variable

return c # here return is kwd used for giving the result of local var values to the main program

#main program

x=float(input("Enter First Value:"))

y=float(input("Enter Second Value:"))

z=addop(x,y) # Function call

print("sum({},{})={}".format(x,y,z))

print("-----------------OR-------------------------")

a=float(input("Enter First Value:"))

b=float(input("Enter Second Value:"))

c=addop(a,b) # Function call

print("sum({},{})={}".format(a,b,c))

#Function for adding of Twe Numbers

#Approach2.py

#INPUT : Inside of Function Body

#PROCESS : Inside of Function Body

#RESULT : Inside of Function Body

def addop():

#Taking Input

a=float(input("Enter First Value:"))

b=float(input("Enter Second Value:"))

#Processing

c=a+b

#Result

print("sum({},{})={}".format(a,b,c))

#main program

addop() # Function Call

#Function for adding of Two Numbers

#Approach3.py

#INPUT : Through Function calls

#PROCESS : Inside of Function Body

#RESULT : Inside of Function Body

def addop(a,b):

#Processing

c=a+b

#Result

print("sum({},{})={}".format(a,b,c))

#main program

k=float(input("Enter First Value:"))

v=float(input("Enter Sedond Value:"))

addop(k,v) # Function Call

#Function for adding of Two Numbers

#Approach4.py

#INPUT : Inside of Function Body

#PROCESS : Inside of Function Body

#RESULT : Holding in Function Call

def addop():

# Taking Input

a = float(input("Enter First Value:"))

b = float(input("Enter Second Value:"))

#Process

c=a+b

return a,b,c # here return can return One OR More Number of Values

#main program

a,b,c=addop() # Function Call with Multi line assigment

print("sum({},{})={}".format(a,b,c))

print("---------------OR--------------------")

res=addop() # Function Call with Single Line Assignment

# here res is an object of type <class, tuple>

print("sum({},{})={}".format(res[0],res[1],res[2]))

print("-------------OR----------------------------")

print("sum({},{})={}".format(res[-3],res[-2],res[-1]))

#Sample.py

def greet(): # Function Definition

print("Good Morning")

def hello():

print("Monday is the Holiday")

#main program

print("I am Before Function Definition")

greet() # Function Call

print("I am from after Function Definition")

hello() # Function call

#SimpleIntEx1.py

def simpleint():

p=float(input("Enter Principle Amount:"))

t=float(input("Enter Time:"))

r=float(input("Enter Rate of Interest:"))

#Cal si and totamt

si=(p\*t\*r)/100

totamt=p+si

return p,t,r,si,totamt

#main program

**s=simpleint() # FUNCTION CALL WITH SINGLE ASSIGMENT**

**#here s is an object whose type is tuple**

print("\*"\*50)

print("\t\tPrinciple Amount:{} ".format(s[0]))

print("\t\tTime: {}".format(s[1]))

print("\t\tRate of Interest: {}".format(s[2]))

print("\t\tSIMPLE INTEREST: {}".format(s[3]))

print("\t\tTOTAL AMT TO PAY: {}".format(s[4]))

print("\*"\*50)

**#Program for Reading the list of numerical values and find sum and average**

#AddlFunEx1.py

def readvalues():

n=int(input("Enter How Many Values sum and average want to find:"))

if(n<=0):

return [] # returning empty list OR return list()

else:

lst=[] # create an empty list

for i in range(1,n+1):

val=float(input("Enter {} Value:".format(i)))

lst.append(val)

return lst # returning non-empty list

def findsumavg(lst):

if(len(lst)==0):

print("Given List is empty--can't find sum and average:")

else:

print("Given List Elements={}".format(lst)) # [10.0, 2.3, 12.0, 5.6, 0.0]

s=0

for val in lst:

s+=val

else:

print("Sum({})={}".format(lst,s))

print("Avg(({})={}".format(lst, s/len(lst)))

#main program

lst=readvalues() # Function Call

findsumavg(lst) # Function Call

**#Write a python program which will accept list of words and find there lengths and find highest length word.**

#AddlFunEx2.py

def readwords():

lst=[] # create an empty list

print("Enter Number of Words and presss @ to stop:")

while(True):

word=input()

if(word=="@"):

break

lst.append(word)

return lst # this fucntion returns List object

def findwordslength(lst):

d={} # Create an empty dict for storing Name:length Pairs

for word in lst:

d[word]=len(word)

return d # this fucntion returns dict object

def findhighestlengthword(dobj):

print("="\*40)

print("\t\tWord\tLength")

print("="\*40)

for word in dobj:

print("\t\t{}\t{}".format(word,dobj.get(word)))

print("="\*40)

#Code Finding Max Length Word(s)

mlen=max(dobj.values())

maxlenwords=[] # For Storing Max length words

for word,length in dobj.items():

if(mlen==length):

maxlenwords.append(word)

else:

print("--------------------------------------------------------")

print("Max Length Words")

print("--------------------------------------------------------")

for word in maxlenwords:

print("\t\t{}".format(word))

print("--------------------------------------------------------")

#main program

wordlist=readwords() # Function Call

dobj=findwordslength(wordlist) # Function Call taking list object as Parameter

findhighestlengthword(dobj) # Function Call Taking dict object as Parameter

"""

Sample Data

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

Word Length

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

HTML 4

JAVA 4

PYTHON 6

C 1

C++ 3

Kotlin 6

R 1

========================================"""

**#Write a python program which will accept list of words and get palindrome words and also find highest length palindrome word.**

#AddlFunEx3.py

def readwords():

lst=[] # create an empty list

print("Enter Number of Words and presss @ to stop:")

while(True):

word=input()

if(word=="@"):

break

lst.append(word)

return lst # this fucntion returns List object

def findpalindomewords(lst):

d={} # Create an empty dict for storing PalindromeName:length Pairs

for word in lst:

if(word==word[::-1]):

d[word]=len(word)

return d # this fucntion returns dict object

def findhighestlengthpalindromeword(dobj):

print("="\*40)

print("\t\tWord\tLength")

print("="\*40)

for word in dobj:

print("\t\t{}\t{}".format(word,dobj.get(word)))

print("="\*40)

#Code Finding Max Length Word(s)

mlen=max(dobj.values())

maxlenwords=[] # For Storing Max length words

for word,length in dobj.items():

if(mlen==length):

maxlenwords.append(word)

else:

print("--------------------------------------------------------")

print("Max Length Words")

print("--------------------------------------------------------")

for word in maxlenwords:

print("\t\t{}".format(word))

print("--------------------------------------------------------")

#main program

wordlist=readwords() # Function Call

dobj=findpalindomewords(wordlist) # Function Call taking list object as Parameter

findhighestlengthpalindromeword(dobj) # Function Call Taking dict object as Parameter

"""

Sample Data

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

Word Length

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

HTML 4

JAVA 4

PYTHON 6

C 1

C++ 3

Kotlin 6

R 1

========================================"""

**#Write a python program which will find whether the given number is Magic number or not**.

#AddlEx4.py

def magicornot():

n=int(input("Enter Any Number for deciding for magic number:")) # 25

sn=n\*\*2 # sn=625

res="Magic Number" if(str(n)==str(sn)[ -(len(str(n))):]) else "Not a Magic Number"

print("{} is {}".format(n,res))

#main program

magicornot()

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

Parameters and Arguments

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

--------------

Parameter:

---------------

=>The variables used in Function Heading are called Formal Parameters and They are used for Storing

the inputs coming Function Calls.

=>The Variables Used in Function Body are called Local Variables / Parameters and They are used for

Storing Temporary Results / Function Processing Logic Results.

=>The Values Formal Parameters and Local parameters can be accessed within corresponding Function

Definition but not possible to access in Other Part of the Program

Examples: def sumop(a,b): # Here a, b are called Formal parameters

c=a+b # Here c is local Parameter/Variable

----------

-----------------

Arguments:

-----------------

=>**Arguments are the variables / Values which are used as Variables in Function Calls.**

Examples: sumop(10,20) # Here 10 20 are called Argument Values

(OR)

a=10

b=20

sumop(a,b) # Here a,b are called Arguments (Actual variables OR Parameters)

=>The relationship between Arguments and Parameters is that all the Values of arguments are passing to Parameters. This Mechanism is called Arguments Passing.

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

Types of Arguments Passing Techniques OR Mechanisms

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

=>Based on the Values of Arguments of Function call passing to Formal Parameters of Function Definition, Arguments are classified into 5 Types. They are

1. Positional Arguments

2. Default Arguments

3. Keyword Arguments

4. Variable Length Arguments

5. Keyword Variable Length Arguments

=========================================x===========================

1. Posstional Arguments

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

=>Positional Arguments Mechanism is the default arguments passing mechanism used by PVM in Functions for Passing the values of Arguments of Function Call to Formal Parameters of Function Defintion.

=>Positional Arguments concept says that Every Argument Value Passing Every to Formal Parameter Based on their Posstion by maintaining Order and Meaning for Higher Accuracy. In Otherwords The number of arguments must be equal to Number of Formal Parameters.

=>Positional Arguments concept always used for Passing Specific Data from Function calls to Function Definitions.

=>PVM gives High Priority for Positional Arguments

---------------------------------

Syntax: def functionname(Param1,Param2,....Param-n): # Function Definition

-----------------------------------------------

-----------------------------------------------

Block of Statements--perform Operation

------------------------------------------------

-----------------------------------------------

-------------

Syntax: functioname(arg1,arg2,.....,arg-n) # Function call

=>Here the values of arg1,arg2,.....,arg-n of Function call are passing to Param1,Param2,....Param-n of Function Definition Respectively.

===============================================x===========================

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

2) Default Parameters (or) arguments

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

=>When there is a Common Value for family of Similar 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 arguments OR 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 parameters".

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 (Possitional ) follows default 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

=>When we specify Keyword arguments before Possitional Arguments in Function Calls(s) then we get SyntaxError: positional argument follows keyword argument

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

#Program for Demonstrating default Arguments

#DefaultArgsEx1.py

def studinfo(sno,sname,marks,cname,crs="PYTHON"): # here crs is called Defeualt Argument

print("\t{}\t{}\t{}\t{}\t{}".format(sno,sname,marks,cname,crs))

#main Program

print("\*"\*50)

print("\tSNO\tNAME\tMARKS\tCNAME\tCOURSE")

print("\*"\*50)

studinfo(10,"RS",45.67,"PSFU") # Function call with Posstional Arguments

studinfo(20,"AJ",55.67,"MUMU") # Function call with Posstional Arguments

studinfo(30,"KP",46.67,"AU") # Function call with Posstional Arguments

studinfo(40,"FD",46.17,"NU") # Function call with Posstional Arguments

studinfo(50,"KV",22.22,"OU") # Function call with Posstional Arguments

studinfo(50,"DR",66.66,"NLU","JAVA") # Function call with Posstional Arguments

studinfo(60,"SR",16.66,"MU")

print("\*"\*50)

#Program for Demonstrating default Arguments

#DefaultArgsEx2.py

def studinfo(sno,sname,marks,cname,crs="PYTHON",cnt="INDIA"): # here crs and cnt are called Defeualt Arguments

print("\t{}\t{}\t{}\t{}\t{}\t{}".format(sno,sname,marks,cname,crs,cnt))

#main Program

print("\*"\*70)

print("\tSNO\tNAME\tMARKS\tCNAME\tCOURSE\tCOUNTRY")

print("\*"\*70)

studinfo(10,"RS",45.67,"PSFU") # Function call with Posstional Arguments

studinfo(20,"AJ",55.67,"MUMU") # Function call with Posstional Arguments

studinfo(30,"KP",46.67,"AU") # Function call with Posstional Arguments

studinfo(40,"FD",46.17,"NU") # Function call with Posstional Arguments

studinfo(50,"KV",22.22,"OU") # Function call with Posstional Arguments

studinfo(50,"DR",66.66,"NLU","JAVA") # Function call with Posstional Arguments

studinfo(60,"SR",16.66,"MU") # Function call with Posstional Arguments

studinfo(70,"DT",11.66,"UU","HTML","USA") # Function call with Posstional Arguments

studinfo(80,"JB",81.66,"SU",cnt="USA") # Function call with Posstional Arguments

studinfo(70,"KM",10.66,"KU",cnt="KY",crs="JAVA") # Function call with Posstional Arguments

print("\*"\*70)

#Program for Demonstrating efault Arguments

#DefaultArgsEx3.py

def studinfo1(sno,sname,marks,cname,crs="PYTHON",cnt="USA"): # here crs iscalled Defeualt Arguments

print("\t{}\t{}\t{}\t{}\t{}\t{}".format(sno,sname,marks,cname,crs,cnt))

def studinfo2(sno,sname,marks,cname,crs="JAVA",cnt="INDIA"): # here crs is called Defeualt Arguments

print("\t{}\t{}\t{}\t{}\t{}\t{}".format(sno,sname,marks,cname,crs,cnt))

#main Program

print("\*"\*70)

print("\tSNO\tNAME\tMARKS\tCNAME\tCOURSE\tCOUNTRY")

print("\*"\*70)

studinfo1(10,"RS",45.67,"PSFU") # Function call with Posstional Arguments

studinfo1(20,"AJ",55.67,"MUMU") # Function call with Posstional Arguments

studinfo1(30,"KP",46.67,"AU") # Function call with Posstional Arguments

studinfo1(40,"FD",46.17,"NU") # Function call with Posstional Arguments

studinfo1(50,"KV",22.22,"OU") # Function call with Posstional Arguments

studinfo1(60,"SP",32.22,"OU",cnt="UK") # Function call with Posstional Arguments

print("\*"\*70)

print("\tSNO\tNAME\tMARKS\tCNAME\tCOURSE\tCOUNTRY")

print("\*"\*70)

studinfo2(50,"DR",66.66,"NLU") # Function call with Posstional Arguments

studinfo2(60,"SR",16.66,"MU") # Function call with Posstional Arguments

studinfo2(70,"DT",11.66,"UU") # Function call with Posstional Arguments

studinfo2(80,"JB",81.66,"SU") # Function call with Posstional Arguments

studinfo2(70,"KM",10.66,"KU") # Function call with Posstional Arguments

studinfo2(80,"AK",32.22,"OU",cnt="AUS") # Function call with Posstional Arguments

print("\*"\*70)

#program for Demonstrating Keywords Arguments

#KwdArgsEx1.py

def disp(a,b,c,d):

print("\t{}\t{}\t{}\t{}".format(a,b,c,d))

#main Program

print("\*"\*40)

print("\tA\tB\tC\tD")

print("\*"\*40)

disp(10,20,30,40) # Function Call with Poss. Args

disp(d=40,c=30,b=20,a=10) # Function Call with Keyword. Args

#disp(d=40,c=30,b=20,x=10) # Function Call with Keyword. Args--TypeError: disp() got an unexpected keyword argument 'x'

disp(10,c=30,d=40,b=20) # Function Call with Poss. Args with Keyword args

#disp(c=30,d=40,b=20,10) # Function Call with Poss. Args with Keyword args---SyntaxError: positional argument follows keyword argument

disp(10,20,d=40,c=30) # Function Call with Poss. Args with Keyword args

print("\*"\*40)

#program for Demonstrating Keywords Arguments

#KwdArgsEx2.py

def disp(a,b,c,d,e="HYD"):

print("\t{}\t{}\t{}\t{}\t{}".format(a,b,c,d,e))

#main Program

print("\*"\*40)

print("\tA\tB\tC\tD\tE")

print("\*"\*40)

**disp(10,20,30,40) # Function Call with Poss. Args**

**disp(d=40,c=30,b=20,a=10) # Function Call with Keyword. Args**

#disp(d=40,c=30,b=20,x=10) # Function Call with Keyword. Args--TypeError: disp() got an unexpected keyword argument 'x'

disp(10,c=30,d=40,b=20) # Function Call with Poss. Args with Keyword args

#disp(c=30,d=40,b=20,10) # Function Call with Poss. Args with Keyword args---SyntaxError: positional argument follows keyword argument

disp(10,20,d=40,c=30) # Function Call with Poss. Args with Keyword args

disp(b=20,c=30,a=10,e="AP",d=40) # Function Call with Poss. Args withwith Keyword args

print("\*"\*40)

#Program for Demonstrating Posstional Arguments

#PossArgsEx1.py

def studinfo(sno,sname,marks,cname):

print("\t{}\t{}\t{}\t{}".format(sno,sname,marks,cname))

#main Program

print("\*"\*50)

print("\tSNO\tNAME\tMARKS\tCNAME")

print("\*"\*50)

studinfo(10,"RS",45.67,"PSFU") # Function call

studinfo(20,"AJ",55.67,"MUMU") # Function call

studinfo(30,"KP",46.67,"AU") # Function call

studinfo(40,"FD",46.17,"NU") # Function call

studinfo(50,"KV",22.22,"OU") # Function call

print("\*"\*50)

#Program for Demonstrating Posstional Arguments

#PossArgsEx2.py

def studinfo(sno,sname,marks,cname,crs):

print("\t{}\t{}\t{}\t{}\t{}".format(sno,sname,marks,cname,crs))

#main Program

print("\*"\*50)

print("\tSNO\tNAME\tMARKS\tCNAME\tCOURSE")

print("\*"\*50)

studinfo(10,"RS",45.67,"PSFU","PYTHON") # Function call with Posstional Arguments

studinfo(20,"AJ",55.67,"MUMU","PYTHON") # Function call with Posstional Arguments

studinfo(30,"KP",46.67,"AU","PYTHON") # Function call with Posstional Arguments

studinfo(40,"FD",46.17,"NU","PYTHON") # Function call with Posstional Arguments

studinfo(50,"KV",22.22,"OU","PYTHON") # Function call with Posstional Arguments

print("\*"\*50)

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

4) Variables Length Parameters (or) arguments

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

=>When we have familiy of multiple Similar 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 Implement, Variable length Parameters concept, we must define single Function Definition and takes a formal Parameter preceded with a **symbol called astrisk ( \* param) and the formal parameter with astrisk 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 Posstional formal params, \*param1 , param2=value) :

--------------------------------------------------

--------------------------------------------------

=>Here \*param1 is called Variable Length parameter and it can hold any number of argument values (or) variable number of argument values and \*param1 type is <class,'tuple'>

=>Rule:- The \*param1 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) .

=====================================x=====================================

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

5) Key Word Variables Length Parameters (or) arguments

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

=>When we have familiy of multiple function calls with Key Word 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 Keyword Variable length Parameters .

=>To Implement, Keyword Variable length Parameters concept, we must define single Function Definition and takes a formal Parameter preceded with a symbol called double astrisk ( \*\* param) and the formal parameter with double astrisk symbol is called Keyword Variable length Parameters and whose purpose is to hold / store any number of (Key,Value) 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 Key word argument values (or) Keyword variable number of 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)

---------------------------------------------------------------

Final Syntax for defining a Function

---------------------------------------------------------------

def funcname(PosFormal parms, \*Varlenparams, default params, \*\*kwdvarlenparams):

-------------------------------------------------

---------------------------------------------------

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

#Program for Demonstrating Variable Length Arguments

#VarArgsEx1.py

#This Program will not execute as it is bcoz PVM is an Interpreter It can Consider only Latest Function Definition

def disp(a,b,c,d,e): # Function Def-1 with 5 Params

print(a,b,c,d,e)

def disp(a,b,c,d): # Function Def-2 with 4 Params

print(a,b,c,d)

def disp(a,b,c): # Function Def-3 with 3 Params

print(a,b,c)

def disp(a,b): # Function Def-4 with 2 Params

print(a,b)

def disp(a): # Function Def-5 with 1 Param

print(a)

def disp(): # Function Def-6 with zero Param

print("empty")

#main program

disp(10,20,30,40,50) # Function Call-1 with 5 args

disp(10,20,30,40) # Function Call-2 with 4 args

disp(10,20,30) # Function Call-3 with 3 args

disp(10,20) # Function Call-4 with 2 args

disp(10) # Function Call-5 with 1 args

disp() # Function Call-6 with Zero args"""

#Program for Demonstrating Variable Length Arguments

#VarArgsEx2.py

#This Program will execute as it is.

def disp(a,b,c,d,e): # Function Def-1 with 5 Params

print(a,b,c,d,e)

disp(10,20,30,40,50) # Function Call-1 with 5 args

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

def disp(a,b,c,d): # Function Def-2 with 4 Params

print(a,b,c,d)

disp(10,20,30,40) # Function Call-2 with 4 args

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

def disp(a,b,c): # Function Def-3 with 3 Params

print(a,b,c)

disp(10,20,30) # Function Call-3 with 3 args

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

def disp(a,b): # Function Def-4 with 2 Params

print(a,b)

disp(10,20) # Function Call-4 with 2 args

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

def disp(a): # Function Def-5 with 1 Param

print(a)

disp(10) # Function Call-5 with 1 args

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

def disp(): # Function Def-6 with zero Param

print("empty")

disp() # Function Call-6 with Zero args

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#Program for demonstrating Keyword Variable Length Arguments

#KwdVarLenArgsEx1.py

#This Program will not execute as it is bcoz PVM is an Interpreter It can Consider only Latest Function Definition

def disp(sno,sname,cm,cppm,osm): # Function Def-1 with 5 kwd params

print(sno,sname,cm,cppm,osm)

def disp(eno,ename,sal,comm): # Function Def-2 with 4 kwd params

print(eno,ename,sal,comm)

def disp(id,cname,hobby1,hobby2,hobby3,hobby4):

print(id,cname,hobby1,hobby2,hobby3,hobby4)

def disp(a,b,c):

print(a,b,c)

def disp():

print("empty")

#main program

disp(sno=10,sname="RS",cm=40,cppm=45,osm=45) # Function call-1 with 5 kwd args

disp(eno=20,ename="TR",sal=5000,comm=2000)# Function call-2 with 4 kwd args

disp(id=30,cname="SS",hobby1="Sleeping",hobby2="Eating",hobby3="Chatting",hobby4="watching Inst Reels")

disp(a=10,b=20,c=30)

#Program for demonstrating Keyword Variable Length Arguments

#KwdVarLenArgsEx2.py

#This Program will execute as it is

def disp(sno,sname,cm,cppm,osm): # Function Def-1 with 5 kwd params

print(sno,sname,cm,cppm,osm)

disp(sno=10,sname="RS",cm=40,cppm=45,osm=45) # Function call-1 with 5 kwd args

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

def disp(eno,ename,sal,comm): # Function Def-2 with 4 kwd params

print(eno,ename,sal,comm)

disp(eno=20,ename="TR",sal=5000,comm=2000)# Function call-2 with 4 kwd args

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

def disp(id,cname,hobby1,hobby2,hobby3,hobby4):

print(id,cname,hobby1,hobby2,hobby3,hobby4)

disp(id=30,cname="SS",hobby1="Sleeping",hobby2="Eating",hobby3="Chatting",hobby4="watching Inst Reels")

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

def disp(a,b,c):

print(a,b,c)

disp(a=10,b=20,c=30)

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

def disp():

print("empty")

disp()

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#Program for demonstrating Keyword Variable Length Arguments

#PureKwdVarLenArgsEx1.py

def disp(\*\*a): # here \*\*a is called Keyword Var length Param and its type is dict and it stores (Key,value) of Kwg var length values

print(a,type(a),len(a))

#main program

disp(sno=10,sname="RS",cm=40,cppm=45,osm=45) # Function call-1 with 5 kwd args

disp(eno=20,ename="TR",sal=5000,comm=2000)# Function call-2 with 4 kwd args

disp(id=30,cname="SS",hobby1="Sleeping",hobby2="Eating",hobby3="Chatting",hobby4="watching Inst Reels")

disp(a=10,b=20,c=30)

disp()

#Program for demonstrating Keyword Variable Length Arguments

#PureKwdVarLenArgsEx2.py

def disp(\*\*a): # here \*\*a is called Keyword Var length Param and its type is dict and it stores (Key,value) of Kwg var length values

print("--------------------------------------")

for k,v in a.items():

print("\t{}--->{}".format(k,v))

print("--------------------------------------")

#main program

disp(sno=10,sname="RS",cm=40,cppm=45,osm=45) # Function call-1 with 5 kwd args

disp(eno=20,ename="TR",sal=5000,comm=2000)# Function call-2 with 4 kwd args

disp(id=30,cname="SS",hobby1="Sleeping",hobby2="Eating",hobby3="Chatting",hobby4="watching Inst Reels")

disp(a=10,b=20,c=30)

disp()

#Program for demonstrating Keyword Variable Length Arguments

#PureKwdVarLenArgsEx3.py

def disp(inf,\*\*a): # here \*\*a is called Keyword Var length Param and its type is dict and it stores (Key,value) of Kwg var length values

print("--------------------------------------")

print("'{}' Information".format(inf))

print("----------------------------------------")

for k,v in a.items():

print("\t{}--->{}".format(k,v))

print("--------------------------------------")

#main program

disp("Student",sno=10,sname="RS",cm=40,cppm=45,osm=45) # Function call-1 with 5 kwd args

disp("Employee",eno=20,ename="TR",sal=5000,comm=2000)# Function call-2 with 4 kwd args

disp("Lazy Student",id=30,cname="SS",hobby1="Sleeping",hobby2="Eating",hobby3="Chatting",hobby4="watching Inst Reels")

disp("Numbers",a=10,b=20,c=30)

disp("Empty")

#Program for demonstrating Keyword Variable Length Arguments

#PureKwdVarLenArgsEx4.py

def findstudtotmarks(sno,sname,cls,\*\*submarks):

print("\*"\*40)

print("\tS t u d e n t I n f o r m a t i o n")

print("\*"\*40)

print("\tStudent Number:{}".format(sno))

print("\tStudent Name:{}".format(sname))

print("\tStudent Class:{}".format(cls))

print("-"\*40)

tot=0

for subject,marks in submarks.items():

print("\t{}\t{}".format(subject,marks))

tot=tot+marks

print("-"\*40)

print("TOTAL MARKS={}".format(tot))

print("\*"\*40)

#main program

findstudtotmarks(10,"Rajesh","X",Telugu=45,Hindi=56,English=45,Maths=67,Science=78,Social=78)

findstudtotmarks(20,"Abrar","XII",Sanskrit=95,English=55,Maths=75,Physics=60,Chemistry=60)

findstudtotmarks(30,"Praveen","B.Tech",OS=30,DBMS=35,GP=35)

findstudtotmarks(40,"Rossum","MS")

#Program for demonstrating Keyword Variable Length Arguments

#PureKwdVarLenArgsEx5.py

def findstudtotmarks(sno,sname,cls,city="HYD",\*\*submarks):

print("\*"\*40)

print("\tS t u d e n t I n f o r m a t i o n")

print("\*"\*40)

print("\tStudent Number:{}".format(sno))

print("\tStudent Name:{}".format(sname))

print("\tStudent Class:{}".format(cls))

print("\tStudent Living City:{}".format(city))

print("-"\*40)

tot=0

for subject,marks in submarks.items():

print("\t{}\t{}".format(subject,marks))

tot=tot+marks

print("-"\*40)

print("TOTAL MARKS={}".format(tot))

print("\*"\*40)

#main program

findstudtotmarks(10,"Rajesh","X",Telugu=45,Hindi=56,English=45,Maths=67,Science=78,Social=78)

findstudtotmarks(20,"Abrar","XII",Sanskrit=95,English=55,Maths=75,Physics=60,Chemistry=60)

findstudtotmarks(30,"Praveen","B.Tech",OS=30,DBMS=35,GP=35)

findstudtotmarks(40,"Rossum","MS","NL")

findstudtotmarks(50,"TRAVIS","Research",RM=100,city="AUS"

#Program for demonstrating Keyword Variable Length Arguments

#PureKwdVarLenArgsEx6.py

def findstudtotmarks(sno,sname,cls,\*vals,city="HYD",\*\*submarks):

print("\*"\*40)

print("\tS t u d e n t I n f o r m a t i o n")

if(len(vals)==0):

print("'{}' did't Passed the Variable Length Values".format(sname))

else:

print("'{}' Passed the following Variable Length Values".format(sname))

for val in vals:

print("\t{}".format(val))

print("\*"\*40)

print("\tStudent Number:{}".format(sno))

print("\tStudent Name:{}".format(sname))

print("\tStudent Class:{}".format(cls))

print("\tStudent Living City:{}".format(city))

print("-"\*40)

if(len(submarks)==0):

print("'{}' did't Passed the Keyword--Variable Length Values".format(sname))

else:

tot=0

for subject,marks in submarks.items():

print("\t{}\t{}".format(subject,marks))

tot=tot+marks

print("-"\*40)

print("TOTAL MARKS={}".format(tot))

print("\*"\*40)

#main program

findstudtotmarks(10,"Rajesh","X",100,200,300,400,500,Telugu=45,Hindi=56,English=45,Maths=67,Science=78,Social=78)

findstudtotmarks(20,"Abrar","XII",10,20,30,40,Sanskrit=95,English=55,Maths=75,Physics=60,Chemistry=60)

findstudtotmarks(30,"Praveen","B.Tech",1.2,2.3,3.4,OS=30,DBMS=35,GP=35)

findstudtotmarks(40,"Rossum","MS",12,13,14,city="NL")

findstudtotmarks(50,"TRAVIS","Research",RM=100,city="AUS")

findstudtotmarks(60,"KVR","Faculty")

#Program for Demonstrating Variable Length Arguments

#PureVarArgsEx1.py

#This Program will execute as it

def disp( \*a ): # Here \*a is called Variable Length Parameter whose type is tuple and it can store Variable Length vals

print(a, type(a), len(a))

#main program

disp(10,20,30,40,50) # Function Call-1 with 5 args

disp(10,20,30,40) # Function Call-2 with 4 args

disp(10,20,30) # Function Call-3 with 3 args

disp(10,20) # Function Call-4 with 2 args

disp(10) # Function Call-5 with 1 args

disp() # Function Call-6 with Zero args"""

#NOTE: Same Function Call with Variable Length Arguments

#Program for Demonstrating Variable Length Arguments

#PureVarArgsEx2.py

#This Program will execute as it

def disp( \*a ): # Here \*a is called Variable Length Parameter whose type is tuple and it can store Variable Length vals

for val in a:

print("\t{}".format(val))

print("---------------------------------------")

#main program

disp(10,20,30,40,50) # Function Call-1 with 5 args

disp(10,20,30,40) # Function Call-2 with 4 args

disp(10,20,30) # Function Call-3 with 3 args

disp(10,20) # Function Call-4 with 2 args

disp(10) # Function Call-5 with 1 args

disp() # Function Call-6 with Zero args"""

#NOTE: Same Function Call with Variable Length Arguments

#**Program for Demonstrating Variable Length Arguments**

#PureVarArgsEx3.py

#This Program will execute as it

def disp(sid,sname, \*a ): # Here \*a is called Variable Length Parameter whose type is tuple and it can store Variable Length vals

print("Student ID:{}".format(sid))

print("Student Name:{}".format(sname))

s=0

for val in a:

print("\t{}".format(val))

s=s+val

print("---------------------------------------")

print("Sum={}".format(s))

print("---------------------------------------")

#main program

disp("R100","Rossum",10,20,30,40,50) # Function Call-1 with 5 args

disp("T100","TRAVIS",10,20,30,40) # Function Call-2 with 4 args

disp("D100","Dennis",10,20,30) # Function Call-3 with 3 args

disp("K100","KINNEY",10,20) # Function Call-4 with 2 args

disp("S100","STRAUP",10) # Function Call-5 with 1 args

disp("H100","HUNGUNIAN") # Function Call-6 with Zero args"""

#NOTE: Same Function Call with Variable Length Arguments

**#Program for Demonstrating Variable Length Arguments**

#PureVarArgsEx4.py

#This Program will execute as it

def disp(sid,sname, \*a,city="HYD"): # Here \*a is called Variable Length Parameter whose type is tuple and it can store Variable Length vals

print("Student ID:{}".format(sid))

print("Student Name:{}".format(sname))

print("Student Living City:{}".format(city))

s=0

for val in a:

print("\t{}".format(val))

s=s+val

print("---------------------------------------")

print("Sum={}".format(s))

print("---------------------------------------")

#main program

disp("R100","Rossum",10,20,30,40,50) # Function Call-1 with 5 args

disp("T100","TRAVIS",10,20,30,40) # Function Call-2 with 4 args

disp("D100","Dennis",10,20,30) # Function Call-3 with 3 args

disp("K100","KINNEY",10,20) # Function Call-4 with 2 args

disp("S100","STRAUP",10) # Function Call-5 with 1 args

disp("H100","HUNGUNIAN") # Function Call-6 with Zero args

disp("W100","WES",1.2,3.4,4.5,city="USA") # Function Call-7 with 3 args

#disp("Z100","ZEN",city="AUS",100,200,1.2,3.4) SyntaxError: positional argument follows keyword argument

disp("Z100","ZEN",100,200,1.2,3.4,city="AUS")

#disp(0.2,0.3,9.4,city="BANG",sid="S400",sname="RAM") --TypeError: disp() got multiple values for argument 'sid'

#NOTE: Same Function Call with Variable Length Arguments

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

Local variables and Global Variables

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

Local Variables

-------------------

=>The Variables used inside of Function Body are called Local Variables.

=>The Purpose of Local Variables is that "To Store the Temporary results".

=>Local Variables Can be accessed inside of Corresponding Function Body But Not possible to access in other Part of the program.

------------

Syntax:

-----------

def functionname(list of formal Params if any):

-------------------------------

var1=val1

var2=Val2

------------

var-n=val-n

---------------

=>here var1,var2..var-n are called Local Variables bcoz they are used inside of Function Body .

---------------------------

Global Variables

---------------------------

=>Global variables are those which are common values for different function calls.

=>In Other words, if the Value is common for all Different Function Calls then such type

of values must be taken as Global Variables.

=>To access the values of Global Variables then They Must be defined Before Function

Calls only otherwise we get **NameError**.

Syntax: var1=val1

var2=val2

-------------

var-n=val-n

def fun1():

-----------

-----------

def fun2():

-----------

-----------

def fun-n():

-----------

-----------

(OR)

Syntax: def fun1():

-----------

-----------

def fun2():

-----------

-----------

def fun-n():

-----------

-----------

#main program

var1=val1

var2=val2

-------------

var-n=val-n

fun1() # Function Call

fun2() # Function Call

fun3() # Function Call

Here Var1, Var2..var-n are called Global variables and we can access those values inside of fun1(), fun2()....fun-n().

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

global key word

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

=>When we want MODIFY the GLOBAL VARIABLE values in side of function definition 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

---------------------------------

**NOTE: To MODIFY Global variable Values inside of Function Definition, we use global Keyword**

**NOTE: To ACCESS Global variable Values inside of Function Definition, we Don't use global Keyword**

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

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

global and local variables and globals()

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

**=>When we come across same global Variable names and Local Variable Names in same function definition then PVM gives preference for local variables but not for global variables.**

**=>In this context, to extract / retrieve the values of global variables names along with local variables, we must use globals() and it returns an object of <class,'dict'>** and this dict object stores all global variable Names as Keys and global variable values as values of value.

=>Syntax:-

var1=val1

var2=val2

--------------

var-n=val-n # var1, var2...var-n are called global Variables

def functionname():

------------------------

var1=val11

var2=val22

-----------------

var-n=val-nn # var1, var2...var-n are called local Variables

# Extarct the global variables values

dictobj=globals()

------------------------

globalval1=dictobj['var1'] # or dictobj.get("var1") or globals()['var1'] or

global().get('var1')

globalval2=dictobj['var2'] # or dictobj.get("var2") or globals()['var2']

-----------------------------------------------------

-----------------------------------------------------

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

Examples:

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

#globalsfunex3.py

a=10

b=20

c=30

d=40

def operations():

obj=globals()

for gvn,gvv in obj.items():

print("\t{}---->{}".format(gvn,gvv))

print("="\*50)

print("\nProgrammer-defined Global Variables")

print("="\*50)

print("Val of a=", obj['a'])

print("Val of b=", obj['b'])

print("Val of c=", obj['c'])

print("Val of d=", obj['d'])

print("="\*50)

print("\nProgrammer-defined Global Variables")

print("="\*50)

print("Val of a=", obj.get('a'))

print("Val of b=", obj.get('b'))

print("Val of c=", obj.get('c'))

print("Val of d=", obj.get('d'))

print("="\*50)

print("\nProgrammer-defined Global Variables")

print("="\*50)

print("Val of a=", globals().get('a'))

print("Val of b=", globals().get('b'))

print("Val of c=", globals().get('c'))

print("Val of d=", globals().get('d'))

print("="\*50)

print("\nProgrammer-defined Global Variables")

print("="\*50)

print("Val of a=", globals()['a'])

print("Val of b=", globals()['b'])

print("Val of c=", globals()['c'])

print("Val of d=", globals()['d'])

print("="\*50)

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

#main program

operations()

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

Examples:

-----------------------

#Program for demonstrating globals()

#globalsfunex2.py

a=10

b=20

c=30

d=40 # Here a,b,c,d are called Global Variables

def operation():

a=100

b=200

c=300

d=400 # Here a,b,c,d are called Local Variables

res=a+b+c+d+globals()['a']+globals().get('b')+globals()['c']+globals()['d']

print(res)

#main program

operation()

*#program for Demonstrating Global Keyword  
#GlobalKeywordEx1.py*def modify1():  
 global a  
 a=a+1  
def modify2():  
 global a  
 a=a\*2  
def modify3():  
 c=a+2 *# Here No need to write global keyword bcoz we are not modifying global var val but we are just accessing global variable value 'a'* print("Val of c in Modify3()=",c) *# 24  
  
#Main program*a=10 *#Here a is global Varaible*print("In Main Program-Global Var a before modify1()={}".format(a)) *# 10*modify1() *# Function Call*print("In Main Program-Global Var a after modify1()={}".format(a)) *# 11*modify2()  
print("In Main Program-Global Var a after modify2()={}".format(a)) *# 22*modify3()  
print("In Main Program-Global Var a after modify2()={}".format(a)) *# 22*

*#program for Demonstrating Global Keyword  
#GlobalKeywordEx2.py*def modify1():  
 global a,b  
 a=a+1 *# Here we are accessing Global Var Values and Modyfing* b=b+1  
def modify2():  
 global a,b  
 a=a\*2 *# Here we are accessing Global Var Values and Modyfing* b=b\*2  
  
*#Main program*a=10  
b=20 *#Here a and b are global Variables*print("In Main Program-Global Var a={} b={} before modify1()".format(a,b)) *# a=10 b=20*modify1() *# Function Call*print("In Main Program-Global Var a={} b={} after modify1()".format(a,b)) *# a=11 b=21*modify2() *# Function Call*print("In Main Program-Global Var a={} b={} after modify2()".format(a,b)) *# a=22 b=42*

***#Program demonstrating globals()*** *#GlobalsFunEx3.py  
#In This Program we have Same Global and Local Variable Have*a=10  
b=20  
c=30  
d=40 *# Here a,b,c, and d are called Global Variables*def operation():  
 d1=globals()  
 a=100  
 b=200  
 c=300  
 d=400 *# Here a,b,c, and d are called Local Variables* res=a+b+c+d+globals().get('a')+d1.get('b')+d1['c']+globals()['d']  
 print("Sum=",res)  
  
*#main program*

operation()

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

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

Anonymous OR Lambda Functions or Name Less functions

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

=>Anonymous Function are those which does not contain any Name Explicitly( Function which does not contain any name is called Anonymous Function)

=>The purpose of Anonymous Functions is that "To Perform Instant Operations".

=>Instant Operations are nothing but the operations which are performing at that Point of Time only and no longer interested to Re-Use in further programs OR applications

=>Anonymous Functions contains Single Executable Statement.

**=>Anonymous Functions returns the result of Single Executable Statement automatically OR Implicitly without using return statement.**

=>To Develop Anonymous Functions, we use a keyword called **lambda** and Anonymous Functions are called Lambda Functions

-------------------------------------------------------------------------------------------------------------------------------

**Syntax: varname = lambda params-list : Expression**

-------------------------------------------------------------------------------------------------------------------------------

Explanation

-------------------

=>varname---->

=>lambda------>Represents Keyword used for defining Anonymous Function

=>params-list--->Represents list for Variables used for Storing the inputs coming from Functions calls

=>Expression--->Represents Single Executable Statement and whose Result returned automatically OR Implicitly and No need to use return statement.

=========================================================================== Q) Define a Function for adding Two Numbers

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

By using Normal Function By using

Anonymous Function

--------------------------------------------------- -----------------------------------

def sumop(a,b): # Fun Def. sumop = lambda a,b : a+b

c=a+b

return c

#Main Program

res=sumop(10,2 0) # Function call

***#AnonymousFunEx1.py***def sumop(a,b): *# Normal Fun Def.* c=a+b  
 return c  
  
addop = lambda a,b : a+b *# Anonymous Function  
# here addop is of type <class 'function'>  
  
#Main Program*print("Type of sumop=",type(sumop))  
res=sumop(10,20) *# Function call*print("Sum=",res)  
print("--------------------------------------")  
print("type of addop=",type(addop))  
res1=addop(100,200) *# Function call*print("sum=",res1)

***#AnonymousFunEx1.py***sumop=lambda x,y: x+y *# Anonymous Function  
  
#main program*a,b=float(input("Enter First Value:")), float(input("Enter Second Value:"))  
res=sumop(a,b) *# Function Call*print("sum({},{})={}".format(a,b,res))  
print("==========OR==============")  
print("sum({},{})={}".format(a,b,sumop(a,b)))

*#AnonymousFunEx3.py*findbig=lambda k,v,r: k if k>=v and k>r else v if v>k and v>=r else r if r>=k and r>v else " all values are equal"  
  
*#Main program*a,b,c=float(input("Enter First Value:")), float(input("Enter Second Value:")),float(input("Enter Third Value:"))  
bv=findbig(a,b,c) *# Function Call*print("Max({},{},{})={}".format(a,b,c,bv))

*#AnonymousFunEx4.py*def readvalues():  
 n=int(input("Enter How Many Value u want in list:"))  
 if(n<=0):  
 return []  
 else:  
 lst=[]  
 for i in range(1,n+1):  
 value=float(input("Enter {} Value:".format(i)))  
 lst.append(value)  
 return lst  
  
maxv=lambda itrobj : "List is empty, can't Find Max" if len(itrobj)==0 else max(itrobj) *# Anonymous Function*minv=lambda itrobj : "List is empty, can't Find Min" if len(itrobj)==0 else min(itrobj) *# Anonymous Function  
#main program*lstobj=readvalues()  
mv=maxv(lstobj) *#print(lstobj) # [10.0, 2.0, 45.0, -34.0, 0.0]*miv=minv(lstobj) *#print(lstobj) # [10.0, 2.0, 45.0, -34.0, 0.0]*print("Max({})={}".format(lstobj,mv))  
print("Min({})={}".format(lstobj,miv))

***#AnonymousFunEx5.py***def readvalues():  
 n=int(input("Enter How Many Value u want in list:"))  
 if(n<=0):  
 return []  
 else:  
 lst=[]  
 for i in range(1,n+1):  
 value=float(input("Enter {} Value:".format(i)))  
 lst.append(value)  
 return lst  
  
def findmax(lstobj): *# lstobj=[100,20,3,25,4,45,6]* maxval=lstobj[0]  
 for value in lstobj:  
 if(value>maxval):  
 maxval=value  
 return maxval  
  
def findmin(lstobj):  
 minval = lstobj[0]  
 for value in lstobj:  
 if (value < minval):  
 minval = value  
 return minval  
maxv=lambda itrobj : "List is empty, can't Find Max" if len(itrobj)==0 else "All Values Are Equal" if(len(set(lstobj))==1) else findmax(itrobj) *# Anonymous Function*minv=lambda itrobj : "List is empty, can't Find Min" if len(itrobj)==0 else "All Values Are Equal" if(len(set(lstobj))==1) else findmin(itrobj) *# Anonymous Function  
#main program*lstobj=readvalues()  
mv=maxv(lstobj) *#print(lstobj) # [10.0, 2.0, 45.0, -34.0, 0.0]*miv=minv(lstobj) *#print(lstobj) # [10.0, 2.0, 45.0, -34.0, 0.0]*print("Max({})={}".format(lstobj,mv))  
print("Min({})={}".format(lstobj,miv))

***#AnonymousFunEx6.py***splitline=lambda line: line.split()  
touppercase=lambda word: word.upper()  
toreverse=lambda word:word[::-1].upper()  
*#Main program*line=input("Enter a Line of Text:")  
words=splitline(line)  
ucwords=[]  
rwords=[]  
for word in words:  
 uppercasewords=touppercase(word)  
 reversewords=toreverse(word)  
 ucwords.append(uppercasewords)  
 rwords.append(reversewords)  
else:  
 print("Given Words: {}".format(words))  
 print("Upper Case Words={}".format(ucwords))  
 print("Reverse Words={}".format(rwords))

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

**comprehension**

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

=>Python is famous for allowing you to write code that’s elegant, easy to write, and almost as easy to read as plain English. One of the language’s most distinctive features is the list comprehension, which you can use to create powerful functionality within a single line of code rather than writing legacy lines of Code.

=>The purpose of List comprehension is that to read the values dynamically from key board separated by a delimeter ( space, comma, colon..etc)

=>List comprehension is the most effective way for reading the data for list instead tradtional reading the data.

**=>Syntax:- listobj=[ expression for varname in Iterable\_object if test Cond ]**

=>here expression represents either type casting or mathematical expression

Examples:

----------------------

print("Enter List of values separated by space:") # [10 2 22 50 10 4 55 -3 0 22]

lst= [float(val) for val in input().split() ]

print("content of lst",lst)

------------------

Examples:

------------------

lst=[4,3,7,-2,6,3]

newlst=[ val\*2 for val in lst ]

print("new list=",newlst) # [ 8, 6, 14,-4,12,6 ]

*#reading the values from KBD by using List Comprehension  
#ListComprehensionEx1.py*print("Enter List of Values Separated Space:")  
lst= [ int(num) for num in input().split() ]  
print("List of Values:{} and whose type is {}".format(lst,type(lst)))

*#Program for Rewading List of Values an Obtain their Squares  
#ListComprehensionEx2.py*print("Enter List of Values separated comma:")  
lst=[int(val) for val in input().split(",") ]  
print("Given List of Elements={}".format(lst))  
*#Obtain sqaure*sqlist=[val\*\*2 for val in lst]  
print("Square List={}".format(sqlist))

*#Program for Reading Only +Ve values from Keyboard  
#ListComprehensionEx3.py*print("Enter List of Values separated by space:")  
lst1=[ float(value) for value in input().split() if float(value)>0 ]  
  
print("Enter List of Values separated by space:")  
lst2=[ float(value) for value in input().split() if float(value)<0 ]  
  
print("List of +Ve Values:{}".format(lst1))  
print("List of -Ve Values:{}".format(lst2))

*#ListComprehensionEx4.py*print("Enter List of Words Separated space:")  
d=dict([ ( word,len(word) ) for word in input().split() ])  
for w in d:  
 print("\t{}--->{}".format(w,d.get(w)))

*#Program for Reading Only +Ve values from Keyboard  
#SetComprehensionEx1.py*print("Enter List of Values separated by space:")  
set1={ float(value) for value in input().split() if float(value)>0 }  
  
print("Enter List of Values separated by space:")  
set2={ float(value) for value in input().split() if float(value)<0 }  
  
print("List of +Ve Values:{}".format(set1))  
print("List of -Ve Values:{}".format(set2))

*#SetComprehensionEx2.py  
#reading the values from KBD by using List Comprehension*print("Enter List of Values Separated Space:")  
set1= { int(num) for num in input().split()}  
print("List of Values:{} and whose type is {}".format(set1,type(set1)))

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

1. filter()

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

=>filter() is used for "Filtering out some elements from list of elements by applying to function".

**=>Syntax:- varname=filter(FunctionName, Iterable\_object)**

---------------------

Explanation:

---------------------

=>here 'varname' is an object of type **<class,'filter'>** and we can convert into any iteratable object by using type casting functions.

=>"Function Name" represents either Normal function or anonymous functions.

=>"Iterable\_object" represents Sequence, List, set and dict types.

=>The execution process of filter() is that " Each Value of Iterable object sends to Function Name. If the function return True then the element will be filtered. if the Function returns False then that element will be neglected/not filtered ". This process will be continued until all elements of Iterable object completed.

-------------------------------------------------------------------------------------------------------------------------------

*#Obtain list of Even Numbers by using Filter()  
#FilterEx1.py*def even(n):  
 if(n%2==0):  
 return True  
 else:  
 return False  
  
*#main program*lst=[10,20,25,37,38,23,39,13,14]  
obj=filter(even,lst) *# here obj is an object of <class, 'filter'>  
#here filter() returns an object <class,'filter'>*elist=tuple(obj) *# Type case Filter object into list type*print("========================")  
print("Given Elements=",lst)  
print("Even Elements=",elist)  
print("========================")

*##Obtain list of Even Numbers by using Filter()  
#FilterEx2.py*def even(n):  
 return( True if n%2==0 else False)  
def odd(n):  
 return (True if n%2!=0 else False)  
  
*#Main program*print("Enter List of Values separated by space")  
lst=[ int(val) for val in input().split()]  
elist=list(filter(even,lst))  
olist=tuple(filter(odd,lst))  
print("==================================")  
print("Given Elements=",lst)  
print("Even Elements=",elist)  
print("Odd Elements=",olist)  
print("===================================")

*##Obtain list of Even Numbers by using Filter()  
#FilterEx3.py*print("Enter List of Values separated by space")  
lst=[int(val) for val in input().split()] *# lst=11 13 12 15 14 67 69 78 24 23 101 100 103*elist=list(filter(lambda n: n%2==0 , lst ) ) *# filter() with anonymous Function*olist=tuple(filter(lambda n: n%2!=0,lst)) *# filter() with anonymous Function*print("==================================")  
print("Given Elements=",lst)  
print("Even Elements=",elist)  
print("Odd Elements=",olist)  
print("===================================")

*#program for Filtering list of primes from list of values by using filter()  
#FilterEx4.py*def prime(n):  
 if(n<=1):  
 return False  
 res=True  
 for i in range(2,n):  
 if(n%i==0):  
 res=False  
 break  
 return res  
*#main program*print("Enter List of Values separated by space")  
lst=[int(val) for val in input().split()]  
primelist=list(filter(lambda n: prime(n),lst))  
print("==================================")  
print("Given Elements=",lst)  
print("List of Prime Elements=",primelist)  
print("===================================")

*#Program for accepting line of text and get alphabets , digits and special symbols  
#FilterEx5.py*line=input("Enter Line of Text:") *# Py$5th6#on*alphalist=list(filter(lambda ch: ch.isalpha() , line))  
digitslist=list(filter(lambda ch: ch.isdigit(),line))  
symbolslist=list(filter(lambda ch:not ch.isalnum() or ch.isspace(),line))  
print("\*"\*60)  
print("Given Line=",line)  
print("Alphabets={} No.Alpabets={}".format(alphalist,len(alphalist)))  
print("Digits={} No.Digits={}".format(digitslist,len(digitslist)))  
print("Spcial Symbols={} No.Spcial Symbols={}".format(symbolslist,len(symbolslist)))  
print("\*"\*60)

====================================  
 2) map()  
 ====================================  
=>map() is used for obtaining new Iterable object from existing iterable object by applying old iterable elements to the function.  
=>In otherwords, map() is used for obtaining new list of elements from existing list of elements by applying old list elements to the function.  
  
**=>Syntax:- varname=map(FunctionName,Iterable\_object)**   
=>here 'varname' is an object of type **<class,map'>** and we can convert into any iteratable object by using type casting functions.  
=>"FunctionName" represents either Normal function or anonymous functions.  
=>"Iterable\_object" represents Sequence, List, set and dict types.  
=>The execution process of map() is that " map() sends every element of iterable object to the specified function, process it and returns the modified value (result) and new list of elements will be obtained". This process will be continued until all elements of Iterable\_object completed.  
-----------------------------------------------------------------------------------

*#MapEx1.py*def squares(n):  
 return n\*\*2  
*#main program*print("Enter List of Values separated by space:")  
lst1=[int(val) for val in input().split()]  
obj1=map(squares,lst1) *# here obj is an object of <class,'map'>  
#convert map object into list*sqlist=list(obj1)  
print("Given List of Elements=",lst1)  
print("Square List=",sqlist)

*#MapEx2.py*print("Enter List of Values separated by space:")  
lst1=[int(val) for val in input().split()]  
sqlist=list(map(lambda n:n\*\*2 , lst1))  
*#convert map object into list*print("Given List of Elements=",lst1)  
print("Square List=",sqlist)

*#MapEx3.py*print("Enter List of Salaries of Employee separated by space:")  
oldsals=[ int(sal) for sal in input().split() if (int(sal)>0)]  
print("Old Salaries List=",oldsals)  
newsals=list(map(lambda sal:sal+sal\*50/100,oldsals))  
print("="\*50)  
print("\t\tOld Salary\t\tNew Salary")  
print("="\*50)  
for osal,nsal in zip(oldsals,newsals):  
 print("\t\t{}\t\t\t\t{}".format(osal,nsal))  
print("="\*50)

*#MapEx4.py--sum of two list content*lst1=[10,20,30,40]  
lst2=[1,2,3,4]  
lst3=list(map(lambda x,y: x+y, lst1,lst2))  
print("First List Elements=",lst1)  
print("Second List Elements=",lst2)  
print("Sum List=",lst3)

*#MapEx5.py*print("Enter List of Elements for First List separated by space:")  
lst1=[int(val) for val in input().split()]  
print("Enter List of Elements for Second List separated by space:")  
lst2=[int(val) for val in input().split()]  
if (len(lst1)>len(lst2)):  
 for i in range( len(lst1)-len(lst2) ):  
 lst2.append(0)  
elif(len(lst2)>len(lst1)):  
 for i in range(len(lst2)-len(lst1)):  
 lst1.append(0)  
lst3=list(map(lambda x,y:x+y,lst1,lst2))  
print("-------------------------------------------")  
print("\t\tList1\t\tList2\t\tSumList")  
print("-------------------------------------------")  
for i,j,k in zip(lst1,lst2,lst3):  
 print("\t\t{}\t\t\t{}\t\t\t{}".format(i,j,k))  
print("-------------------------------------------")

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

3. reduce()

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

=>reduce() is used for obtaining a single element / result from given iterable object by applying to a function.

=>**Syntax:-varname=reduce(function-name,iterable-object)**

=>**here varname is an object of int, float,bool,complex,str only**

=>The reduce() belongs to a pre-defined module called" functools".

---------------------------------------

Internal Flow of reduce()

---------------------------------------

step-1:- Initially, reduce() selects First Two values of Iterable object and place them in First var and Second var .

step-2:- The function-name(lambda or normal function) utilizes the values of First var and

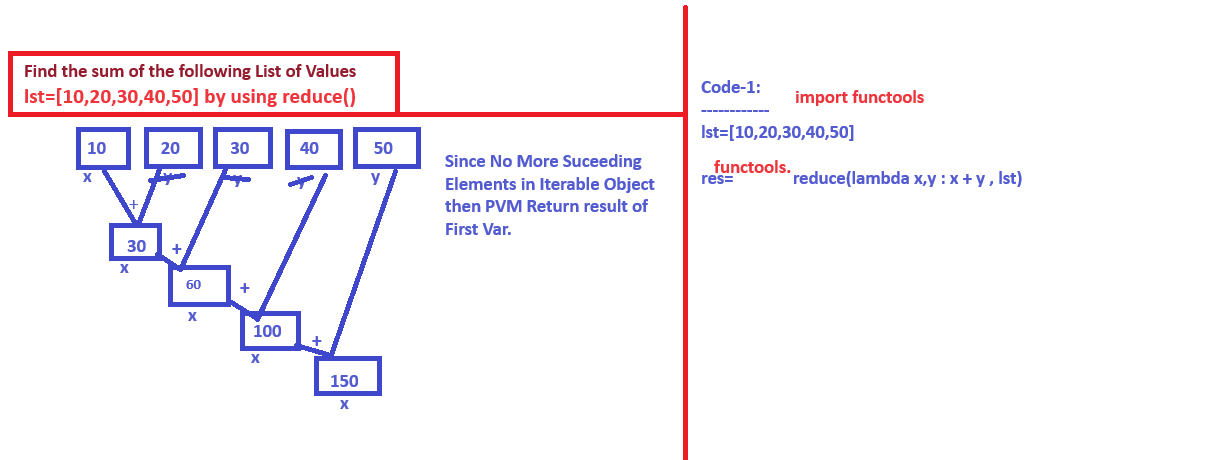
Second var and applied to the specified logic and obtains the result.

Step-3:- reduce () places the result of function-name in First variable and reduce()

selects the succeeding element of Iterable object and places in second variable.

Step-4: Repeat Step-2 and Step-3 until all elements completed in

Iterable object and returns the result of First Variable.



*#ReduceEx1.py*import functools  
print("Enter List of Values separated by space:")  
lst=[int(val) for val in input().split()]  
res=functools.reduce(lambda x,y: x+y, lst)  
print("sum{})={}".format(lst,res))

*#ReduceEx2.py*import functools  
def sumop(k,v):  
 return(k+v)  
*#main program*print("Enter List of Values separated by space:")  
lst=[int(val) for val in input().split()]  
res=functools.reduce(sumop, lst)  
print("sum{})={}".format(lst,res))

*#ReduceEx3.py*import functools  
def maxop(k,v):  
 return( k if k>v else v)  
*#main program*print("Enter List of Values separated by space:") *# [10,20,15,25,14]*lst=[int(val) for val in input().split()]  
res=functools.reduce(maxop, lst)  
print("Max{})={}".format(lst,res))

*#ReduceEx4.py*import functools  
print("Enter List of Values separated by space:") *# [10,20,15,25,14]*lst=[int(val) for val in input( ).split( )]  
res=functools.reduce(lambda k,v: k if k>v else v, lst)  
print("Max{})={}".format(lst,res))

*#ReduceEx5.py*import functools  
print("Enter List of Words separated by Comma:")  
lst=[str(val) for val in input().split(",")]  
res=functools.reduce(lambda k,v: k+" "+v, lst)  
print("Line({})={}".format(lst,res))

=============================================================  
 Modules  
 =============================================================  
Index  
----------------------------------------------------------------------------------------------------------------------------  
=>Purpose of Modules  
=>Definition of Module  
=>Types of Modules  
 a) Pre-Defined Modules  
 b) Programmer OR User Or Custom Defined Modules  
=>Steps for Developing Programmer OR User Or Custom Defined Modules  
=>Programming Examples  
=>Number of Approaches to Re-Use Modules  
 a) By using import statement  
 b) By using from...import statement  
=>Programming Examples  
=>Re-Loading Modules  
=>Implementation of imp or importlib modules  
=>Programming Examples  
=======================================================================

====================================================  
 Modules in Python  
 ====================================================  
=>We know that Functions are used for "Performing Certain Operations and Provides Code Re-Usability within the same Program but not able to provide Code Re-Usability across the programs.".  
**=>The Purpose of Modules Concept is that "To Re-use the functions, global variables and Class Names" from One Program to another Program provided Both The Programs present in Same Folder.**  
------------------------------------------  
**=>Definition of Module**------------------------------------------  
**A Module is a collection of Functions, Global Variable Names and Class Names.**------------------------------------------------------  
Types of Modules  
-------------------------------------------------------  
=>In Python Programming, we have Two Types of Modules. They are  
 1. Pre-Defined Modules  
 2. Programmer OR User OR Custom Defined Module  
-------------------------------------------------------------  
1. Pre-Defined Modules  
-------------------------------------------------------------  
=>These Modules are already defined by Python Lang Developers and Available in Python Software and Used by all Python Lang Programmers and for dealing with Universal Requirements.  
**Examples: functools,sys,math,calendar,re,pickle,threading,csv..etc**=>**Out-of Many Pre-defined Modules, By default One of the pre-defined module called "builtins" imported to all python programs and It is called Default imported python Module.**--------------------------------------------------------------------------------  
2. Programmer OR User OR Custom Defined Module  
-----------------------------------------------------------------------------------=>These Modules are developed by Python Programmers and avilable in Python Project and Used by Other Members of Same Project for dealing with Common Requirements.  
Examples: Aop,MathsInfo,icici...etc  
---------------------------------------------------------------------------------- =================================================  
 Development of Programmer-Defined Module  
 =================================================  
**=>To develop Programmer-Defined Modules, we must use the following steps  
  
 Step-1 : Define Variables (Global variables)  
 Step-2: Define Functions   
 Step-3: Define Classes**  
  
=>After developing step-1, step-2 and step-3 , we must save on some file name with an extension .py (FileName.py) and it is treated as module name.   
**=>Hence Every Python File is treated as Module Name.**  
**=>When a file name treated as a module name , internally Python execution environment creates a folder automatically on the name of \_\_pycache\_\_ and it contains module name on the name of "filename.cpython-312.pyc ".**  
-------------------  
Examples:  
-------------------  
 \_\_pycache\_\_ <-----Folder Name   
 -------------------------------------------  
 Aop.cpython-312.pyc <-------------------Module Name  
 mathsinfo.cpython-312.pyc<--------------Module Name  
 icici.cpython-312.pyc<----------------------Module Name  
 ----------------------------------------------

===============================================  
 Number of approaches to re-use Modules  
 ===============================================  
=>We know that A Module is a collection of variables, Functions and Classes.  
=>To re-use the features(Variable Names, Function Names and Class Names ) of module, we have 2 approaches.They are  
 1) By using import statement  
 2) By using from.... import statement.  
-----------------------------------------------------------------------------------  
**1) By using import statement:**-----------------------------------------------------------------------------------  
=>'import' is a keyword  
=>The purpose of import statement is that "**To refer or access the variable names, function names and class names of module in the context of current program"**=>we can use import statement in 4 ways.  
-------------------  
=>Syntax-1: import module name  
-------------------  
=>This syntax imports single module  
----------------  
Example: import icici  
 import aop  
 import mathsinfo  
--------------------------------------------------------------------  
=>Syntax-2: import module name1, module name2....Module name-n  
-------------------  
=>This syntax imports multiple modules  
----------------  
Example: import icici , aop, mathsinfo  
----------------------------------------------------------------------------------------------------------------------------  
=>Syntax-3: import module name as alias name  
-------------------  
=>This syntax imports single module and aliased with another unique names  
----------------  
Example: import icici as i  
 import aop as a  
 import mathsinfo as m  
----------------------------------------------------------------------------------------------------------------------------  
=>Syntax-4: import module name1 as alias name, module name2 as alias name......module name-n as alias name  
-------------------  
=>This syntax imports multiple modules and aliased with another unique names  
----------------  
Example: import icici as i, aop as a , mathsinfo as m  
  
=>Hence after importing module name(s) by using "import statement", all the variable names, Function names and class names must access variable names, Function names and class names w.r.t Module Names or alias names.  
 Module Name.Variable Name  
 Module Name.Function Name  
 Module Name.Class Name  
 (OR)  
 Alias Name.Variable Name  
 Alias Name.Function Name  
 Alias Name.Class Name  
============================================================================  
**2) By using from.... import statement.**  
=======================================  
**=>Here "from" "import" are the key words**=>The purpose of from.... import statement is that " To refer or access the variable names, function names and class names of any modules in the context of current program directly without writing module name OR alias name of Module name."   
=> we can use from.... import statement in 3 ways.  
-------------------  
**Syntax-1: from module name import Variable Names,Function Names, Class Names**------------------   
=>This syntax imports the Variable Names,Function Names, Class Names of a module.  
  
Example: from calendar import month  
 from aop import addop,subop  
 from mathinfo import pi,e  
 from icici import bname,addr, simpleint  
  
-----------------------------------------------------------------------------------  
**Syntax-2: from module name import Variable Names as alias name,Function Names as   
 alias name , Class Names as alias names.**-----------------------------------------------------------------------------------  
=>This syntax imports the Variable Names,Function Names, Class Names of a module with Unique alias Names  
  
Example: from calendar import month as m  
 from aop import addop as a,subop as s, mulop as m  
 from mathinfo import pi as p ,e as k  
 from icici import bname as b,addr as n , simpleint as si  
---------------------------------------------------------------------------------------------------------------------   
**Syntax-3: from module name import \***---------------  
=>This syntax imports ALL Variable Names,Function Names, Class Names of a module.  
=>This syntax is not recommmended to use bcoz it imports required Features of Module and also import un-interrested features also imported and leads more main memory space.  
  
Example: from calendar import \*  
 from aop import \*  
 from mathsinfo import \*  
  
=>Hence after importing all the variable names, Function names and class names by using "from ....import statement" , we must access variable names, Function names and class names Directly without using Module Names or alias names.  
  
 Variable Name  
 Function Name  
 Class Name  
  
=>**Hence with "import statement" we can give alias name for module names only but not for Variables Names, Function Names and Class Names.**  **Where as with "from ... import statement " we can give alias names for Variables Names, Function Names and Class Names but not for Module Name.**=======================================X=================================

==========================================  
 realoding a modules in Python  
 ==========================================  
=>To reaload a module in python , we use a pre-defined function called **reload(),** which is present in imp module and it was deprecated in favour of importlib module.  
**=>Syntax:- imp.reload(module name)  
 (OR)  
 importlib.reload(module name) ----->recommended**----------------------------------  
=>Purpose / Situation:  
-----------------------------------  
**=>reaload() reloads a previously imported module.**   
**=>if we have edited the module source file by using an external editor and we want to use the changed values/ updated values / new version of previously loaded module then we use reload().**  
===================================X======================================  
#shares.py---file and treated as module name  
def sharesinfo():  
 d={"Tech":19,"Pharma":11,"Auto":1,"Finance":00}  
 return d  
  
#main program  
#sharesdemo.py  
import shares  
import time  
import importlib  
def disp(d):  
 print("-"\*50)  
 print("\tShare Name\tValue")  
 print("-"\*50)  
 for sn,sv in d.items():  
 print("\t{}\t\t:{}".format(sn,sv))  
 else:  
 print("-"\*50)  
#main program  
d=shares.sharesinfo()   
disp(d)  
time.sleep(15)  
importlib.reload(shares) # relodaing previously imported module  
d=shares.sharesinfo() # obtaining changed / new values of previously imported module  
disp(d)

========================================================  
 **Exception Handling in Python--Most Imp**  
 ========================================================  
Index  
---------------------------------------  
=>Purpose of Exception Handling  
=>Types of Errors  
 a) Compile Time Errors  
 b) Logical Errors  
 c) Runtime Errors  
=>What is exception  
=>What is exception handling  
=>Building Points of Exception Handling  
=>Types of Exceptions  
 a) Pre-Defined OR Built-In exceptions  
 b) Programmer OR User OR Custom-Defined Exceptions  
=>Programming Examples  
------------------------------------------------------------------------  
=>Keywords for Handling the Exceptions  
 a) try  
 b) except   
 c) else  
 d) finally  
 e) raise  
=>Explanation of Keywords for Handling the Exceptions  
=>Various Forms of except block  
=>Programming Examples  
------------------------------------------------------------------------  
=>Development of Programmer OR User OR Custom-Defined Exceptions  
=>Programming Examples  
=>Use -Cases / Case Study  
=========================================================================

=================================================================  
 Introduction to Exception Handling  
 (OR)  
 Types of Errors  
 =================================================================  
=>The purpose of Exception Handling is that " To develop Robust(Strong) Applications".  
=>When we enter Valid Input then we get Valid Result   
=>**When we Enter Invalid Input then we Must Define a Code for generating User-Friendly Error Message by using Exception Handling.**  
=>In Real Time to develop any project, we must use a Programming Language. By using Programming Language, we Develop, Compile and Execute Various Programs. During this Process, we get 3 Types of Errors. They are  
 1. Compile Time Errors  
 2. Logical Errors  
 3. Runtime Errors  
------------------------------------------------------------------------  
1. Compile Time Errors  
------------------------------------------------------------------------**=>Compile Time Errors are those which are occuring during Compilation Process ( .py------>**.pyc)  
**=>Compile Time Errors occurs due to Wrong Syntaxes.**  
=>Compile Time Errors are solved by programmers during Development Level  
------------------------------------------------------------------------2. Logical Errors  
------------------------------------------------------------------------  
=>Logical Errors are those which are **occuring during Execution / Run Time**  
=>Logical Errors **occurs due to Invalid / Wrong Logics used in Program and always gives Wrong Result**  
=>Logical Errors are solved by programmers during Development Level  
------------------------------------------------------------------------3. Runtime Errors  
------------------------------------------------------------------------=>Runtime Errors are those which are occuring during Execution / Run Time.  
=>Runtime Errors occurs due to Invalid input OR Wrong Inputs Entered by END-USER OR Application User  
=>Runtime Errors are solved by programmers during Development Level by studying Forecasting Requirments of END-Users.  
------------------------------------------------------------------------

================================================  
 Explanation for the keywords used in Syntax of Handling Exception  
 ================================================  
------------------------------------------------------------------------  
1.try  
------------------------------------------------------------------------  
=>It is the block in which we write block of statements generating exceptions. In otherwords what   
 are all the statements generating exceptions, those statements must be written within try block and hence try block is called Exception monitering block.  
=>When an exception occurs in try block then PVM comes out of try block and executes appropriate except block.  
=>After executing appropriate except block, PVM never goes to try block for executing rest of the statements in try block.  
=>Every try block must be immediately followed by except block ( Otherwise we get SyntaxError)  
=>Every try block must contain atleast one except block . It is recommended to write multiple   
 except blocks for generating User-Friendly error messages.  
------------------------------------------------------------------------  
2.except  
------------------------------------------------------------------------  
=>It is the block in which we write block of statements generates User-Friendly Error Friendly Messages. In Otherwords except block suppreses Technical error messages and generates User-Freindly Error Messages and hence except block is called Exception Processing Block.  
Note: Handling exception= try block + except block  
=>except block will execute when there is an exception occurs in try block.  
=>Even we write multiple except blocks , PVM executes Appropriate except block(Single Block) depends on type of exception occurs in try block.  
=>The place for writing except block is that after try block and before else block.  
------------------------------------------------------------------------  
3.else   
------------------------------------------------------------------------  
=>It is the block in which we write block of statements will display results of the program and hence else block is called Result Generated Block.  
=>else block will execute when there is no exception occurs in try block.  
=>Writing else block is optional  
=>The place of writing else block is that after except block and before finally block (if it present).  
------------------------------------------------------------------------  
4.finally   
------------------------------------------------------------------------  
=>It is the block, in which we write block of statements will relinquish (release / close / give-up/clean-up) the resources ( Files, Database softwares) which are obtained in try block and finally block is called Resouces relinquishing Block.  
=>finally block will execute compulsorily.  
=>finally block is optional to write  
=>The place of writing finally block is that after else block ( if else block present)  
=======================x==================================================

*#Program for accepting Two Integer Value and perform Division  
#Div1.py----Not Robust Program*print("Program Execution Strated")  
s1=input("Enter First Value:")  
s2=input("Enter Second Value:")  
*#Convert s1 and s2 into int type values*a=int(s1) *# Exception Generated Statement-----ValueError*b=int(s2) *# Exception Generated Statement-----ValueError*print("First Value:{}".format(a))  
print("Second Value:{}".format(b))  
*#computer div*c=a/b *# Exception Generated Statement-----ZeroDivisionError*print("Div={}".format(c))  
print("Program Execution Ended")

*#Program for accepting Two Integer Value and perform Division  
#Div2.py--- Robust Program*try:  
 print("Program Execution Strated")  
 s1=input("Enter First Value:")  
 s2=input("Enter Second Value:")  
 a=int(s1)   
 b=int(s2)   
 c=a/b   
except ZeroDivisionError:  
 print("\tDon't Enter Zero for Den...")  
except ValueError:  
 print("\tDon't enter alnums,strs and symbols..")  
else:  
 print("\*\*\*\*\*\*\*\*\*\*else block\*\*\*\*\*\*\*\*\*\*\*")  
 print("First Value:{}".format(a))  
 print("Second Value:{}".format(b))  
 print("Div={}".format(c))  
finally:  
 print("Program Execution Ended--finally block")

========================================================================  
 Development of Programmer OR User OR Custome-Defined Exceptions ========================================================================  
=>Programmer OR User OR Custome-Defined Exceptions are those which are developed by Python Language Programmers and they are avilable as part of Python Project and used by Othert Team members for dealing with Common Problems.  
=>Some of the Common Problems are  
 a) Attempting to enter Invalid PIN in ATMs  
 b) Attempting to Invalid User Name / Passward   
 c) Attempting withdraw more amount from existing bal of account ...etc  
=>If the exception class already developed by Python Lang Developers Then it is Called Pre-Defined Exception.  
=>If the exception class developed by Python Lang Programmers Then it is Called Prtogrammer-Defined Exception.  
------------------------------------------------------------------------Steps  
------------------------------------------------------------------------  
Step-1: Choose the Programmer-Defined Class Name  
Step-2: The Programmer-Defined Class Name must Inherit From "Exception" or "BaseException" bcoz These Provides Properties of Exception Handling  
Step-3: Save the above Development on Some File Name with an extension .py (Module Name)  
------------------------------------------------------------------------  
Exmaples: Develop a Programmer-Defined Exception for Login failures  
  
 class LoginError(Exception):pass  
  
Exmaples: Develop a Programmer-Defined Exception class for Insufficient Funds  
   
 class InSuffFundError(BaseException):pass  
Exmaples: Develop a Programmer-Defined Exception class for Invalid Pin   
   
 class PinError(Exception):pass  
=========================================================================

=================================================  
 raise key word  
 =================================================  
=>raise keyword is used for hitting / raising / generating the exception provided some condition must be satisfied.  
=>raise keyword always used inside of Function Definition only.  
=>PVM uses raise keyword implicitly for hitting pre-defined Exceptions where as Programmer makes the PVM to use raise keyword explicitly for Hitting or Generating Programmer-defined Exceptions.  
  
=>Syntax-1:- if (Test Cond):  
 raise <exception-class-name>  
  
=>Syntax-2:- def functionname(list of formal parms if any):  
 -----------------------------------------------------  
 -----------------------------------------------------  
 if (Test Cond):  
 raise <exception-class-name>  
 --------------------------------------------------------  
=========================================================================

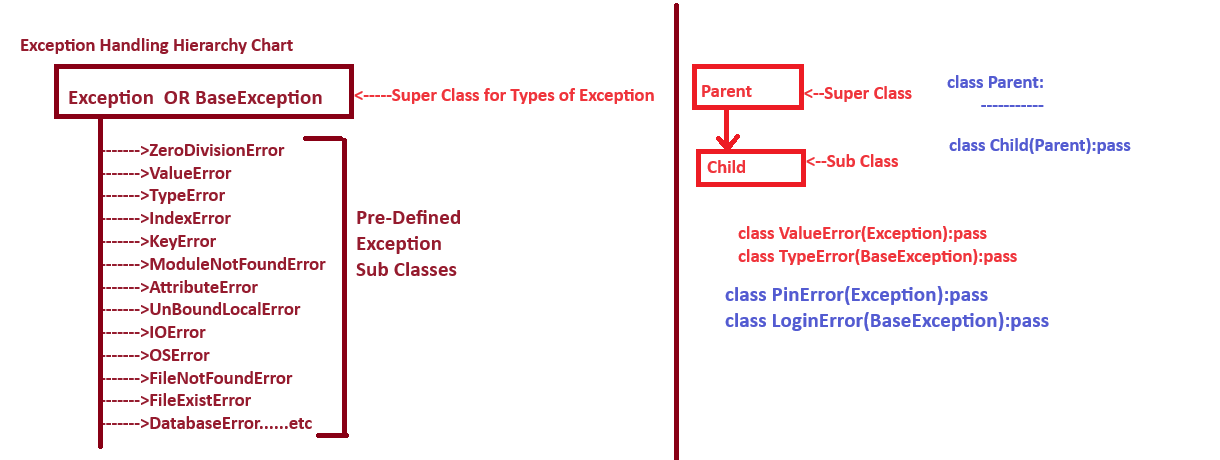
*#Program for cal Division of Two Numbers and hit the exception if Required  
#DivisionOp.py*from DivExcept import DenZeroError  
def divsion(a,b):  
 if(b==0):  
 raise DenZeroError *# Explicitly Hitting the exception* else:  
 return (a/b)  
  
*#Here Function Perform Performs and Hit the exception in the case Invalid Input  
# and gives the Result in the case of Valid Input ---Phase-2*

*#DivOpAllInOne.py*class DenZeroError(Exception):pass  
  
def divsion(a,b):  
 if(b==0):  
 raise DenZeroError *# Explicitly Hitting the exception* else:  
 return (a/b)  
*#main Program*try:  
 x=float(input("Enter First Value:"))  
 y = float(input("Enter Second Value:"))  
 res=divsion(x,y) *# Function call--gives either Result or Exception*except DenZeroError:  
 print("\tDon't Enter Zero for Den...")  
except ValueError:  
 print("\tDon't enter alnums , strs and symbols")  
else:  
 print("Div({},{})={}".format(x,y,res))  
finally:  
 print("I am from finally block")

*#MulTable.py--File Name and Module Name*from MulTabExcept import ZeroError,NegNumError  
def table(n):  
 if(n==0):  
 raise ZeroError  
 elif(n<0):  
 raise NegNumError  
 else:  
 print("\*"\*50)  
 print("Mul Table for :{}".format(n))  
 print("\*" \* 50)  
 for i in range(1,11):  
 print("\t\t{} x {}={}".format(n,i,n\*i))  
 print("\*" \* 50)

*#MulTabExcept.py----File Name and Module Name*class ZeroError(Exception):pass  
class NegNumError(Exception):pass

*#MulTableDemo.py--main program*from MulTable import table  
from MulTabExcept import ZeroError,NegNumError  
try:  
 n=int(input("Enter a Number for Generating Mul table:"))  
 table(n)  
except ZeroError:  
 print("\t\tDon't enter Zero for Mul Table")  
except NegNumError:  
 print("\t\tDon't enter Neg Number for Mul table")  
except ValueError:  
 print("\t\tDon't enter alnum,strs and symbols for mul table")



Validate name program and atm project

===============================================================  
 Types of exceptions  
 ===============================================================  
=>In Python Programming, we have Two types of exceptions. They are  
 1. Pre-Defined OR Built-In exceptions  
 2. Programmer OR User OR Custome-Defined Exceptions  
------------------------------------------------------------------------1. Pre-Defined OR Built-In exceptions  
------------------------------------------------------------------------  
=> Pre-Defined OR Built-In exceptions are those which are developed by Python Language Developers and they are available in Python Software and used by All Python Programmers for dealing Universal Problems  
=>Some of the Universal Problems are  
 a) Attempt to Convert Invalid Type of Values into Some Other Type ( ValueError, TypeError)  
 b) Attempting to zero for Denominator (ZeroDivisionError)  
 c) Attempting to enter Invalid Index value (IndexError)  
 d) Attempting to import Invalid Module Name (ModuleNotFoundError)...etc  
------------------------------------------------------------------------  
2. Programmer OR User OR Custom-Defined Exceptions  
------------------------------------------------------------------------  
=> Programmer OR User OR Custome-Defined Exceptions are those which are developed by Python Language Programmers and they are avilable as part of Python Project and used by Othert Team members for dealing with Common Problems.  
=>Some of the Common Problems are  
  
 a) Attempting to enter Invalid PIN in ATMs  
 b) Attempting to Invalid User Name / Passward   
 c) Attempting withdraw more amount from existing bal of account ...etc  
=========================================================================

=============================================================  
 Various Forms of except block   
 =============================================================  
-----------------------------------------------------------------------  
Form-1 : This except Block can handle one exception at a time for generating single User-Friendly Error Message  
-----------------------------------------------------------------------------------------------------------------------------------------------  
 try:  
 Block of statements  
 generates Exceptions  
 except <exception-class-name-1>:  
 Block of statements  
 User-Friendly Error Messages  
 except <exception-class-name-2>:  
 Block of statements  
 User-Friendly Error Messages  
 ---------------------------------------------------  
 ---------------------------------------------------  
 except <exception-class-name-n>:  
 Block of statements  
 User-Friendly Error Messages  
------------------------------------------------------------------------  
Form-2 : This except block can handle Multiple specific exceptions at a time generating Multiple User-Friendly Error   
 Message--This Facility is called Multi exception handling block  
------------------------------------------------------------------------   
try:  
 Block of statements  
 generates Exceptions  
except <exception-class-name-1,exception-class-name-2,.....exception-class-name-n>:  
 Block of statements  
 User-Friendly Error Messages for all the specific Exceptions

------------------------------------------------------------------------Form-3 : This except block can handle exceptions of Third Party Softwares related exceptions in Python Program by using ALIAS name for exception  
------------------------------------------------------------------------  
  
 try:  
 Block of statements  
 generates Exceptions  
 except <exception-class-name-1> as alias name:  
 print(alias name)  
 except <exception-class-name-2> as alias name:  
 print(alias name)  
 ---------------------------------------------------  
 ---------------------------------------------------  
 except <exception-class-name-n> as alias name:  
 print(alias name)  
------------------------------------------------------------------------  
Form-4 : we write except block without exception name called default except block.  
 The default except block can handle all types of exceptions without giving Proper user-friendly Error Messages(It is Not recommended). But Whose Real time purpose is that "To Handling Future Un-known exception at present " and It must be written at Last otherwise we get Syntaxerror  
------------------------------------------------------------------------  
 try:  
 Block of statements  
 generates Exceptions  
 except: # Default except block  
 print("Default Message")  
  
 NOTE : This Syntax like the above Not Recommeded  
=========================================================================  
Final Syntax for handling the exceptions  
--------------------------------------------------------------------------------------------------  
Syntax-1:  
--------------------------   
 try:  
 Block of statements  
 generates Exceptions  
 except <exception-class-name-1>:  
 Block of statements  
 User-Friendly Error Messages  
 except <exception-class-name-2>:  
 Block of statements  
 User-Friendly Error Messages  
 ---------------------------------------------------  
 ---------------------------------------------------  
 except <exception-class-name-n>:  
 Block of statements  
 User-Friendly Error Messages  
 except: # default except block  
 Default message  
 else:  
 -------------------------------------  
 --------------------------------------  
 finally:  
 -----------------------------------  
 -----------------------------------  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

------------------------   
Syntax-2:  
------------------------   
 try:  
 Block of statements  
 generates Exceptions  
 except <exception-class-name-1,exception-class-name-2....exception-class-name-n>:  
 Block of statements  
 User-Friendly Error Messages  
 except: # default except block  
 Default message  
 else:  
 -------------------------------------  
 --------------------------------------  
 finally:  
 ----------------------------------  
=============================================x============================

*#Program for accepting Two Integer Value and perform Division  
#DefaultExceptEx1.py---Not a Robust Program--This is not a recommended Program*try:  
 print("Program Execution Strated")  
 s1=input("Enter First Value:")  
 s2=input("Enter Second Value:")  
 a=int(s1)   
 b=int(s2)   
 c=a/b   
except: *# default except Block* print("oooops, some thing went wrong!!!")  
else:   
 print("\*\*\*\*\*\*\*\*\*\*else block\*\*\*\*\*\*\*\*\*\*\*")  
 print("First Value:{}".format(a))  
 print("Second Value:{}".format(b))  
 print("Div={}".format(c))  
finally:  
 print("Program Execution Ended--finally block")

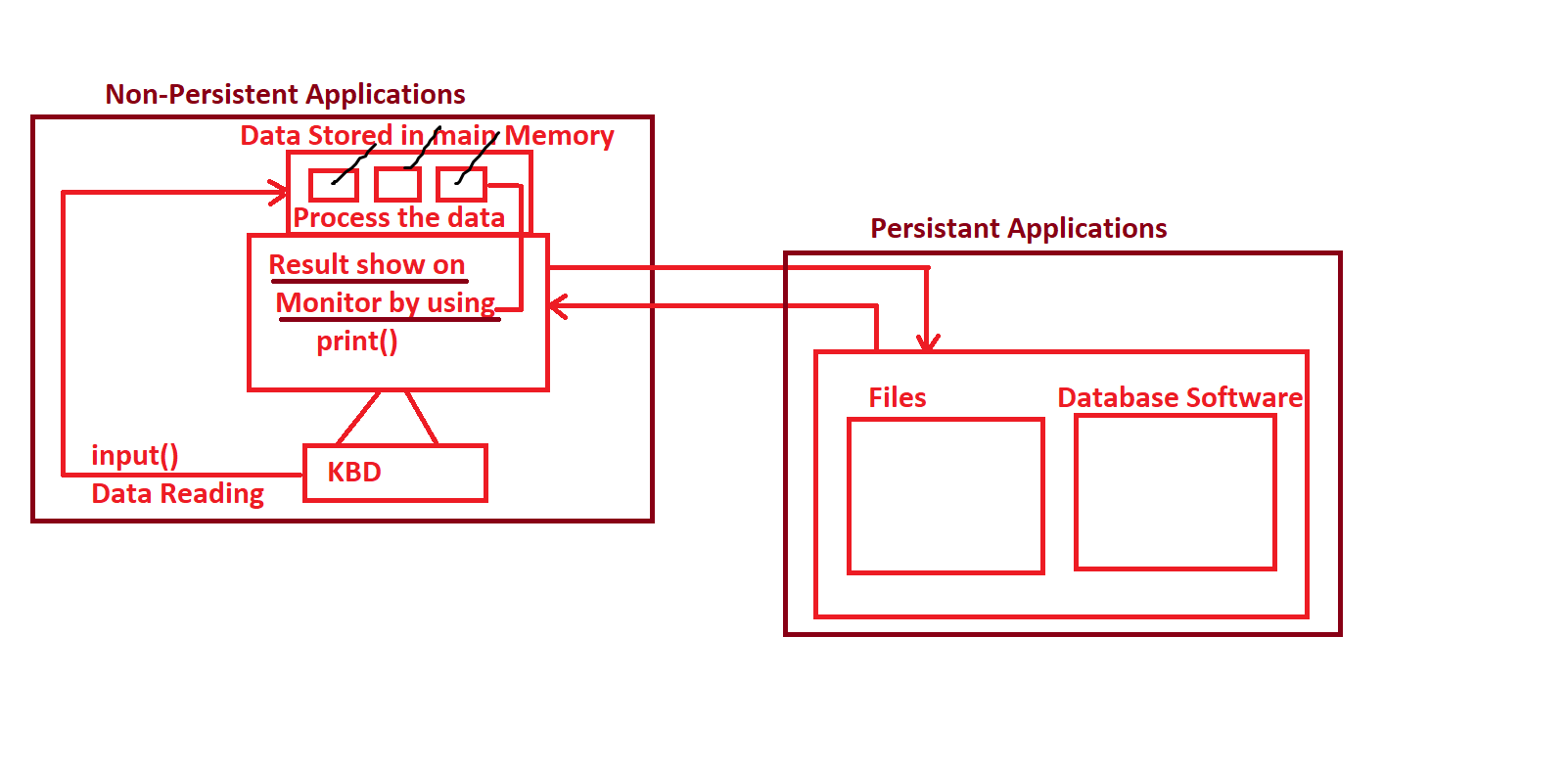
====================================================  
 Files OR Sterams in Python--5 days  
 ====================================================  
Index  
----------------------  
=>Purpose of Files  
=>Types of applications   
 a) Non-Persistant Applications  
 b) Persistant Applications  
=>Definition of File, Stream  
=>Operations on Files  
 a) Write Opertion  
 b) Read Operation  
=>Types of Files  
 a) Text Files  
 b) Binary Files  
---------------------------------------------------------------------------  
=>File Opening Modes  
 1. r 2. w 3. a  
 4. r+ 5. w+ 6. a+  
 7. x 8. x+   
=>Number of Approaches to Open the Files  
 a) By using "open()"  
 b) By using "with open() as "   
=>Programming Examples  
---------------------------------------------------------------------------

=>Functions for Writing the Data to the file  
 a) write()  
 b) writelines()  
=>Programming Examples  
=>Functions for Reading the Data from the file  
 a) read()  
 b) readlines()  
=>Programming Examples  
=>Random Access files  
 a) tell()  
 b) seek()  
=>Programming Examples  
---------------------------------------------------------------------------  
=>Pickling (Object Serialization) and Un-Pickling(Object De-Serialization) Operation  
=>Implemenattion Pickling and Un-Pickling Operation.  
=>pickle module  
 a) dump()  
 b) load()  
=>Programming Examples  
---------------------------------------------------------------------------  
=>Working with CSV Files  
=>Implementation of CSV Files  
=>csv module  
 a) csv.reader()  
 b) csv.writer()  
 c) csv.DictReader()  
 d) csv.DictWriter()  
=>Programming Examples  
---------------------------------------------------------------------------  
=>Working with OS Based Operations  
=>os module  
=>Programming Examples  
===============================================x=============================

===============================================  
 Types of Files in Python  
 ===============================================  
=>In any Programming lang, we have Two Types of Files. They are  
  
 1. Text Files  
 2. Binary Files  
-------------------------------------------------------------------------  
1. Text File  
----------------------------------------------------------------------  
=>A Text File is One, which always contains alphabets, Digits and Special Symbols.  
=>Text Files are denoted by a letter "t". (by deafult)  
=>By deafult Python Programming Lang Treats every file as Text File.  
Examples: .py .java .cpp .c  
 .txt .xlsx, .doc...etc  
---------------------------------------------------------------------------2. Binary File  
---------------------------------------------------------------------------=>A Binary File is One, which contains the data in the form of Binary Format (Pixels).  
=>Binary Files are denoted by a letter "b".  
----------------------  
=>Examples  
----------------------  
 => Images (.png, .jpeg, .jpg, .gif....etc)  
 => Audio and Video Files (.mvi, .avi...etc)  
 => PDF File with Images  
===================================================================

=================================================  
 Types of Application in Files  
 =================================================  
=>The purpose of Files in any programming language is that " To maintain Data Persistency".  
=>The Process of storing the data permanently is called Data Persistency.  
=>In this context, we can develop two types of applications. They are  
 1) Non-Persistant Applications  
 2) Persistant Applications  
=>In Non-Persistant Applications development, we read the data from Keyboard , stored in main memory(RAM) in the form of objects, processed and whose results displayed on Moniter.  
Examples: ALL our previous examples comes under Non-Persistant Applications.  
  
=>We know that Data stored in Main Memory(RAM) is temporary.  
  
=>In Persistant Applications development, we read the data from Keyboard , stored in main memory(RAM) in the form of objects, processed and whose results stored Permanently.  
=>In Industry, we have two ways two store the Data Permanently. They are  
 1) By using Files  
 2) By Using RDBMS DataBase Softwares ( Oracle, MySQL,   
 MongoDB, DB2,PostgreySQL,SQL Server,SQLITE3..etc)  
========================================x====================================

============================================  
 Data Persistenecy by Using Files of Python   
 ============================================  
-----------------------------------------------------  
Def. of File:  
-----------------------------------------------------  
=>A File is a collection of Records.  
=>Files Resides in Secondary Memory(HDD).  
=>Technically, File Name is considered as a named location in Secondary Memory.  
=>The purpose of Files is that "To get Data Persistency".  
-----------------------------------------------------  
=>All the objects data of main memory becomes records in File of Secondary memory and records of file of secondary memory becomes the objects in main memory.  
----------------------------------------------------  
Def. of Stream:  
----------------------------------------------------  
**=>The Flow of Data between object(s) of Main Memory and Files of Seconday memory is called Stream.**  
---------------------------------------------------------------------------



==============================================  
 Operations on Files  
 ==============================================  
=>On the files, we can perform Two Types of Operations. They are  
 1) Write Operation  
 2) Read Operation.  
------------------------------  
1) Write Operation:  
------------------------------  
=>The purpose of write operation is that " To transfer or save the object data of main memory as record in the file of secondary memory".  
=>Steps:   
 1) Choose the File Name  
 2) Open the File Name in Write Mode   
 3) Perform cycle of Write Operations.  
=>While we are performing write operations, we get the following exceptions.  
 **a) IOError  
 b) OSError  
 c) FileExistError**------------------------------  
2) Read Operation:  
------------------------------  
=>The purpose of read operation is that " To transfer or read the record from file of secondary memory into the object of main memory".  
=>Steps  
 a) Choose the file name  
 b) Open the file name in Read Mode   
 c) Perform cycle of read operations.  
=>While we are performing read operations, we get the following exceptions.  
 a) **FileNotFoundError  
 b) EOFError**================================x============================================

===========================================================  
 File Opening Modes  
 ===========================================================  
=>File Opening Modes concept makes us to understand, In which mode the file is opening.  
=>In Python Programming, we have 8 File Opening modes. They are  
 1. r   
 2. w  
 3. a  
 4. r+   
 5. w+   
 6. a+  
 7. x   
 8. x+  
---------------------------------------------------------------------------  
1. r  
---------------------------------------------------------------------------=>The "r" mode is used for Opening the File Name in Read Mode for Performing Read Operation.  
=>If we open the file name in"r" mode and if that file name does not exist then we get **FileNotFoundError**.  
=>If we don't sepcify the any file mode then PVM Takes the default mode as "r" mode.  
---------------------------------------------------------------------------2. w  
---------------------------------------------------------------------------=>The "w" Mode is used for Creating File Name and Open it write Mode and Performs Write Operations.  
=>If we Choose NEW FILE and opening in "w" then Creating New File Name in Write Mode and Performs Write Operations.  
=>If we Choose EXISTING FILE and opening in "w" then Existing File Data OVERLAPPED with NEW DATA.  
---------------------------------------------------------------------------  
3. a  
---------------------------------------------------------------------------=>The "a" Mode is used for Creating File Name and Open it **write Mode** and Performs Write Operations.  
=>If we Choose NEW FILE and opening in "a" then Creating New File Name in Write Mode and Performs Write Operations.  
=>If we Choose EXISTING FILE and opening in "a" then Existing File Data APPENDED with NEW DATA.  
---------------------------------------------------------------------------4. r+  
---------------------------------------------------------------------------=>The "r+" mode is used for Opening the File Name in Read Mode and First Performs Read Operation and Later we can also perform Write Operation.  
=>If we open the file name in"r+" mode and if that file name does not exist then we get **FileNotFoundError.**  
---------------------------------------------------------------------------5. w+  
---------------------------------------------------------------------------  
=>The "w+" Mode is used for Creating File Name and Open it write Mode and Performs Write Operations and later we can also read the data from the file.  
=>If we Choose NEW FILE and opening in "w+" then Creating New File Name in Write Mode and Performs Write Operations and later we can also read the data from the file..  
=>If we Choose EXISTING FILE and opening in "w+" then Existing File Data OVERLAPPED with NEW DATA.  
---------------------------------------------------------------------------6. a+  
---------------------------------------------------------------------------=>The "a+" Mode is used for Creating File Name and Open it write Mode and Performs Write Operations and later we can also read the data from the file.  
=>If we Choose NEW FILE and opening in "a+" then Creating New File Name in Write Mode and Performs Write Operations and later we can also read the data from the file..  
=>If we Choose EXISTING FILE and opening in "a+" then Existing File Data APPENDED with NEW DATA.  
---------------------------------------------------------------------------7. x  
---------------------------------------------------------------------------  
=>The "x" mode is used for Opening The File Name eXclusively in write Mode and Performs Write Operations.  
=>If we open existing file name in "x" mode then we get **FileExistError.**---------------------------------------------------------------------------8. x+  
---------------------------------------------------------------------------=>The "x+" mode is used for Opening The File Name eXclusively in write Mode and Performs Write Operations and later we can also perform Read Operations.  
=>If we open existing file name in "x+" mode then we get **FileExistError.**---------------------------------------------------------------------------

===========================================================  
 Number of Approaches to Open the Files  
 ===========================================================  
=>In Python Programming, we have 2 Approaches to Open the file. They are  
  
 1. By using open()  
 2. By using "with open() as "  
---------------------------------------------------------------------------1. By using open()  
---------------------------------------------------------------------------  
Syntax: varname=open("File Name","File Mode")  
=>varname is an object of type <class, \_io.TextIOWrapper> and It is treated as File Pointer, which is always points to the file.  
=>open() is pre-defined function present in builtins module, which is used for Opening the Specified File Name in Specified File Mode  
=>File Name represent Name of the file   
=>The File Modes in Python are "r", "w", "a", "r+", "w+" , "a+", "x", "x+"  
=>Once we open any file name with open() then we must close the File by using close() and It is mandatory for maintaining Consistency of Data(Manual Closing).  
---------------------------------------------------------------------------2. By using "with open() as "  
---------------------------------------------------------------------------Syntax: with open("File Name","File Mode") as varname:  
 ------------------------------------------------------  
 -------------------------------------------------------  
 Block of Statements--peforms File Operations  
 ------------------------------------------------------  
 ------------------------------------------------------  
 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
 Other statements in Program  
 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
  
------------------  
Explanation:  
-----------------  
=>"with" and "as" are the keywords  
=>open() is a pre-defined fucntion present in builtins module and It is used for Opening the Filename in Specified File Mode.  
 =>"FileName" represents Name of the file  
=>"FileMode" represents r,w,a,r+,w+,a+,x,x+  
=>Varname represents an object pointing the file and it is called File Pointer and whose type is <class, \_io.TextIOWrapper>  
=>The execution Process of "with open(---) as " is that "As Long as PVM present in side of " with open(---) as " Indentation then File Name is actively Available and once PVM comes out of " with open(---) as" Indentation then File name closed Automatically and This Facility is Called Auto-Closeability of File". No Need to close the file by using close() manually.  
=============================================================================

*#Program for Demonstrating How to open the file  
#FileOpenEx1.py*try:  
 fp=open("kvr.data","r")  
except FileNotFoundError:  
 print("File Does not Exist")  
else:  
 print("File Name opened in Read Mode sucessfully")  
 print("Type of fp=", type(fp))  
 print("Is File Closed=", fp.closed)  
finally:  
 fp.close() *# manual closing of file* print("finally block---Is File Closed=", fp.closed)  
   
*#Program for Demonstrating How to open the file  
#FileOpenEx2.py*fp=open("kvr.py","w")  
print("File Opened in write mode ")  
print("Type of fp=",type(fp))  
  
  
*#Program for Demonstrating How to open the file  
#FileOpenEx3.py*try:  
 with open("kvr.data","r") as fp:  
 print("\*\*\*\*\* within with open() as\*\*\*\*\*\*\*\*\*")  
 print("File Name opened in Read Mode sucessfully")  
 print("Type of fp=", type(fp))  
 print("Is File Closed=", fp.closed)  
 print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")  
 print("\_\_\_\_\_after with open() as\_\_\_\_\_\_\_")  
 print("Is File Closed=", fp.closed)  
except FileNotFoundError:  
 print("File does not exist")  
   
   
*#Program for Demonstrating How to open the file  
#FileOpenEx4.py*with open("kvr1.data","a+") as fp:  
 print("\*"\*40)  
 print("File Opened in write mode sucessfully")  
 print("Type of fp=",type(fp))  
 print("\*" \* 40)  
 print("Misc Information about Files")  
 print("\tFile Name=",fp.name)  
 print("\tFile Mode=",fp.mode)  
 print("\tIs File Readable=",fp.readable())  
 print("\tIs File Writeable=",fp.writable())  
  
  
*#Program for Demonstrating How to open the file  
#FileOpenEx5.py*try:  
 with open("kvr2.data","x") as fp:  
 print("\*"\*40)  
 print("File Opened in write mode sucessfully")  
 print("Type of fp=",type(fp))  
 print("\*" \* 40)  
 print("Misc Information about Files")  
 print("\tFile Name=",fp.name)  
 print("\tFile Mode=",fp.mode)  
 print("\tIs File Readable=",fp.readable())  
 print("\tIs File Writeable=",fp.writable())  
except FileExistsError:  
 print("File alerady exist")  
  
*#Program for Demonstrating How to open the file  
#FileOpenEx6.py*try:  
 with open("e:\\kvr3.data","x+") as fp:  
 print("\*"\*40)  
 print("File Opened in write mode sucessfully")  
 print("Type of fp=",type(fp))  
 print("\*" \* 40)  
 print("Misc Information about Files")  
 print("\tFile Name=",fp.name)  
 print("\tFile Mode=",fp.mode)  
 print("\tIs File Readable=",fp.readable())  
 print("\tIs File Writeable=",fp.writable())  
except FileExistsError:  
 print("File alerady exist")

======================================================  
 Writing the Data to the file  
 ======================================================  
=>To write the Data to the File, we have Two Pre-Defined Functions present in File Pointer which is an object of TextIOWrapper.They are  
 1. write()  
 2. writelines()  
---------------------------------------------------------------------------1. write()  
---------------------------------------------------------------------------  
Syntax: FilePointerObject.write(str data)  
=>This Function is used for writing or Saving any type of Data to the file in the form of str  
---------------------------------------------------------------------------2. writelines()  
---------------------------------------------------------------------------  
Syntax: FilePointerObject.writelines(Iterable-Object)  
=>This Function is used for writing or Saving any type of Iterable Object Data to the file in the form of str  
---------------------------------------------------------------------------  
NOTE: here write() and writelines() can write Value by Value to the file

======================================================  
 Reading the Data From the File  
 ======================================================  
=>To Read the Data from the File, we have 2 Pre-Defined Functions present in File Pointer Object of type TextIoWrapper. They are  
 1. read()  
 2. readlines()  
---------------------------------------------------------------------------1. read()  
---------------------------------------------------------------------------  
=>Syntax: varname=filpointerobject.read()  
=>Here read() is used for reading Entire Content of the File in the form of Value by Value and placed in LHS Variable of type <class, 'str'>.  
---------------------------------------------------------------------------  
2. readlines()  
----------------------------------------------------------------------------=>Syntax: varname=filpointerobject.readlines()  
=>Here read() is used for reading Entire Content of the File in the form of Value by Value and placed in LHS Variable of type <class, 'list'>.  
---------------------------------------------------------------------------NOTE: here read() and readlines() can read the values in the form of Value by Value from the file

*#program for Demonstrating writing the Data to the file  
#FileWriteEx1.py*sno=30  
sname="Jon Hunter"  
marks=63.75  
*#Save the above Objects Data to the file*with open("student.data","a") as fp:  
 fp.write(str(sno)+"\t")  
 fp.write(sname+"\t")  
 fp.write(str(marks)+"\n")  
 print("Student Data Written to the File")  
  
*#program for Demonstrating writing the Data to the file  
#FileWriteEx2.py*sno=int(input("Enter Student Number:"))  
sname=input("Enter Student Name:")  
marks=float(input("Enter Student Marks:"))  
*#Save the above Objects Data to the file*with open("student.data","a") as fp:  
 fp.write(str(sno)+"\t")  
 fp.write(sname+"\t")  
 fp.write(str(marks)+"\n")  
 print("Student Data Written to the File")  
  
  
  
*#program for Demonstrating writing the Iterable Object Data to the file  
#FileWriteEx3.py*x={10:"Python",20:"Java",30:"HTML",40:"Django"} *# Here x is called dict object*with open("student.data","a") as fp:  
 fp.writelines(str(x)+"\n")  
 print("Iterable Object Data Written to the File")  
  
  
*#program for Writing dynamic data to the file until we stop entering by pressing a symbol @  
#FileWriteEx4.py*with open("Hyd.info","a") as fp:  
 print("Enter the Data and Press @ to stop:")  
 while(True):  
 kbdata=input()  
 if(kbdata!="@"):  
 fp.write(kbdata+"\n")  
 else:  
 print("Data Written to the File--Verify")  
 break

*#Program for Copying the Image  
#ImageFileCopyEx.py*try:  
 srcfile=input("Enter Source File Name:")  
 with open(srcfile,"rb") as rp:  
 destfile=input("Enter Destination File:")  
 with open(destfile,"wb") as wp:  
 srcfiledata=rp.read() *# reading the data from SRC File* wp.write(srcfiledata) *# writing SRC File Data to the Dest File* print("Image File Copied ---Verify")  
except FileNotFoundError:  
 print("Source File Does not Exist:")

*#Program for Reading The Data from the file  
#FileReadEx1.py*try:  
 with open("hyd.info","r") as fp:  
 filedata=fp.read()  
 print("---------------------------------------")  
 print("Content of File")  
 print("---------------------------------------")  
 print(filedata)  
 print("---------------------------------------")  
except FileNotFoundError:  
 print("File Does not Exist")  
   
  
*#program for displaying the content of any file name  
#FileReadEx2.py*try:  
 filename=input("Enter Any File Name:")  
 with open(filename,"r") as fp:  
 filedata=fp.read()  
 print("---------------------------------------")  
 print("Content of File")  
 print("---------------------------------------")  
 print(filedata)  
 print("---------------------------------------")  
except FileNotFoundError:  
 print("File Name does not Exist")  
   
*#program for displaying the content of any file name  
#FileReadEx3.py*try:  
 filename=input("Enter Any File Name:")  
 with open(filename,"r") as fp:  
 filedata=fp.readlines()  
 print("---------------------------------------")  
 print("Content of File")  
 print("---------------------------------------")  
 for line in filedata:  
 print(line,end="")  
 print()  
 print("---------------------------------------")  
except FileNotFoundError:  
 print("File Name does not Exist")

*#program for Demonstrating Random Access File  
#RandomAcessFileEx.py  
#tell()---->Gives Index of File Pointer  
#seek()--->will re-set the File Pointer to Specified Index*with open("kvr.data","r") as fp:  
 print("Initial Pos of fp=",fp.tell()) *# 0* filedata=fp.read(6)  
 print("File Data=",filedata) *# PYTHON* print("Now Pos of fp=", fp.tell()) *# 6* filedata = fp.read(4)  
 print("File Data=", filedata) *# \_IS\_* print("Now Pos of fp=", fp.tell()) *# 10* filedata = fp.read(3)  
 print("File Data=", filedata) *# AN* print("Now Pos of fp=", fp.tell()) *# 14* filedata = fp.read(3)  
 print("File Data=", filedata) *# OOP* print("Now Pos of fp=", fp.tell()) *#17* filedata = fp.read()  
 print("File Data=", filedata) *# LANG AND ALSO FUN LANG* print("Now Pos of fp=", fp.tell()) *#41* print("-------------------------------------")  
 filedata = fp.read()  
 print("File Data=", filedata) *# No Data Displayed* print("Now Pos of fp=", fp.tell()) *#41* print("-------------------------------------")  
 fp.seek(14) *# seek() will re-set the File Pointer Index* filedata = fp.read()  
 print("File Data=", filedata) *# No Data Displayed* print("Now Pos of fp=", fp.tell()) *# 41* print("-------------------------------------")

=================================================  
 Pickling and Un-Pickling   
 (OR)  
 Object Serialization or Object De-Serialization  
 =================================================  
--------------  
Pickling ( Object Serialization )  
--------------  
=>Let us assume there exist an object which contains multiple values. To save or write an object data of main memory into the file of secondary memory by using write() and writelines() , they transfers the values in the form of value by value and it is one of the time consuming process( bcoz of multiple write operations).  
=>To Overcome this time consuming process, we must use the concept of Pickling.  
=>The advantage of pickling concept is that with single write operation , we can save or write entire object data of main memory into the file of secondary memory.  
 ----------------------------------  
=>Definition of Pickling:  
 ----------------------------------  
 =>The Process of saving or transfering entire object content of main memory into the file of secondary memory by performing single write operation is called Pickling.  
=>Pickling concept participates in Write Operations.  
-----------------------------------------------------------  
Steps for implementing Pickling Concept:  
-----------------------------------------------------------  
=>import pickle module, here pickle is one of the pre-defined module  
=>Choose the file name and open it into write mode.  
=>Create an object with collection of values (Iterable object)  
=>use the dump() of pickle module. dump() save the content of any object into the file with single write operation.  
 Syntax: pickle.dump(object , filepointer)  
=>NOTE That pickling concept always takes the file in Binary Format.  
---------------------------------------------------------------------------  
Un-Pickling (Object De-Serialization)  
--------------------------------------------------------  
=>Let us assume there exists a record with multiple values in a file of secondary memory. To read or transfer the entire record content from file of secondary memory, if we use read(), readlines() then they read record values in the form of value by value and it is one of the time consuming process( bcoz of multiple read operations).  
=>To overcome this time consuming process, we must use the concept of Un-pickling.  
=>The advantange of Un-pickling is that with single read operation, we can read entire record content from the file of secondary memory into the object of main memory.  
--------------------------------------------  
=>Definition of Un-Pickling:  
--------------------------------------------  
=>The process of reading or transfering the entire record content from file of secondary memory into the object of main memory by performing single read operation is called Un-pickling.  
=>Un-Pickling concept participates in Read Operations.  
---------------------------------------------------------------------------  
Steps for implementing Un-Pickling Concept:  
---------------------------------------------------------------------------  
=>import pickle module  
=>Choose the file name and open it into read mode.  
=>Use the load() of pickle module. load() is used for transfering or loading the   
 entire record content from file of secondary memory into object of main memory.  
 Syntax: objname=pickle.load(filepointer)  
=>NOTE That Un-pickling concept always takes the file in Binary Format.  
---------------------------------------------------------------------------

*#Program for reading employee details and save them as Record in Emploiyee File  
#EmpPickEx1.py--------pickling Process--Progrtam-(A)*import pickle  
def saveempdata():  
 with open("employee.data","ab") as fp:  
 *#read employee values from KBD* print("-"\*50)  
 eno=int(input("Enter Employee Number:"))  
 ename=input("Enter Employee Name:")  
 sal=float(input("Enter Employee Salary:"))  
 dsg=input("Enter Employee Designation:")  
 print("-" \* 50)  
 *#add the employee values to list object* lst=[] *# Create an empty list and append employee values* lst.append(eno)  
 lst.append(ename)  
 lst.append(sal)  
 lst.append(dsg)  
 *#Save lst data to the file* pickle.dump(lst,fp)  
 print("Employee Record Saved in File Sucessfully ")  
 print("-" \* 50)  
*#main program*saveempdata()  
  
  
*#Program for reading employee details and save them as Record in Emploiyee File  
#EmpPickEx2.py--pickling Process--Progrtam-(A)*import pickle  
def saveempdata():  
 with open("employee.data","ab") as fp:  
 while(True):  
 *#read employee values from KBD* print("-"\*50)  
 eno=int(input("Enter Employee Number:"))  
 ename=input("Enter Employee Name:")  
 sal=float(input("Enter Employee Salary:"))  
 dsg=input("Enter Employee Designation:")  
 print("-" \* 50)  
 *#add the employee values to list object* lst=[] *# Create an empty list and append employee values* lst.append(eno)  
 lst.append(ename)  
 lst.append(sal)  
 lst.append(dsg)  
 *#Save lst data to the file* pickle.dump(lst,fp)  
 print("Employee Record Saved in File Sucessfully ")  
 print("-" \* 50)  
 ch=input("Do u want to Insert Another Emp Record(yes/no):")  
 if(ch.lower()=="no"):  
 break  
*#main program*saveempdata()

*#Program for Reading employee Records from employee File  
#EmpUnPickEx1.py----Un-pickling Process--Program-(B)*import pickle  
def readrecord():  
 with open("employee.data","rb") as fp:  
 print("--------------------------------------------------")  
 while(True):  
 try:  
 record=pickle.load(fp)  
 for val in record: *# record=[100, 'RS', 2.3, 'Author']* print("\t{}".format(val),end=" ")  
 print()  
 except EOFError:  
 print("--------------------------------------------------")  
 break  
*#main program*readrecord() *# Function call*

*#EmpUnPickSearchEx1.py*import pickle  
def searchrecord():  
 with open("employee.data","rb") as fp:  
 print("--------------------------------------------------")  
 records=[]  
 while(True):  
 try:  
 record=pickle.load(fp)  
 records.append(record)  
 except EOFError:  
 break  
  
 eno=int(input("Enter Emp Number for getting Other details:"))  
 res=False  
 for record in records:  
 if(eno==record[0]):  
 res=True  
 break  
 if(res):  
 print("\*" \* 50)  
 print("Employee Name:{}".format(record[1]))  
 print("Employee Salary:{}".format(record[2]))  
 print("Employee Designation:{}".format(record[3]))  
 print("\*" \* 50)  
 else:  
 print("Employee Number does not exist")  
  
*#main program*searchrecord()

================================  
 JSON file  
 ================================  
=>JSON (JavaScript Object Notation) is a popular data format used for representing structured data.  
It's common to transmit and receive data between a server and Client web application development in the form of JSON format.  
=>In otherwords,JSON is a lightweight data format for data interchange which can be easily read and written by humans,  
easily parsed and generated by machines.  
=>It is a complete language-independent text format. To work with JSON data, Python has a built-in module called json.  
=================================================  
Parse JSON (Convert from JSON to Python Dict)  
-------------------------------------------------------------------------------------  
=>json.loads() method can parse a json string and converted into Python dictionary.  
Syntax:  
 dictobj=json.loads(json\_string)  
---------------------  
Examples:  
---------------------  
# Python program to convert JSON to Python  
 import json  
 # JSON string  
 employee = ' {"id":"09", "name": "Rossum", "department":"IT"} '  
 # Convert JSON string to Python dict  
 employee\_dict = json.loads(employee)  
 print(employee\_dict)  
------------------------------------------------------------------------Python--- read JSON file  
------------------------------------------------------------------------  
=>json.load() method can read the data from JSON file which contains a JSON Data.  
Syntax:  
 dictobj=json.load(file\_object)  
  
Examples:  
---------------  
#Program for JSON File Data into Dict Object  
#JsonFiletoDict.py---reading the data from JSON File to Dictobj  
import json  
try:  
 with open("emp.json","r" ) as fp:  
 dictobj=json.load(fp)  
 print(dictobj,type(dictobj))  
 print("-------------------------------------------------")  
 for k,v in dictobj.items():  
 print("\t{}-->{}".format(k,v))  
 print("-------------------------------------------------")  
except FileNotFoundError:  
 print("Json File does not exist")  
------------------------------------------------------------------------  
Python--- write to JSON file  
------------------------------------------------------------------  
=>json.dump() method can be used to write dict object data to a JSON file.   
Syntax:  
 json.dump(dict object, file\_pointer)  
-----------------  
Examples:  
-------------------  
#Program for Dict data into JSON File  
#DicttoJsonFile.py----Writing Dict data to JSON File  
import json  
dictobj={"ENO":100,"ENAME":"TRAVIS","SAL":56,"DSG":"AUTHOR"}  
with open("emp.json","w") as fp:  
 json.dump(dictobj,fp) # Here dump() is saving dictobj data into the json file  
 print("Dict Data Saved in JSON FILE Format--verify")  
  
------------------------------------------------------------------------

Student.json

{"sno": 100, "name": "Rossum", "marrks": 56.78,"ColName":"OUCET"}

Dicttojson

*#Program for Saving Dict Object Data into JSON File by using dump()  
#DictToJsonFile.py*import json  
d={"sno":100,"name":"Rossum","marrks":56.78} *# dict Object*with open("student.json","w") as fp:  
 json.dump(d,fp)  
 print("Dict Data Saved in a File--verify")

jsontodict

*#Program for Reading the Dict Data from JSON File by using load()  
#JSONFileToDict.py*import json  
with open("student.json","r") as fp:  
 d=json.load(fp)  
 for key,val in d.items():  
 print("\t{}--->{}".format(key,val))

*#Program for converting JSON String data into dict object---loads()  
#JsonStrToDict.py*import json  
print("-----------------------------------------------------------")  
jsonstr='{"sno":"100" ,"sname":"Rossum","marks":"34.56","colname":"PSFUINV" }'  
print(jsonstr,type(jsonstr))  
*#Convert json str data into dict type data--we use json.loads(jsonstr data)*print("------------------------------------------------------------")  
d=json.loads(jsonstr)  
print("Content of d={} and Type={}".format(d,type(d)))  
for key,val in d.items():  
 print("\t{}--->{}".format(key,val))  
print("-----------------------------------------------------------------")