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

Literals and Its Types

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

=>A Literal is nothing but a value passing as input to the program.

=>In Python Programming, Primary, Literals are classified into 5 types. They are

1. Integer Literals------Example---> 234 567 23

2. String Literals--------Examples--->"Python", "Rossum", "Ram"

3. Float Literals---------Examples----> 34.56 4.5 99.99 0.999

4. Boolean Literals-----Examples-----> True False

5. Date Literals-----------Examples: ----> 29-08-2022, 17-08-2022...etc

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

Identifiers or Variables

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

=>We know that all types of Literals are stored in main memory by allocating Sufficient amount of Memory with help of Data Types. To Process the Data / Literals stored in main memory, we must give distinct names to the created memory space and these distinct names makes us to identify the values and they are also called IDENTIFIERS.

=>Ducting Program Execution IDENTIFIER Values can be changed / Varying and hence IDENTIFIERs are called VARIABLES.

=>Hence All Types of LITERALS Must Be stored in the form of VARIABLES.

=>In Python Programming All Variables are called Objects.

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

Definition of Variable

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

=>A Variable is one of the Identifier whose can value(s) can be changed during Program execution.

=>In Programming Language to do any data processing, we must use Variables Objects (Python).

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

Rules for Using Variables or Identifiers in Python Program

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

=>To Use Variables in Python Programming, we use the following Rules.

1. A Variable Name is a combination of Alphabets, Digits and a Special Symbol Under Score (\_).

2. First Letter of the Variable Names must starts Either with Alphabet or Special Symbol Under Score (\_ )

Examples:

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

sal=23------valid

$sal=45----Invalid

@name="python"----Invalid

-sal=45-----Invalid

2sal=56----Invalid

456=3.4---Invalid

\_sal\_=45--valid

\_=56---valid

\_\_=5.6--valid

--=45----invalid

3. Within the variable name, No special symbols are allowed except Under Score (\_ )

Examples:

tot sal=45----Invalid

tot\_marks=456--valid

tot#sal=56-----NameError

4. No Keywords to be used as Variable Names (bcoz Keywords are the Reserved Words and they give special Meaning to the compilers).

Example:

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

if=45---------Invalid

else=67---invalid

for=4.5----Invalid

if1=56--Valid

\_else=67--valid

\_for\_=5.6--valid

Note:All Class Name can be used as Variable Names bcoz Class Names are not Keywords

5. All Variable Name are Case Sensitive

Examples:

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

>>> age=99----------------Valid

>>> AGE=98---------------Valid

>>> Age=97---------------Valid

>>> aGe=96--------------Valid

>>> print(age,AGE,Age,aGe)----- 99 98 97 96

>>> a=12

>>> A=13

>>> print(a,A)---------- 12 13

--------------------------------------------------------------------------- Data Types in Python

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

=>The Purpose of Data Types in Python is that " To allocate Sufficient amount of memory space for storing inputs in main memory of computer".

=>In Python Programming, we have 14 Data Types and They are Classified into 6 types.

I. Fundamental Category Data Types

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

1. int

2. float

3. bool

4. complex

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

II.Sequence Category Data Types

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

1. str

2. bytes

3. bytearray

4. range

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

III. List Category Data Types (Collections Data Types or Data Structures)

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

1. list

2. tuple

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

IV. Set Category Data Types (Collections Data Types or Data Structures)

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

1. set

2. frozenset

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

VI. Dict Category Data Types (Collections Data Types or Data Structures)

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

1. dict

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

VI. NoneType Category Data Types

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

1. NoneType

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

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

I. Fundamental Category Data Types

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

=>The purpose of Fundamental Category Data Types is that " To store Single Value".

=>In Python Programming, we have 4 data types in Fundamental Category. They are

1. int

2. float

3. bool

4. complex

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

Properties

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

=>'int' is one of the pre-defined classes and treated as Fundamental Data Type.

=>The purpose of int data type is that " To store Integer Data or Whole Numbers or Integral Values (Numbers or digits without decial Places) and Different Number System data".

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

Examples:

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

Python Instructions Output

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

>>> a=100

>>> b=123

>>> c=a+b

>>> print(a,type(a))------------------------------100 <class 'int'>

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

>>> print(c,type(c))------------------------------223 <class 'int'>

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

=>with int data type we can also Store Different types of Number Systems Values.

=>In Programming languages, we we have 4 Types of Number Systems. They are

1. Decimal Number System (default)

2. Binary Number System

3. Octal Number System

4. Hexa Decimal Number System

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

1. Decimal Number System (default)

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

=>This is one of the default number System.

=>This Number System Contains

Digits: 0 1 2 3 4 5 6 7 8 9 ------Total Digits =10

Base: 10

=>All Base 10 Literals are called Integer Data.

=>By default, python Execution Environment always displays the result in the form decimal number System.

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

2. Binary Number System

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

=>This Number System Contains

Digits: 0 1 ------Total Digits =2

Base: 2

=>All Base 2 Literals are called Binary Data.

=>To Store Binary Data in python environment, The Binary Data Must be preceded by a letter

'b' or 'B'

=>Syntax: varname=0b Binary data

(OR)

varname=0B Binary data

=>When we store the Binary data in python environment,python Execution Environment

converts automatically into decimal number System data.

Examples:

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

>>> a=0b1010

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

>>> bin (10) -----------------------------------'0b1010'

>>> a=0B1111

>>> print(a,type(a))--------------------------15 <class 'int'>

>>> a=0b10120-------------------------------SyntaxError: invalid digit '2' in binary literal

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

3. Octal Number System

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

=>This Number System Conatins

Digits: 0 1 2 3 4 5 6 7 ------Total Digits =8

Base: 8

=>All Base 8 Literals are called Octal Data.

=>To Store Octal Data in python environment, The Octal Data Must be preceded by a letter

'o' or 'O'

=>Syntax: varname=0o Octal data

(OR)

varname=0O Octal data

=>When we store the Octal data in python environment, python Execution Environment converts automatically into decimal number System data.

Examples:

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

>>> a=0o27

>>> print(a,type(a))-------------23 <class 'int'>

>>> oct(a)-----------------------'0o27'

>>> a=0O123

>>> print(a,type(a))------------83 <class 'int'>

>>> oct(83)-----------------'0o123'

>>> a=0o148-------------SyntaxError: invalid digit '8' in octal literal

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

4. Hexa Decimal Number System

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

=>This Number System Conatins

Digits: 0 1 2 3 4 5 6 7 8 9

A(10) B(11) C(12) D(13) E(14) F(15) ------Total Digits =16

Base : 8

=>All Base 16 Literals are called Hexa Decimal Data.

=>To Store Hexa Decimal Data in python environment, The Hexa Decimal Data Must be preceded by a letter ‘x' or 'X'

=>Syntax: varname=0x Hexa Decimal data

(OR)

varname=0X Hexa Decimal data

=>When we store the Hexa Decimal data in python environment,python Execution Environment converts automatically into decimal number System data.

Examples:

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

>>> a=0xAC

>>> print(a,type(a))------------------------172 <class 'int'>

>>> hex(172)--------------------------------'0xac'

>>> a=0xBEE

>>> print(a,type(a))----------------------3054 <class 'int'>

>>> hex(3054)-----------------------------'0xbee'

>>> a=0XFacE

>>> print(a,type(a))--------------------64206 <class 'int'>

>>> a=0xBEER------------------------SyntaxError: invalid hexadecimal literal

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

2. float

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

Properties:

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

=>'float' is one of the pre-defined classes and treated as Fundamental data Type.

=>The purpose of float data type is that " To store Real Constant Values OR Floating Point

Values (Numbers with Decimal Places)".

=>Example: Percentage of Marks, Taxable income for Financial year 22-23.etc

=>float data type can store the data which belongs to Scientific Notation.

=>The Advantage of Scientific Notation is that " It Takes Less Memory Space for Extremly Large Floting Point Values."

=>float data type does not support to store directly the values of Binary, Ocral and Hexa Decimal Number Systems. But it allows to store only Deciaml Number System Values (Default)

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

Examples:

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

>>> a=12.34

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

>>> a=10

>>> b=2.3

>>> c=a+b

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

>>> print(b,type(b))-----------------2.3 <class 'float'>

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

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

>>> a=2e3

>>> print(a,type(a))---------------------------2000.0 <class 'float'>

>>> a=10e-2

>>> print(a,type(a))---------------------0.1 <class 'float'>

>>> a=0.000000000000000000000000000000000000000000000000000001

>>> print(a,type(a))------------------1e-54 <class 'float'>

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

>>> a=0b1010.0b1010----------------SyntaxError: invalid decimal literal

>>> a=0b1010.34----------------------SyntaxError: invalid syntax

>>> a=0xACC.0b1010--------------SyntaxError: invalid decimal literal

>>> a=0o23.0o45------------------SyntaxError: invalid decimal literal

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

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

3. bool

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

Properties

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

=>'bool' is one of the pre-defined classes and treated as Fundamental Data Type.

=>The purpose of bool data type is that "To Store True and False Values".

=>In Python Programming, True and False are of the KeyWords and They are the values for bool data type.

=>Internally, The value of True is 1 and the value of False is 0

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

Examples:

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

>>> a=True

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

>>> b=False

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

>>> a=true-----------------------------NameError: name 'true' is not defined. Did you mean: 'True'?

>>> b=false-----------------NameError: name 'false' is not defined. Did you mean: 'False'?

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

>>> a=True

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

>>> b=False

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

>>> print(a+b)-------------------------1

>>> print(False+False)------------0

>>> print(False-True)----------------1

>>> print(True+True+False)----------2

>>> print(2\*True-4+False) ------------2

>>> print(0b1010+True+1)------------12

>>> print(0b1111\*False+2\*True)------2

>>> print(0b100\*2+3\*True)------------11

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

>>> print (True>False)-------------------True

>>> print (True>1)-------------------------False

>>> print (True>=0b0000001)-----------True

>>> print (False>=0.0)---------------------True

>>> print (True\*False>True)-----------False

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

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

4. complex

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

=>Properties

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

=>'complex' is one of the pre-defined classes and treated as Fundamental Data Type.

=>The purpose of complex data type is that " To Store and Process Complex Data".

=>The General Notation of Complex Data Type is shown below.

a+bj or a-bj

=>here 'a' is called Real Part

=>here 'b' is called Imginary Part

=>Here 'j' represents sqrt(-1)

=>Internally, the real and imginary parts are by default belongs to float.

=>To Extract or get Real Part from Complex object, we use a pre-defined attribute called "real"

Syntax: - Complexobj.real

=>To Extract or get Imaginary Part from Complex object, we use a pre-defined attribute called "imag"

Syntax: - Complexobj.imag

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

Examples:

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

>>> a=2+3j

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

>>> b=4-5j

>>> print(b,type(b))--------------------------(4-5j) <class 'complex'>

>>> c=-2-4j

>>> print(c,type(c))--------------------------(-2-4j) <class 'complex'>

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

>>> a=1.4+3.4j

>>> print(a,type(a))-------------------------(1.4+3.4j) <class 'complex'>

>>> b=-3.5-5.6j

>>> print(b,type(b))-------------------------(-3.5-5.6j) <class 'complex'>

>>> c=10+2.3j

>>> print(c,type(c))-------------------------(10+2.3j) <class 'complex'>

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

>>> a=0+2j

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

>>> b=9.5j

>>> print(b,type(b))-----------------------9.5j <class 'complex'>

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

>>> a=10.4+3j

>>> print(a,type(a))--------------------(10.4+3j) <class 'complex'>

>>> a.real-------------------------10.4

>>> a.imag----------------------3.0

>>> a=9.5j

>>> print(a,type(a))-------------9.5j <class 'complex'>

>>> a.real-----------------------0.0

>>> a.imag-------------------9.5

>>> a.imagiary-----------------AttributeError: 'complex' object has no attribute 'imagiary'

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

>>> a=-3.4-4.5j

>>> print(a,type(a))--------------(-3.4-4.5j) <class 'complex'>

>>> a.real--------------------- -3.4

>>> a.imag------------------ -4.5

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

>>> (12+4j).real---------------------12.0

>>> (12+4j).imag-------------------4.0

>>> (-0-2.3j).real--------------------0.0

>>> (-0-2.3j).imag--------------------2.3

>>> (0b1111+0b1010j).real----------SyntaxError: invalid binary literal

>>> (0b1111+0b1010j).imag---------SyntaxError: invalid binary literal

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

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

II.Sequence Category Data Types

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

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

=>We have 4 data types int Sequence Category . They are

1. str

2. bytes

3. bytearray

4. range

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

1. str (Part-1 )

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

=>"str" is one of the pre-defined classes and treated as Sequence Data Type,

=>The purpose of str data type is that " To store Text Data or Numeric Data or Alpha-numeric Data and special symbols enclosed within Single or Double Quotes or Tripple Single or Tripple Double Quotes."

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

=>Def. of str (String):

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

=>A String is sequnece or Collection of Characters or Numbers or Alpha-numeric Data and special symbols enclosed within Single or Double Quotes or Tripple Single or Tripple Double Quotes.

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

Types of str data

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

=>In Python Programming, we have 2 types of str data. They are

1. Single Line String Data

2. Multi Line String Data

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

1. Single Line String Data

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

=>Syntax: " String Data "

--------------- (OR)

' String data '

=>Single Line string data always enclosed within Double or Single Quotes.

=>Double or Single Quotes are not useful for organizing Multi Line String Data.

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

2. Multi Line String Data

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

=>Syntax: " " " String Line 1

String Line 2

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

String Line n " " "

(OR)

' ' ' String Line 1

String Line 2

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

String Line n ' ' '

=>With Tripple Double Quotes or Tripple Single Quotes we can organize Multi Line String data and also we can organize Single Line String data.

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

Examples:

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

>>> s1="Guido Van Rossum"

>>> print(s1, type(s1))---------------------------Guido Van Rossum <class 'str'>

>>> s2="123456"

>>> print(s2,type(s2))---------------------------123456 <class 'str'>

>>> s2="Python3.10.6"

>>> print(s2,type(s2))---------------------------Python3.10.6 <class 'str'>

>>> s3='Travis Oliphant'

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

>>> s4='1234python%$'

>>> print(s4,type(s4))--------------------------1234python%$ <class 'str'>

>>> s5='A'

>>> print(s5,type(s5))---------------------------A <class 'str'>

>>> s6='6'

>>> print(s6,type(s6))-------------------------6 <class 'str'>

>>> s7='$%^&@'

>>> print(s7,type(s7))---------------------------$%^&@ <class 'str'>

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

>>> s1="Python Programming"

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

>>> s2='Python Programming'

>>> print(s2,type(s2))--------------------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, Red Sea Side

... Python Software Foundation

... Nether Lands

... Pin-57 """

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

Guido Van Rossum

FNO:3-4, Red Sea Side

Python Software Foundation

Nether Lands

Pin-57 <class 'str'>

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

>>> addr2='''Travis Oliphant

... Numpy Organization

... FNO-34-56 Nether lands

... PIN-45 '''

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

Travis Oliphant

Numpy Organization

FNO-34-56 Nether lands

PIN-45 <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="""K"""

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

>>> s2='''K'''

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

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

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

Operations on str data (Part-1)

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

=>On str data, we can perform 2 types of Operations. They are

1. Indexing Operation

2. Slicing Operations

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

1. Indexing Operation

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

=>The Process of Obtaining Single Character from given str object by passing valid Index is called Indexing.

=>Syntax:

--------------- strobj [ Index]

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

=>Index can be either +Ve Indexing or -Ve Indexing

=>If we enter valid Index value then we get Corresponding Charcter from strobj.

=>If we enter invalid Index value then we get IndexError.

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

Examples:

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

>>> s="PYTHON"

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

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

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

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

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

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

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

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

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

>>> print(s[-13])--------------IndexError: string index out of range

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

>>> s="PYTHON"

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

>>> len(s)-----------------6

>>> s="Python Prog"

>>> len(s)------------------ 11

>>> print(s[34])--------------IndexError: string index out of range

-----------------------------------------------------------------------2. Slicing Operations

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

=>The Process of obtaining Range of Characters or Sub String from given Str object is called Slicing.

=>Slicing Operation can be performed by using 5 Syntaxes. -

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

Syntax-1: Strobj[ Begin Index : End Index]

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

This Syntax obtains range of characters from BeginIndex to EndIndex-1 provided Begin Index<End Index Otherwise we never get any output (‘ ')

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

Examples

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

>>> s="PYTHON"

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

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

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

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

>>> print(s[2:5] )----------------------THO

>>> print( s[0:6] )--------------------PYTHON

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

>>> print(s[-6:-3])----------------PYT

>>> print(s[-4:-1])----------------THO

>>> print(s[-1:-6])------------------ Empty

>>> print(s[2:6])---------------------THON

>>> print(s[2:-2])---------------------- TH ( Most Imp )

>>> print(s[1:-1])---------------------- YTHO

>>> print(s[-1:-6])-------------------- empty

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

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

Syntax-2:

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

5 Syntax---------------------Strobj[ Begin:End]

Strobj[Begin: ]

Strobj[ : End]

Strobj[ : ]

Strobj[Begin:End:Step]------6 Rules

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

Operations on str data (Part-1)

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

=>On str data, we can perform 2 types of Operations. They are

1. Indexing Operation

2. Slicing Operations

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

1. Indexing Operation

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

=>The Process of Obtaining Single Character from given str object by passing valid Index is called Indexing.

=>Syntax:

--------------- strobj [ Index]

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

=>Index can be either +Ve Indexing or -Ve Indexing

=>If we enter valid Index value then we get Corresponding Charcter from strobj.

=>If we enter invalid Index value then we get IndexError.

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

Examples:

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

>>> s="PYTHON"

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

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

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

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

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

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

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

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

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

>>> print(s[-13])--------------IndexError: string index out of range

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

>>> s="PYTHON"

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

>>> len(s)-----------------6

>>> s="Python Prog"

>>> len(s)------------------ 11

>>> print(s[34])--------------IndexError: string index out of range

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

2. Slicing Operations

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

=>The Process of obtaining Range of Characters or Sub String from given Str object is called Slicing.

=>Slicing Operation can be performed by using 5 Syntaxes. -

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

Syntax-1: Strobj [ Begin Index: End Index]

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

This Syntax obtains range of characters from BeginIndex to EndIndex-1 provided Begin Index<End Index Otherwise we never get any output (‘ ')

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

Examples

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

>>> s="PYTHON"

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

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

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

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

>>> print( s[2:5] )----------------------THO

>>> print( s[0:6] )--------------------PYTHON

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

>>> print(s[-6:-3])----------------PYT

>>> print(s[-4:-1])----------------THO

>>> print(s[-1:-6])------------------ Empty

>>> print(s[2:6])---------------------THON

>>> print(s[2:-2])---------------------- TH ( Most Imp )

>>> print(s[1:-1])---------------------- YTHO

>>> print(s[-1:-6])-------------------- empty

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

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

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

>>> s[2:-4]----------------------------- ' ' ( Empty String )

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

Syntax-2: StrObj [Begin Index : ]

=>In This Syntax We specified Begin Index and Did't not specify End Index.

=>If we don't Specify End Index then PVM always takes End Character Index as End Index OR len(strobj)-1

Examples:

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

>>> s="PYTHON"

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

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

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

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

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

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

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

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

Syntax-3: StrObj[ : EndIndex ]

=>In This Syntax We specified End Index and Did't not specify Begin Index.

=>If we don't Specify Begin Index then PVM always takes First Character Index as Begin

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

Index.

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

Examples:

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

>>> s="PYTHON"

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

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

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

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

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

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

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

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

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

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

Syntax-4: StrObj[ : ]

=>In This Syntax We Did't not specify Begin Index and End Index.

=>If we don't Specify Begin Index then PVM always takes First Character Index as Begin.

Index and If we don't Specify End Index then PVM always takes Last Character Index as End Index (OR) len(strobj)-1 as End Index.

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

Examples:

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

>>> s="PYTHON"

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

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

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

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

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

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

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

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

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

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

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

Most IMP:

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

>>> s="PYTHON"

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

>>> s[-13:-6]----------------------' '

>>> s[-13:6]-----------------------'PYTHON'

>>> s[0:123]----------------------'PYTHON'

>>> s[-123:345]----------------------'PYTHON'

NOTE: - All the Above Syntaxes are obtaining Range of Characters In Forward Direction.

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

Syntax-5: Strobj[BeginIndex :End Index : Step]

Rules:

1) Here BeginIndex, End Index and Step can either +VE INDEX and -VE INDEX

2) If the value of STEP is +VE then PVM takes the Range of Characters from Begin Index to

End Index-1 in Forward Direction provided Begin Index<End Index otherwise we get empty String (' ‘)

3) if the value of STEP is -VE then PVM Takes Range of Characters from BeginIndex to End

Index+1 in Backward Direction provided Begin Index > End Index

4) When we are retrieving the data in forward Direction if the EndIndex Value is 0 then we never get any result / outout.

5) When we are retrieving the data in backward Direction if the EndIndex Value is -1 then we never get any result / outout.

Examples:

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

>>> s="PYTHON"

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

>>> s [0:6:1]------------------------------'PYTHON'

>>> s[0:6:2]------------------------------'PTO'

>>> s[2:4:1]-----------------------------'TH'

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

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

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

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

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

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

>>> s="PYTHON"

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

>>> s[0:6:2]--------------------------'PTO'

>>> s[0:6:-2]------------------- ' '

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

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

>>> s[-1:-7:-1]--------------------'NOHTYP'

>>> s[-1:-7:-2]-------------------'NHY'

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

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

>>> s="MADAM"

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

>>> s="LIRIL"

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

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

>>> "RACECAR"[::]=="RACECAR"[::][::-1] -----------------True

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

>>> print("KVR"[::3])--------------------K

>>> "KVR"[::3]=="KVR"[::-1][-1]-----------------True

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

>>> s="PYTHON"

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

>>> s="PYTHON PROG"

>>> s[::-1]------------------'GORP NOHTYP'

>>> s="121"

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

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

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

>>> s="PYTHON"

>>> print(s)

PYTHON

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

>>> s[2:0:1]---------------------' ' (Rule-5)

>>> s[1:0:2]--------------------' '

>>> s[-6:-1:-1]--------------- ' ' (Rule-6)

>>> s[-3:-1:-2]-----------------' '

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

>>> s="PYTHON"

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

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

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

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

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

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

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

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

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

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

Type Casting Techniques in Python

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

=>The Process of Converting One Type of Possible Value into Another Type of Value is called

Type Casting.

=>Fundamentally, we have 5 types of Type Casting Techniques. They are

1. int ()

2. float ()

3. bool ()

4. complex ()

5. str ()

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

1. int ()

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

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

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

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

Examples: float into int--->Possible

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

>>> a=12.34

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

>>> b=int(a)

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

>>> a=0.99

>>> print(a,type(a))-----------------------0.99 <class 'float'>

>>> b=int(a)

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

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

Examples: bool into int--->Possible

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

>>> a=True

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

>>> b=int(a)

>>> print(b,type(b))-------------- 1 <class 'int'>

>>> a=False

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

>>> b=int(a)

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

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

Examples:complex into int--->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'

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

Examples:

-----------------------------------------------------------------------ase-1: Str int------->int-----Possible

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

>>> a="123” # str in

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

>>> b=int(a)

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

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

Case-2: Str float---->int--->Not Possible

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

>>> a="12.34" # Str float

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

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

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

Case-3: Str bool------> int--->Not Possible

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

>>> a="True" # str bool

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

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

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

Case-4: str complex--->int---->Not Possible

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

>>> a="2+3j"

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

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

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

Case-5----Pure Str--->int--->Not Possible

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

>>> a="KVR"

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

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

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

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

2. float()

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

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

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

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

Example: int----->float--->Possible

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

>>> a=10

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

>>> b=float(a)

>>> print(b,type(b))---------------10.0 <class 'float'>

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

Example: bool----->float--->Possible

--------------------------------------------------------------------------->> a=True

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

>>> b=float(a)

>>> print(b,type(b))---------------------1.0 <class 'float'>

>>> a=False

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

>>> b=float(a)

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

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

Example: complex----->float--->Not Possible

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

>>> a=2.3+4.5j

>>> print(a,type(a))------------------(2.3+4.5j) <class 'complex'>

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

>>> a=2.3+4.5j

>>> print(a,type(a))--------------(2.3+4.5j) <class 'complex'>

>>> b=float(a.real)

>>> print(b,type(b))--------------2.3 <class 'float'>

>>> b=float(a.imag)

>>> print(b,type(b))-----------4.5 <class 'float'>

---------------------------------------------------------------------------Example:

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

Case-1 str int----->float -->Possible

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

>>> a="12"

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

>>> b=float(a)

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

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

Case-2 str float----->float -->Possible

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

>>> a="12.34"

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

>>> b=float(a)

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

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

Case-3 str bool----->float -->Not Possible

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

>>> a="True"

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

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

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

Case-4 str complex----->float -->Not Possible

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

>>> a="2+3.5j"

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

>>> b=float(a)----------------------------------------ValueError: could not convert string to float: '2+3.5j'

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

Case-5 Pure str ----->float -->Not Possible

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

>>> a="Python.kvr"

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

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

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

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

3. bool()

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

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

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

=>ALL NON-ZERO VALUES ARE TREATED AS TRUE

=>ALL ZERO VALUES ARE TREATED AS FALSE

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

Example: int----->bool---->Possible

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

>>> a=10

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

>>> b=bool(a)

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

>>> a=-123

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

>>> b=bool(a)

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

>>> a=0

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

>>> b=bool(a)

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

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

Example: float----->bool---->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=0.000000000000000000000000000000000000000000000000000000001

>>> print(a,type(a))

1e-57 <class 'float'>

>>> print(b,type(b))

False <class 'bool'>

>>>

>>> a=0.000000000000000000000000000000000000000000000000000000001

>>> print(a,type(a))

1e-57 <class 'float'>

>>> b=bool(a)

>>> print(b,type(b))

True <class 'bool'>

>>> a=0.00000000000000000000000000000000000000000000000000000

>>> print(a,type(a))

0.0 <class 'float'>

>>> b=bool(a)

>>> print(b,type(b))

False <class 'bool'>

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

Example: complex----->bool---->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'>

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

Example: Str int,float,complex,bool and pure str are possible to convert into bool type

Here bool type True provided len(strobj) is >0

Here bool type Frue provided len(strobj) is ==0

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

>>> a="123"

>>> print(a,type(a))

123 <class 'str'>

>>> b=bool(a)

>>> print(b,type(b))

True <class 'bool'>

>>> a="0"

>>> len(a)

1

>>> print(a,type(a))

0 <class 'str'>

>>> b=bool(a)

>>> print(b,type(b))

True <class 'bool'>

>>> a="False"

>>> print(a,type(a))

False <class 'str'>

>>> b=bool(a)

>>> print(b,type(b))

True <class 'bool'>

>>> a="12.34"

>>> print(a,type(a))

12.34 <class 'str'>

>>> b=bool(a)

>>> print(b,type(b))

True <class 'bool'>

>>> a=" "

>>> print(a,type(a))

<class 'str'>

>>> b=bool(a)

>>> print(b,type(b))

True <class 'bool'>

>>> a=""

>>> len(a)

0

>>> print(a,type(a))

<class 'str'>

>>> b=bool(a)

>>> print(b,type(b))

False <class 'bool'>

>>> a="KVR"

>>> print(a,type(a))

KVR <class 'str'>

>>> b=bool(a)

>>> print(b,type(b))

True <class 'bool'>

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

complex ()

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

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

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

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

Examples: int------>complex----->Possible

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

>>> a=12

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

>>> b=complex(a)

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

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

Examples: float------>complex----->Possible

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

>>> a=2.3

>>> print(a,type(a))----------------2.3 <class 'float'>

>>> b=complex(a)

>>> print(b,type(b))-----------------(2.3+0j) <class 'complex'>

----------------------------------------------------------------------s

Examples: bool------>complex----->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'>

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

Examples:

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

>>> a="12” # str int-----complex--Possible

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

>>> b=complex(a)

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

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

>>> a="12.45” #str float------complex---->Possible

>>> print(a,type(a))-----------------12.45 <class 'str'>

>>> b=complex(a)

>>> print(b,type(b))----------------(12.45+0j) <class 'complex'>

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

>>> a="True" # str bool----->complex----Not Possible

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

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

----------------------------------------------------------------------->>> a="KVR-PYTHON" # Pure Str-------->Complex--Not Possible

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

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

-----------------------------------------------------------------------========================================

5. str()

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

=>str () is used converting all types of values into str type value.

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

-----------------------------------------------------------------------Examples:

----------------------------------------------------------------------->>> a=123

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

>>> b=str(a)

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

>>> b---------------------------------------- '123'

>>> 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)'

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

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

2. bytes

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

Properties:

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

=>"bytes" if one of the pre-defined classes and treated as Sequence Data Type.

=>Cipher Text

=>The Internal Implementation of bytes data type is that "End-to-End Encryption (OR) Cipher Text (OR) Encrypted Data" of Normal Text.

=>The bytes data stores the data in the range of 0 to 256 (It stores from 0 to 255 (256-1) only)

=>bytes data type does not contains any symbolic notation but we can convert other type of values into bytes type values by using bytes().

=>Syntax: varname=bytes(object)

=>An object of bytes belongs to Immutable bcoz bytes' object does not support item assignment

=>An object of bytes data type supports Both Indexing and Slicing Operations.

=>An object of bytes maintains Insertion Order (i.e Which ever order we insert the data in the same order Value will display)

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

Examples:

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

>>> l1= [10,20,30,40,256]

>>> print (l1, type(l1))------------------------[10, 20, 30, 40, 256] <class 'list'>

>>> b=bytes(l1) -------------------ValueError: bytes must be in range(0, 256)

>>> l1=[10,0,-20,30,40,255]

>>> print (l1, type(l1))-----------------[10, 0, -20, 30, 40, 255] <class 'list'>

>>> b=bytes(l1) --------------------ValueError: bytes must be in range(0, 256)

>>> l1=[10,0,30,40,255]

>>> print(l1,type(l1))----------------------[10, 0, 30, 40, 255] <class 'list'>

>>> b=bytes(l1)

>>> print (b, type(b),id(b))-------------b'\n\x00\x1e(\xff' <class 'bytes'> 2043775384912

>>> b[-1]--------------------------255

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

>>> b[0]=123-----------------TypeError: 'bytes' object does not support item assignment

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

>>> l1= [10,0,30,40,255]

>>> print (l1, type(l1))----------------------[10, 0, 30, 40, 255] <class 'list'>

>>> b=bytes(l1)

>>> print (b, type(b), id(b))-----------b'\n\x00\x1e(\xff' <class 'bytes'> 2043775382752

>>> for kvr in b:

... print(kvr)

...

10

0

30

40

255

>>> t1=(10,20,30,10,255,45)

>>> print (t1, type(t1))---------------------(10, 20, 30, 10, 255, 45) <class 'tuple'>

>>> b=bytes(t1)

>>> print(b,type(b),id(b))-------------b'\n\x14\x1e\n\xff-' <class 'bytes'> 2043775382800

>>> for v in b:

... print(v)

...

10

20

30

10

255

45

>>> b [0:4] --------------------b'\n\x14\x1e\n'

>>> for v in b [0:4]:

... print(v)

...

10

20

30

10

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

... print(v)

...

45

255

10

30

20

10

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

Mutable and Immutable

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

=>A Mutable object is one, whose content can be changed at Same Memory Address.

=>Examples: list, bytearray,set,dict

=>An immutable object is one, which will satisfy the following Properties

a) The value immutable object can't be changed at Same Memory Address (OR) In otherwards, Value of Immutable object can be changed and place the modified Value in New Memory Address by eliminating Old Memory Address by Garbage Collector.

b) Immutable objects does not support Item Assigmnent.

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

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

3. bytearray

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

Properties:

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

=>"bytearray" if one of the pre-defined classes and treated as Sequence Data Type.

=>The Internal Implementation of bytearray data type is that "End-to-End Encryption (OR) Cipher Text (OR) Encrypted Data" of Normal Text.

=>The bytearray data stores the data in the range of 0 to 256 (It stores from 0 to 255 (256-1) only)

=>bytearray data type does not contains any symbolic notation but we can convert other type of values into bytearray type values by using bytearray().

=>Syntax: varname=bytearray(object)

=>An object of bytearray belongs to Mutable bcoz bytearray object supports item assignment

=>An object of bytearray data type supports Both Indexing and Slicing Operations.

=>An object of bytearray maintains Insertion Order (i.e Which ever order we insert the data in the same order Value will display)

=>NOTE: - The Functionality of bytearray is exactly similar to bytes but an object of bytes belongs to immutable where as an object of bytearray is mutable.

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

Examples:

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

>>> l1=[10,20,30,40,0,256]

>>> print(l1,type(l1))------------------------------[10, 20, 30, 40, 0, 256] <class 'list'>

>>> b=bytearray(l1) --------------------ValueError: byte must be in range(0, 256)

>>> l1=[10,-20,30,40,0,255]

>>> print(l1, type(l1))------------------[10, -20, 30, 40, 0, 255] <class 'list'>

>>> b=bytearray(l1) ----------------------ValueError: byte must be in range(0, 256)

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

>>> l1=[10,20,30,40,0,255]

>>> print(l1,type(l1))-----------------[10, 20, 30, 40, 0, 255] <class 'list'>

>>> b=bytearray(l1)

>>> print(b,type(b),id(b))----bytearray(b'\n\x14\x1e(\x00\xff') <class 'bytearray'>

2376795361136

>>> for k in b:

... print(k)

...

10

20

30

40

0

255

>>> b[0]=120 # Item Assigment---Possible--Mutable

>>> for k in b:

... print(k)

...

120

20

30

40

0

255

>>> print(b,type(b),id(b))--bytearray(b'x\x14\x1e(\x00\xff') <class 'bytearray'> 2376795361136

>>> for k in b:

... print(k)

...

120

20

30

40

0

255

>>> b[1]--------------------------20

>>> b[1:4] ----------------------bytearray(b'\x14\x1e(')

>>> for k in b[1:4]:

... print(k)

...

20

30

40

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

... print(k)

...

255

0

40

30

20

120

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

4. range

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

Properties

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

=>"range" is one of the pre-defined classes and treated as Sequece Data Type

=>The purpose of range data type is that "To store or generate Sequence of Numerical Integer Values by maintaining Equal Interval of Value."

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

=>An object of range belongs to immutable.

=>An object of range maintains Insertion Order.

=>To store or generate Sequence of Numerical Integer Values by maintaining Equal Interval of Value, range data type provides 3 Syntaxes. They are.

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

=>Syntax-1: varname=range (Value)

=>This Syntax generates Range of Values from 0 to Value-1

Examples:

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

>>> r=range (10)

>>> print(r,type(r))----------------range(0, 10) <class 'range'>

>>> for v in r:

... print(v)

...

0

1

2

3

4

5

6

7

8

9

>>> for k in range(6):

... print(k)

...

0

1

2

3

4

5

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

=>Syntax-2: varname=range (Begin , End )

=>This generates Range of Values from Begin to End-1

Examples:

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

>>> r=range (10,16)

>>> print(r,type(r))------------range(10, 16) <class 'range'>

>>> for v in r:

... print(v)

...

10

11

12

13

14

15

>>> for k in range(6):

... print(k)

...

0

1

2

3

4

5

=>NOTE: In the above Two Syntaxes, the default STEP is 1

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

=>Syntax-3: varname=range (Begin, End, Step)

=>This generates Range of Values from Begin to End-1 by maintaining Step as Equal Interval.

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

Examples:

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

>>> r=range(10,21,3)

>>> print(r,type(r))---------------------range(10, 21, 3) <class 'range'>

>>> for v in r:

... print(v)

...

10

13

16

19

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

... print(v)

...

2

4

6

8

10

12

14

16

18

20

>>> for v in range (1,21,2):

... print(v)

...

1

3

5

7

9

11

13

15

17

19

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

Programming Examples:

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

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

>>> for v in range (10):

... print(v)

...

0

1

2

3

4

5

6

7

8

9

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

Q2) 10 11 12 13 14 15---range(10,16)

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

... print(v)

...

10

11

12

13

14

15

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

Q3) 300 301 302 303 304 305---range(300,306)

>>> for v in range(300,306):

... print(v)

...

300

301

302

303

304

305

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

Q4) 10 9 8 7 6 5 4 3 2 1-----range(10,0,-1)

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

>>> for v in range (1

0,0,-1):

... print(v)

...

10

9

8

7

6

5

4

3

2

1

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

Q5) -10 -11 -12 -13 -14 -15------range(-10,-16,-1)

>>> for v in range(-10,-16,-1):

... print(v)

...

-10

-11

-12

-13

-14

-15

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

Q6) 100 110 120 130 140 150--range(100,151,10)

>>> for k in range (100,151,10):

... print(k)

...

100

110

120

130

140

150

>>>

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

Q7) 1000 900 800 700 600 500-----range(1000,499,-100)

>>> for v in range(1000,499,-100):

... print(v)

...

1000

900

800

700

600

500

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

Q8) -5 -4 -3 - 2 -1 0 1 2 3 4 5----range(-5,6)

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

... print(v)

...

-5

-4

-3

-2

-1

0

1

2

3

4

5

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

... print(v)

...

-5

-4

-3

-2

-1

0

1

2

3

4

5

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

>>> r=range(500,601,50)

>>> r[0]

500

>>> r[1]

550

>>> r[-1]

600

>>> r[2]

600

>>> r=range(500,601,10)

>>> r[-1]

600

>>> for v in r:

... print(v)

...

500

510

520

530

540

550

560

570

580

590

600

>>> for v in r [5:]:

... print(v)

...

550

560

570

580

590

600

>>> for v in r [5:][::-1]:

... print(v)

...

600

590

580

570

560

550

-----------------------------------------------------------------------> r=range(500,601,10)

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

range (500, 601, 10) <class 'range'>

>>> r[0]

500

>>> r[1]

510

>>> r[2]

520

>>> r[1]=700-------------------TypeError: 'range' object does not support item assignment

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

>>> print(range(50,60)[5])------------------55

>>> for v in range(50,60)[5:7]:

... print(v)

...

55

56

>>> for v in range(50,60)[::-2]:

... print(v)

...

59

57

55

53

51

List Category Data Types (Collections Data Types or Data Structures)

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

=>The purpose of List Category Data Types in python is that " To Store Multiple Values either of Same Type or Different Type or Both the Types with Unique and Duplicate in single object."

=>We have two data type in List Category. They are

1. list (Mutable)

2. tuple (Immutable)

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

list

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

Index

---------

=>Purpose of list

=>Operations on list

1) Indexing

2) slicing

=>Pre-Defined Functions in list

1) append ()

2) insert ()

3) remove ()

4) pop(index)

5) pop ()

Note: del operator

6)count()

7)index()

8)reverse()

9) sort ()

10)extend()

11)copy()---- Shallow and Deep copy

=>Inner List / Nested List

=>Pre-defined Function in inner / nested list

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

Properties of list

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

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

=>The purpose of list data type is that "To Store Multiple Values either of Same Type or Different Type or Both the Types with Unique and Duplicate in single object.

=>The Elements of list must write or organized or stored within Square Brackets and the elements separated by Comma.

=>An object of list maintains Insertion Order.

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

=>An object of list belongs to Mutable bcoz it allows us to update the values of list at same address.

=>We can convert any type of value into list type value by using list ()

Syntax: listobj=list(object)

=>by using list data type, we can create two types of list objects. They are

1) empty list

2) non-empty list

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

1) empty list

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

Syntax: varname=[]

(OR)

varname=list()

=>An empty list is one, which does not contain any elements and whose length=0

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

2) non-empty list

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

Syntax: varname=[Val1,Val2...Val-n]

=>A non-empty list is one, which contains elements and whose length>0

-----------------------------------------------------------------------Examples:

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

>>> l1=[10,20,30,10,40]

>>> print(l1,type(l1))------------------------------[10, 20, 30, 10, 40] <class 'list'>

>>> l1=[111,"Rossum",34.56,True,"Python"]

>>> print(l1,type(l1))-----------------------------[111, 'Rossum', 34.56, True, 'Python'] <class 'list'>

>>> l1[0] --------------------------------------------111

>>> l1[-1] -------------------------------------------'Python'

>>> l1[0:3] -----------------------------------------[111, 'Rossum', 34.56]

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

>>> print(l1, type(l1))-------------------------[111, 'Rossum', 34.56, True, 'Python'] <class 'list'>

>>> print(l1, type(l1),id(l1))----[111, 'Rossum', 34.56, True, 'Python'] <class 'list'> 2902431303872

>>> l1[0] =222

>>> print(l1, type(l1),id(l1))---[222, 'Rossum', 34.56, True, 'Python'] <class 'list'> 2902431303872

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

>>> l1=[]

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

>>> len(l1) ------------------------------0

>>> l2=list()

>>> print(l2,type(l2))-------------------[] <class 'list'>

>>> len(l2) --------------------------------0

>>> l3=[10,"Rossum","PSF",3.4,True]

>>> print(l3, type(l3))--------------------[10, 'Rossum', 'PSF', 3.4, True] <class 'list'>

>>> len(l3) -----------------------------5

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

>>> s="PYTHON"

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

>>> l1=list(s)

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

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

>>> l1= [10,20,30,40]

>>> b=bytes(l1)

>>> print(b,type(b))-----------------------b'\n\x14\x1e(' <class 'bytes'>

>>> l2=list(b)

>>> print(l2, type(l2))--------------------[10, 20, 30, 40] <class 'list'>

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

>>> l1=[10,20,30,40]

>>> l1[2] -------------------------30

>>> l1[-2] -------------------------30

>>> l1[::2]-----------------------[10, 30]

>>> l1[::-1]-----------------------[40, 30, 20, 10]

>>> l1[::]-------------------------[10, 20, 30, 40]

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

Pre-Defined Functions in list

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

=>Along with the operations on list like Indexing and Slicing, we can perform many more operations by using pre-defined function of list object.

=>The pre-defined functions of list are given bellow.

-----------------------------------------------------------------------1) append ():

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

=>Syntax: listobj.append(Value)

=>This Function is used for adding Value at the end of existing elements of list (known as appending )

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

Examples:

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

>>> l1=[10,"Rossum"]

>>> print(l1,type(l1),id(l1))------------------------[10, 'Rossum'] <class 'list'> 2902435500480

>>> len(l1) --------------2

>>> l1.append(23.45)

>>> print(l1,type(l1),id(l1))----[10, 'Rossum', 23.45] <class 'list'> 2902435500480

>>> l1. append("KVR")

>>> print(l1,type(l1),id(l1))---[10, 'Rossum', 23.45, 'KVR'] <class 'list'> 2902435500480

>>> l1.append(True)

>>> l1.append(2+3.5j)

>>> print(l1,type(l1),id(l1))---[10, 'Rossum', 23.45, 'KVR', True, (2+3.5j)] <class 'list'> 2902435500480

>>> len(l1) ----6

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

>>> l1=[]

>>> print(l1,len(l1), id(l1))------------------[] 0 2902435500544

>>> l1.append(10)

>>> l1.append("Raj")

>>> l1.append(10.34)

>>> l1.append("Hyd")

>>> print(l1,len(l1), id(l1))----------------[10, 'Raj', 10.34, 'Hyd'] 4 2902435500544

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

2) insert()

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

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

=>Here Index can be either +Ve or -Ve

=>Value can be any type.

=>This Function is used for Inserting the Specific Value at specified index.

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

Examples:

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

>>> l1=[10,20,30,"Python","DJango",34.56]

>>> print(l1,id(l1))--------------------[10, 20, 30, 'Python', 'DJango', 34.56] 2902431529344

>>> l1. insert(3,"Rossum")

>>> print(l1, id(l1))-----[10, 20, 30, 'Rossum', 'Python', 'DJango', 34.56] 2902431529344

>>> l1[-3] ="PYTH"

>>> print(l1,id(l1))----[10, 20, 30, 'Rossum', 'PYTH', 'DJango', 34.56] 2902431529344

>>> l1.insert(1,234.99)

>>> print(l1,id(l1))----[10, 234.99, 20, 30, 'Rossum', 'PYTH', 'Django', 34.56] 2902431529344

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

>>> l1=list()

>>> print(l1,id(l1))---------------[] 2902435501056

>>> l1.insert(0,"KVR")

>>> print(l1,id(l1))---------------['KVR'] 2902435501056

>>> l1.insert(0,1111)

>>> print(l1,id(l1))----------------[1111, 'KVR'] 2902435501056

>>> l1.insert(2,"HYD")

>>> print(l1,id(l1))--------------[1111, 'KVR', 'HYD'] 2902435501056

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

>>> l1=[10,20,30]

>>> print(l1,id(l1))

[10, 20, 30] 2902435496128

>>> l1.append("Python")

>>> print(l1,id(l1))

[10, 20, 30, 'Python'] 2902435496128

>>> l1.insert(30,"Rossum") # Most IMP

>>> print(l1,id(l1))----------[10, 20, 30, 'Python', 'Rossum'] 2902435496128

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

3) remove() Based on Value

-----------------------------------------------------------------------=>Syntax: listobj.remove(Value)

=>This Function is used for removing First Occurence of The specific value from list object.

=>If the specific value does not exist in list object then we get ValueError

Examples:

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

>>> l1=[10,20,30,10,40,50,60]

>>> print(l1,id(l1))-------------[10, 20, 30, 10, 40, 50, 60] 2902431529344

>>> l1.remove(20)

>>> print(l1,id(l1))---------------[10, 30, 10, 40, 50, 60] 2902431529344

>>> l1.remove(10)

>>> print(l1,id(l1))-----------[30, 10, 40, 50, 60] 2902431529344

>>> l1.remove(50)

>>> print(l1,id(l1))-------------[30, 10, 40, 60] 2902431529344

>>> l1.remove(100)---------ValueError: list.remove(x): x not in list

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

>>> l1=[]

>>> l1.remove(3)--------------ValueError: list.remove(x): x not in list

>>> list().remove(100)------ValueError: list.remove(x): x not in list

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

4) pop(index): Based Index

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

Syntax: listobj.pop(Index)

=>This Function is used for removing the element of listobj based Index.

=>If index value is invalid then we get IndexError

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

Examples:

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

>>> l1=[10,20,10,30,40,50,60,30]

>>> print(l1, id(l1))-------------[10, 20, 10, 30, 40, 50, 60, 30] 2902435496128

>>> l1.pop(2)-----------10

>>> print(l1,id(l1))----------[10, 20, 30, 40, 50, 60, 30] 2902435496128

>>> l1.pop(-1)---------------30

>>> print(l1,id(l1))-------------[10, 20, 30, 40, 50, 60] 2902435496128

>>> l1.pop(2)---------------30

>>> print(l1,id(l1))------------[10, 20, 40, 50, 60] 2902435496128

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

>>> list(). pop(4)--------------IndexError: pop from empty list

>>> [].pop(3)-----------------IndexError: pop from empty list

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

5) pop() :

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

=>Syntax:- list.pop()

=>This Function is used for Removing Last Element of List object

=>When we call pop() on empty list then we get IndexError

Examples:

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

>>> lst=[10,"Rossum",45.67,True,2+3j]

>>> print(lst,type(lst))-------------------[10, 'Rossum', 45.67, True, (2+3j)] <class 'list'>

>>> lst.pop()----------(2+3j)

>>> print(lst,type(lst))----------[10, 'Rossum', 45.67, True] <class 'list'>

>>> lst.pop()------------True

>>> print(lst,type(lst))-------------[10, 'Rossum', 45.67] <class 'list'>

>>> lst.pop()----------45.67

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

>>> lst.pop()-----------'Rossum'

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

>>> lst.pop()---------------10

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

>>> lst.pop()----------------IndexError: pop from empty list

>>> list().pop()-------------IndexError: pop from empty list

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

>>> lst=[10,20,30,40,50]

>>> print(lst)----------------[10, 20, 30, 40, 50]

>>> lst.insert(2,300)

>>> print(lst)------------------[10, 20, 300, 30, 40, 50]

>>> lst.pop()----------------50

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

NOTE: del operator

=>del operator is used for deleting Elements of any mutable object either based on Index or Based on Slicing or Total Object.

=>Syntax1: del object [Index]

del object [Begin:End:Step]

del object

=>With "del" operator we can't delete Immutable Content But we can delete complete Immutable Object.

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

Examples:

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

>>> lst=[10,"Rossum",45.67, True,2+3j,"Python"]

>>> print(lst)--------------------[10, 'Rossum', 45.67, True, (2+3j), 'Python']

>>> del lst[3] # Deleting Based on Index

>>> print(lst)-------------------[10, 'Rossum', 45.67, (2+3j), 'Python']

>>> del lst[2:4] # Deleting Based on Slicing

>>> print(lst)-----------------------[10, 'Rossum', 'Python']

>>> del lst # Deleting Entire Object

>>> print(lst)-----------------NameError: name 'lst' is not defined. Did you mean: 'list'?

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

>>> s="PYTHON"

>>> print(s,type(s),id(s))----------------PYTHON <class 'str'> 2073554063472

>>> s=s+"Prog"

>>> print(s,type(s),id(s))-----------------PYTHONProg <class 'str'> 2073554063280

>>> del s[0]------------------------TypeError: 'str' object doesn't support item deletion

>>> del s[0:3]--------------------TypeError: 'str' object does not support item deletion

>>> del s # Deleting Immutable object

>>> s-------------------------NameError: name 's' is not defined

>>>

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

6) copy ()

-----------------------------------------------------------------------=>Syntax: object2=object1.copy()

=>This Function is used for Copying the content of one object into another object (Implementation of Sallow Copy )

Example:

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

Examples:

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

>>> l1=[10,"Rossum"]

>>> print(l1,id(l1))---------------------[10, 'Rossum'] 2073549864512

>>> l2=l1.copy() # Shallow Copy

>>> print(l2,id(l2))--------------------[10, 'Rossum'] 2073554063744

>>> l1.append("Python")

>>> l1.append("Python")

>>> l2.insert(1,"PSF")

>>> print(l1,id(l1))----------------[10, 'Rossum', 'Python', 'Python'] 2073549864512

>>> print(l2,id(l2))----------------[10, 'PSF', 'Rossum'] 2073554063744

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

Examples:----Deep Copy

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

>>> l1=[10,"Rossum"]

>>> print(l1,id(l1))-----------------------[10, 'Rossum'] 2073554059392

>>> l2=l1 # Deep Copy

>>> print(l2,id(l2))------------------------[10, 'Rossum'] 2073554059392

>>> l1. append("Python")

>>> print(l1,id(l1))------------------------[10, 'Rossum', 'Python'] 2073554059392

>>> print(l2,id(l2))------------------------[10, 'Rossum', 'Python'] 2073554059392

>>> l2. insert(2,"PSF")

>>> print(l1,id(l1))-----------------------[10, 'Rossum', 'PSF', 'Python'] 2073554059392

>>> print(l2,id(l2))-----------------------[10, 'Rossum', 'PSF', 'Python'] 2073554059392

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

NOTE:- Slice Based Copy

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

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

>>> print(lst1,id(lst1))---------------[10, 20, 30, 40, 50, 60] 2073692289216

>>> lst2=lst1[0:3] # Slice Based Copy

>>> print(lst2,id(lst2))---------------------[10, 20, 30] 2073692289792

>>> lst2.append(12.34)

>>> lst1.append(70)

>>> print(lst1,id(lst1))-------------------[10, 20, 30, 40, 50, 60, 70] 2073692289216

>>> print(lst2,id(lst2))------------------[10, 20, 30, 12.34] 2073692289792

>>>

>>> lst3=lst1[::] # Slice Based Copy

>>> print(lst3,id(lst3))-----------------[10, 20, 30, 40, 50, 60, 70] 2073686948288

>>> lst3.insert(1,"KVR")

>>> lst1.append(80)

>>> print(lst1,id(lst1))---------------[10, 20, 30, 40, 50, 60, 70, 80] 2073692289216

>>> print(lst3,id(lst3))---------------[10, 'KVR', 20, 30, 40, 50, 60, 70] 2073686948288

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

7) count ():

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

Syntax: - listobj.count(Value)

=>This Function is used for Counting Number of Occurences of a Specified Element.

=>If the Specified Element does not exist in list object, then we get 0

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

Examples:

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

>>> lst= [10,20,30,40,10,20,10,60]

>>> print(lst)

[10, 20, 30, 40, 10, 20, 10, 60]

>>> lst.count(10)-----------------3

>>> len(lst)----------------------8

>>> lst.count(20)---------------2

>>> lst.count(30)----------------1

>>> lst.count(300)--------------0

>>> lst.count("H")-------------0

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

>>> list (). count (10) ---------------0

>>> []. count("")-------------------0

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

7) index()

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

=>Syntax:- listobj.index(Value)

=>This Function is used for finding Index of First Occurence of Specified Element.

=>If the Specified Element not existing in list object then we get ValueError.

Examples:

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

>>> lst=[10,20,30,10,60,70,80,20,45]

>>> print(lst)------------------[10, 20, 30, 10, 60, 70, 80, 20, 45]

>>> lst.index(10)-------------0

>>> lst.index(20)----------1

>>> lst.index(60)------------4

>>> lst.index(45)-----------8

>>> lst.index(145)--------------ValueError: 145 is not in list

>>> list().index("KVR")---------ValueError: 'KVR' is not in list

>>> [10,20,30].index(10)---------0

>>> [10,20,30].index(100)--------ValueError: 100 is not in list

>>> [10,20,30].index("10")-----------ValueError: '10' is not in list

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

8) reverse ():--Tomorrow

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

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) Both the Objects Memory Address are Different

c) Modifications are Independent ( Whatever the modifications we do on any one object they are not reflecting another object)

=>To Implement Shallow Copy, we use copy().

=>Syntax: object2=object1.copy()

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

Examples:

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

>>> l1=[10,"Rossum"]

>>> print(l1,id(l1))---------------------[10, 'Rossum'] 2073549864512

>>> l2=l1.copy() # Shallow Copy

>>> print(l2,id(l2))--------------------[10, 'Rossum'] 2073554063744

>>> l1. append("Python")

>>> l1. append("Python")

>>> l2. insert(1,"PSF")

>>> print(l1,id(l1))----------------[10, 'Rossum', 'Python', 'Python'] 2073549864512

>>> print(l2,id(l2))----------------[10, 'PSF', 'Rossum'] 2073554063744

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

2. Deep Copy

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

=>The Properties of Deep Copy are

a) Initial Content of Both the Objects are Same.

b) Both the Objects Memory Address are Same

c) Modifications are Depedent ( Whatever the modifications we do on any one object they are reflecting to another object)

=>To Implement Deep Copy, we use Assigment Operator ( = )

=>Syntax: object2 = object1

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

Examples:

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

>>> l1=[10,"Rossum"]

>>> print(l1,id(l1))-----------------------[10, 'Rossum'] 2073554059392

>>> l2=l1 # Deep Copy

>>> print(l2,id(l2))------------------------[10, 'Rossum'] 2073554059392

>>> l1.append("Python")

>>> print(l1,id(l1))------------------------[10, 'Rossum', 'Python'] 2073554059392

>>> print(l2,id(l2))------------------------[10, 'Rossum', 'Python'] 2073554059392

>>> l2.insert(2,"PSF")

>>> print(l1,id(l1))-----------------------[10, 'Rossum', 'PSF', 'Python'] 2073554059392

>>> print(l2,id(l2))-----------------------[10, 'Rossum', 'PSF', 'Python'] 2073554059392

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

Pre-Defined Functions in list

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

=>Along with the operations on list like Indexing and Slicing, we can perform many more operations by using pre-defined function of list object.

=>The pre-defined functions of list are given bellow.

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

1) append ():

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

=>Syntax: listobj.append(Value)

=>This Function is used for adding Value at the end of existing elements of list(known as appending )

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

Examples:

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

>>> l1=[10,"Rossum"]

>>> print(l1,type(l1),id(l1))------------------------[10, 'Rossum'] <class 'list'> 2902435500480

>>> len(l1)--------------2

>>> l1.append(23.45)

>>> print(l1,type(l1),id(l1))----[10, 'Rossum', 23.45] <class 'list'> 2902435500480

>>> l1.append("KVR")

>>> print(l1,type(l1),id(l1))---[10, 'Rossum', 23.45, 'KVR'] <class 'list'> 2902435500480

>>> l1.append(True)

>>> l1.append(2+3.5j)

>>> print(l1,type(l1),id(l1))---[10, 'Rossum', 23.45, 'KVR', True, (2+3.5j)] <class 'list'> 2902435500480

>>> len(l1)----6

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

>>> l1=[]

>>> print(l1,len(l1), id(l1))------------------[] 0 2902435500544

>>> l1.append(10)

>>> l1.append("Raj")

>>> l1.append(10.34)

>>> l1.append("Hyd")

>>> print(l1,len(l1), id(l1))----------------[10, 'Raj', 10.34, 'Hyd'] 4 2902435500544

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

2) insert()

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

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

=>Here Index can be either +Ve or -Ve

=>Value can be any type.

=>This Function is used for Inserting the Specific Value at specified index.

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

Examples:

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

>>> l1=[10,20,30,"Python","DJango",34.56]

>>> print(l1,id(l1))--------------------[10, 20, 30, 'Python', 'DJango', 34.56] 2902431529344

>>> l1.insert(3,"Rossum")

>>> print(l1,id(l1))-----[10, 20, 30, 'Rossum', 'Python', 'DJango', 34.56] 2902431529344

>>> l1[-3]="PYTH"

>>> print(l1,id(l1))----[10, 20, 30, 'Rossum', 'PYTH', 'DJango', 34.56] 2902431529344

>>> l1.insert(1,234.99)

>>> print(l1,id(l1))----[10, 234.99, 20, 30, 'Rossum', 'PYTH', 'DJango', 34.56] 2902431529344

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

>>> l1=list()

>>> print(l1,id(l1))---------------[] 2902435501056

>>> l1.insert(0,"KVR")

>>> print(l1,id(l1))---------------['KVR'] 2902435501056

>>> l1.insert(0,1111)

>>> print(l1,id(l1))----------------[1111, 'KVR'] 2902435501056

>>> l1.insert(2,"HYD")

>>> print(l1,id(l1))--------------[1111, 'KVR', 'HYD'] 2902435501056

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

>>> l1=[10,20,30]

>>> print(l1,id(l1))

[10, 20, 30] 2902435496128

>>> l1.append("Python")

>>> print(l1,id(l1))

[10, 20, 30, 'Python'] 2902435496128

>>> l1.insert(30,"Rossum") # Most IMP

>>> print(l1,id(l1))----------[10, 20, 30, 'Python', 'Rossum'] 2902435496128

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

3) remove() Based on Value

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

=>Syntax: listobj.remove(Value)

=>This Function is used for removing First Occurence of The specific value from list object.

=>If the specific value does not exist in list object then we get ValueError

Examples:

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

>>> l1=[10,20,30,10,40,50,60]

>>> print(l1,id(l1))-------------[10, 20, 30, 10, 40, 50, 60] 2902431529344

>>> l1.remove(20)

>>> print(l1,id(l1))---------------[10, 30, 10, 40, 50, 60] 2902431529344

>>> l1. remove(10)

>>> print(l1,id(l1))-----------[30, 10, 40, 50, 60] 2902431529344

>>> l1. remove(50)

>>> print(l1,id(l1))-------------[30, 10, 40, 60] 2902431529344

>>> l1.remove(100)---------ValueError: list.remove(x): x not in list

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

>>> l1=[]

>>> l1.remove(3)--------------ValueError: list.remove(x): x not in list

>>> list().remove(100)------ValueError: list.remove(x): x not in list

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

4) pop(index): Based Index

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

Syntax: listobj.pop(Index)

=>This Function is used for removing the element of listobj based Index.

=>If index value is invalid then we get IndexError

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

Examples:

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

>>> l1=[10,20,10,30,40,50,60,30]

>>> print(l1,id(l1))-------------[10, 20, 10, 30, 40, 50, 60, 30] 2902435496128

>>> l1.pop(2)-----------10

>>> print(l1,id(l1))----------[10, 20, 30, 40, 50, 60, 30] 2902435496128

>>> l1.pop(-1)---------------30

>>> print(l1,id(l1))-------------[10, 20, 30, 40, 50, 60] 2902435496128

>>> l1.pop(2)---------------30

>>> print(l1,id(l1))------------[10, 20, 40, 50, 60] 2902435496128

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

>>> list().pop(4)--------------IndexError: pop from empty list

>>> [].pop(3)-----------------IndexError: pop from empty list

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

5) pop() :

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

=>Syntax:- list.pop()

=>This Function is used for Removing Last Element of List object

=>When we call pop() on empty list then we get IndexError

Examples:

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

>>> lst=[10,"Rossum",45.67,True,2+3j]

>>> print(lst,type(lst))-------------------[10, 'Rossum', 45.67, True, (2+3j)] <class 'list'>

>>> lst.pop() ----------(2+3j)

>>> print(lst,type(lst))----------[10, 'Rossum', 45.67, True] <class 'list'>

>>> lst.pop()------------True

>>> print(lst,type(lst))-------------[10, 'Rossum', 45.67] <class 'list'>

>>> lst.pop()----------45.67

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

>>> lst.pop()-----------'Rossum'

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

>>> lst.pop()---------------10

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

>>> lst.pop()----------------IndexError: pop from empty list

>>> list().pop()-------------IndexError: pop from empty list

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

>>> lst=[10,20,30,40,50]

>>> print(lst)----------------[10, 20, 30, 40, 50]

>>> lst.insert(2,300)

>>> print(lst)------------------[10, 20, 300, 30, 40, 50]

>>> lst.pop()----------------50

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

NOTE: del operator

=>del operator is used for deleting Elements of any mutable object either based on Index or Based on Slicing or Total Object.

=>Syntax1: del object[Index]

del object[Begin:End:Step]

del object

=>With "del" operator we can't delete Immutable Content But we can delete complete Immutable Object.

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

Examples:

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

>>> lst=[10,"Rossum",45.67, True,2+3j,"Python"]

>>> print(lst)--------------------[10, 'Rossum', 45.67, True, (2+3j), 'Python']

>>> del lst[3] # Deleting Based on Index

>>> print(lst)-------------------[10, 'Rossum', 45.67, (2+3j), 'Python']

>>> del lst[2:4] # Deleting Based on Slicing

>>> print(lst)-----------------------[10, 'Rossum', 'Python']

>>> del lst # Deleting Entire Object

>>> print(lst)-----------------NameError: name 'lst' is not defined. Did you mean: 'list'?

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

>>> s="PYTHON"

>>> print(s,type(s),id(s))----------------PYTHON <class 'str'> 2073554063472

>>> s=s+"Prog"

>>> print(s,type(s),id(s))-----------------PYTHONProg <class 'str'> 2073554063280

>>> del s[0]------------------------TypeError: 'str' object doesn't support item deletion

>>> del s[0:3]--------------------TypeError: 'str' object does not support item deletion

>>> del s # Deleting Immutable object

>>> s-------------------------NameError: name 's' is not defined

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

6) copy()

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

=>Syntax: object2=object1.copy()

=>This Function is used for Copying the content of one object into another object (Implementation of Sallow Copy )

Example:

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

Examples:

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

>>> l1=[10,"Rossum"]

>>> print(l1, id(l1))---------------------[10, 'Rossum'] 2073549864512

>>> l2=l1.copy() # Shallow Copy

>>> print(l2,id(l2))--------------------[10, 'Rossum'] 2073554063744

>>> l1.append("Python")

>>> l1.append("Python")

>>> l2.insert(1,"PSF")

>>> print(l1,id(l1))----------------[10, 'Rossum', 'Python', 'Python'] 2073549864512

>>> print(l2,id(l2))----------------[10, 'PSF', 'Rossum'] 2073554063744

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

Examples:----Deep Copy

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

>>> l1=[10,"Rossum"]

>>> print(l1,id(l1))-----------------------[10, 'Rossum'] 2073554059392

>>> l2=l1 # Deep Copy

>>> print(l2,id(l2))------------------------[10, 'Rossum'] 2073554059392

>>> l1.append("Python")

>>> print(l1,id(l1))------------------------[10, 'Rossum', 'Python'] 2073554059392

>>> print(l2,id(l2))------------------------[10, 'Rossum', 'Python'] 2073554059392

>>> l2.insert(2,"PSF")

>>> print(l1,id(l1))-----------------------[10, 'Rossum', 'PSF', 'Python'] 2073554059392

>>> print(l2,id(l2))-----------------------[10, 'Rossum', 'PSF', 'Python'] 2073554059392

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

NOTE:- Slice Based Copy

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

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

>>> print(lst1,id(lst1))---------------[10, 20, 30, 40, 50, 60] 2073692289216

>>> lst2=lst1[0:3] # Slice Based Copy

>>> print(lst2,id(lst2))---------------------[10, 20, 30] 2073692289792

>>> lst2.append(12.34)

>>> lst1.append(70)

>>> print(lst1,id(lst1))-------------------[10, 20, 30, 40, 50, 60, 70] 2073692289216

>>> print(lst2,id(lst2))------------------[10, 20, 30, 12.34] 2073692289792

>>>

>>> lst3=lst1[::] # Slice Based Copy

>>> print(lst3,id(lst3))-----------------[10, 20, 30, 40, 50, 60, 70] 2073686948288

>>> lst3.insert(1,"KVR")

>>> lst1.append(80)

>>> print(lst1,id(lst1))---------------[10, 20, 30, 40, 50, 60, 70, 80] 2073692289216

>>> print(lst3,id(lst3))---------------[10, 'KVR', 20, 30, 40, 50, 60, 70] 2073686948288

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

7) count():

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

Syntax:- listobj.count(Value)

=>This Function is used for Counting Number of Occurences of a Specified Element.

=>If the Specified Element does not exist in list object then we get 0

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

Examples:

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

>>> lst=[10,20,30,40,10,20,10,60]

>>> print(lst)

[10, 20, 30, 40, 10, 20, 10, 60]

>>> lst.count(10)-----------------3

>>> len(lst)----------------------8

>>> lst.count(20)---------------2

>>> lst.count(30)----------------1

>>> lst.count(300)--------------0

>>> lst.count("H")-------------0

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

>>> list().count(10)---------------0

>>> [].count("")-------------------0

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

7) index()

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

=>Syntax:- listobj.index(Value)

=>This Function is used for finding Index of First Occurence of Sppecified Element.

=>If the Sppecified Element not existing in list object then we get ValueError.

Examples:

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

>>> lst=[10,20,30,10,60,70,80,20,45]

>>> print(lst)------------------[10, 20, 30, 10, 60, 70, 80, 20, 45]

>>> lst.index(10)-------------0

>>> lst.index(20)----------1

>>> lst.index(60)------------4

>>> lst.index(45)-----------8

>>> lst.index(145)--------------ValueError: 145 is not in list

>>> list().index("KVR")---------ValueError: 'KVR' is not in list

>>> [10,20,30].index(10)---------0

>>> [10,20,30].index(100)--------ValueError: 100 is not in list

>>> [10,20,30].index("10")-----------ValueError: '10' is not in list

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

8) reverse()

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

=>Syntax: listobj.reverse()

=>This Function is used for obtaining reverse the content of listobject (nothing but front to back and back to front)

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

Examples:

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

>>> l1=[10,20,30,-4,-5,100,12,45]

>>> print(l1,id(l1))------------------------[10, 20, 30, -4, -5, 100, 12, 45] 2670070726208

>>> l1.reverse()

>>> print(l1,id(l1))-----------------------[45, 12, 100, -5, -4, 30, 20, 10] 2670070726208

>>> l1=["Python","java","R","DS"]

>>> print(l1,id(l1))------------------------['Python', 'java', 'R', 'DS'] 2670074921088

>>> l1.reverse()

>>> print(l1,id(l1))------------------------['DS', 'R', 'java', 'Python'] 2670074921088

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

9) sort ()

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

=>This function is used for sorting the Homogeneous (Similar) data either in Ascending Order (reverse = False) or in Descending Order (reverse=True)

=>When we call sort () on list object where it contains Hetrogeneous (different) data then we get Type Error.

=>Syntax: listobj.sort() ---- Display the data in Ascending Order

=>Syntax: listobj.sort(reverse=False)---Display the data in Ascending Order

(default value of reverse is False)

=>Syntax: listobj.sort(reverse=True)---Display the data in Descending Order

Examples:

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

>>> l1=[10,-4,23,15,56,3,-5,34,0]

>>> print(l1,id(l1))---------------[10, -4, 23, 15, 56, 3, -5, 34, 0] 2670070726208

>>> l1.sort()

>>> print(l1,id(l1))---------------------[-5, -4, 0, 3, 10, 15, 23, 34, 56] 2670070726208

>>> l2=["Travis","Kinney","Rossum","Trump","Biden","Dennis","Anil"]

>>> print(l2)----['Travis', 'Kinney', 'Rossum', 'Trump', 'Biden', 'Dennis', 'Anil']

>>> l2.sort()

>>> print(l2)----['Anil', 'Biden', 'Dennis', 'Kinney', 'Rossum', 'Travis', 'Trump']

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

>>> l3=[10,"Rossum",34.56,True]

>>> l3.sort()---------TypeError: '<' not supported between instances of 'str' and 'int'

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

>>> l2=["Travis","Kinney","Rossum","Trump","Biden","Dennis","Anil"]

>>> print(l2) ---------['Travis', 'Kinney', 'Rossum', 'Trump', 'Biden', 'Dennis', 'Anil']

>>> l2.sort()

>>> print(l2) -------------['Anil', 'Biden', 'Dennis', 'Kinney', 'Rossum', 'Travis', 'Trump']

>>> l2.reverse()

>>> print(l2) ------------['Trump', 'Travis', 'Rossum', 'Kinney', 'Dennis', 'Biden', 'Anil']

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

>>> l1=[10,-4,23,15,56,3,-5,34,0]

>>> print(l1,id(l1))-----------------[10, -4, 23, 15, 56, 3, -5, 34, 0] 2670074921088

>>> l1.sort()

>>> print(l1,id(l1))------------------[-5, -4, 0, 3, 10, 15, 23, 34, 56] 2670074921088

>>> l1.reverse()

>>> print(l1,id(l1))--------------[56, 34, 23, 15, 10, 3, 0, -4, -5] 2670074921088

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

>>> l1=[10,-4,23,15,56,3,-5,34,0]

>>> print(l1,id(l1))----------------[10, -4, 23, 15, 56, 3, -5, 34, 0] 2670070726208

>>> l1.sort(reverse=True)

>>> print(l1,id(l1))----------------[56, 34, 23, 15, 10, 3, 0, -4, -5] 2670070726208

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

>>> l1=[10,-4,23,15,56,3,-5,34,0]

>>> print(l1,id(l1))

[10, -4, 23, 15, 56, 3, -5, 34, 0] 2670070726208

>>> l1.sort(reverse=False) # OR l1.sort()

>>> print(l1,id(l1))------------------[-5, -4, 0, 3, 10, 15, 23, 34, 56] 2670070726208

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

>>> l1=[10,-4,23,15,56,3,-5,34,0]

>>> print(l1,id(l1))-------------[10, -4, 23, 15, 56, 3, -5, 34, 0] 2670074921088

>>> l1.sort()

>>> print(l1,id(l1))-------------[-5, -4, 0, 3, 10, 15, 23, 34, 56] 2670074921088

-----------------------------------------------------------------------10) extend ()

-----------------------------------------------------------------------=> Syntax: listobj1.extend(listobj2)

=>This Function is used for extending the functionality of listobj1 with the values of listobj2.

=>At any point time, extend () takes one list object as argument

=>If we want extend the functionality of one list object with multiple objects then we can use + operator.

=>Syntax: - listobj1=listobj1+listobj2+......listobj-n

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

Examples:

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

>>> l1=[10,20,30]

>>> l2=["RS","TR","SD"]

>>> l1.extend(l2)

>>> print(l1)------------[10, 20, 30, 'RS', 'TR', 'SD']

>>> print(l2)------------['RS', 'TR', 'SD']

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

>>> l1=[10,20,30]

>>> l2=["RS","TR","SD"]

>>> l2.extend(l1)

>>> print(l1)------------[10, 20, 30]

>>> print(l2)-----------['RS', 'TR', 'SD', 10, 20, 30]

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

>>> l1=[10,20,30]

>>> l2=["RS","TR","SD"]

>>> l3=["Python","R"]

>>> l1.extend(l2,l3)---------------TypeError: list.extend() takes exactly one argument (2 given)

NOTE:

>>> l1=l1+l2+l3

>>> print(l1)-------------[10, 20, 30, 'RS', 'TR', 'SD', 'Python', 'R']

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

11) clear()

=>Syntax: listobj.clear()

=>This Function is used removing all the elements of non-empotylist

=>

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

Examples:

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

>>> l1=[10,-4,23,15,56,3,-5,34,0]

>>> print(l1,id(l1))------------[10, -4, 23, 15, 56, 3, -5, 34, 0] 2670074921088

>>> len(l1)-----------------9

>>> l1.clear()

>>> print(l1,id(l1))----------[] 2670074921088

>>> len(l1)--------------------0

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

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

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

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

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

Inner List OR Nested List

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

=>The Process of defining one list in another list is called Inner or Nested List

=>Syntax:

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

listobj=[Val1, Val2.......[Val11,Val12..] ,[ Val21,Val22.....], Val-n ]

=>here [Val11, Val12..] is one Inner List

=>Here [ Val21, Val22.....] is another Inner list

=> [Val1, Val2......., Val-n ] is called Outer List

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

Examples:

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

>>> sinfo=[10,"Rossum”, [19,17,20] , [78,77,79] ,"OUCET" ]

>>> sinfo[0]--------------------10

>>> sinfo[-1]------------------- 'OUCET'

>>> sinfo[1]--------------------- 'Rossum'

>>> sinfo[2]--------------------[19, 17, 20]

>>> print(sinfo[2], type(sinfo[2]))---------------[19, 17, 20] <class 'list'>

>>> print(sinfo[2], type(sinfo[2]), type(sinfo))--------[19, 17, 20] <class 'list'> <class 'list'>

>>> print(sinfo[-2], type(sinfo[-2]), type(sinfo))--------[78, 77, 79] <class 'list'> <class 'list'>

>>> print(sinfo[0], type(sinfo[0]), type(sinfo))---------10 <class 'int'> <class 'list'>

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

>>> sinfo=[10,"Rossum”, [19,17,20] , [78,77,79] ,"OUCET" ]

>>> print(sinfo)------------[10, 'Rossum', [19, 17, 20], [78, 77, 79], 'OUCET']

>>> sinfo[2][-1]------------20

>>> sinfo[2][::]-------------[19, 17, 20]

>>> sinfo[2][::-1]------------[20, 17, 19]

>>> sinfo[2][2]=18

>>> print(sinfo)---------------[10, 'Rossum', [19, 17, 18], [78, 77, 79], 'OUCET']

>>> sinfo[2]. sort()

>>> print(sinfo)---------------[10, 'Rossum', [17, 18, 19], [78, 77, 79], 'OUCET']

>>> sinfo[-2]. sort(reverse=True)

>>> print(sinfo)-----------------[10, 'Rossum', [17, 18, 19], [79, 78, 77], 'OUCET']

>>> sinfo[2][0:2]--------------[17, 18]

>>> sinfo[2][::2]--------------[17, 19]

>>> sinfo[-3].remove(18)

>>> print(sinfo)-----------------[10, 'Rossum', [17, 19], [79, 78, 77], 'OUCET']

>>> del sinfo[-2][1:]

>>> print(sinfo)---------------[10, 'Rossum', [17, 19], [79], 'OUCET']

>>> sinfo[2].clear()

>>> print(sinfo)-------------[10, 'Rossum', [], [79], 'OUCET']

>>> del sinfo[2]

>>> print(sinfo)----------------[10, 'Rossum', [79], 'OUCET']

>>> del sinfo[-2]

>>> print(sinfo)--------------------[10, 'Rossum', 'OUCET']

>>> im=[16,17,14]

>>> sinfo.insert(2,im)

>>> print(sinfo)------------------------[10, 'Rossum', [16, 17, 14], 'OUCET']

>>> sinfo.insert(3,[67,74,66])

>>> print(sinfo)--------------------[10, 'Rossum', [16, 17, 14], [67, 74, 66], 'OUCET']

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

>>> print(sinfo)------------[10, 'Rossum', [16, 17, 14], [67, 74, 66], 'OUCET']

>>> k=["PYTHON","R"]

>>> sinfo[2].insert(1,k)

>>> print(sinfo)-----------[10, 'Rossum', [16, ['PYTHON', 'R'], 17, 14], [67, 74, 66], 'OUCET']

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

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

2) tuple

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

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

=>The purpose of tuple data type is that "To store Collection of Values or multiple values either of Same type or different type or both the types with unique and duplicate."

=>The elements of tuple must be stored within braces () and the elements must separate by comma.

=>An object of tuple maintains inerstion Order.

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

=>An object of tuple belongs to immutable bcoz tuple' object does not support item assignment

=>To convert any other object into tuple type object, we use tuple ()

Syntax: - tupleobject=tuple (another object)

=>We can create two types of tuple objects. They are

a) empty tuple

b) non-empty tuple

a) empty tuple:

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

=>An empty tuple is one, which does not contain any elements and length is 0

=>Syntax:- tupleobj=()

or

tupleobj=tuple()

Examples:

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

>>> t=()

>>> print(t,type(t),id(l))------------ () <class 'tuple'> 2722448959680

>>> len(t)----------- 0

>>> l1=tuple()

>>> print(l1,type(l1),id(l1))------------- () <class 'tuple'> 2722452472064

>>> len(l1)------------------ 0

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

b) non-empty tuple:

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

=>A non-empty tuple is one, which contains elements and length is >0

Syntax:- tplobj=(val1,val2...val-n)

(OR)

tplobj=val1,val2...val-n

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

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

-----------------------------------------------------------------------Examples:

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

>>> t1=(10,20,30,40,10,10)

>>> print(t1,type(t1))---------------(10, 20, 30, 40, 10, 10) <class 'tuple'>

>>> t2=(10,"Ram",34.56, True,2+4.5j)

>>> print(t2,type(t2),id(t2))-------------(10, 'Ram', 34.56, True, (2+4.5j)) <class 'tuple'> 2328568492208

>>> t2[0]----------------10

>>> t2[1]------------------'Ram'

>>> t2[-1]-----------------(2+4.5j)

>>> t2[1:4]---------------------('Ram', 34.56, True)

>>> t2[2]=56.78-----------TypeError: 'tuple' object does not support item assignment

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

>>> t1=()

>>> print(t1,len(t1))------------------() 0

(OR)

>>> t2=tuple()

>>> print(t2,len(t2))--------------------() 0

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

>>> l1=[10,"Rossum"]

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

>>> t1=tuple(l1)

>>> print(t1,len(t1))---------------(10, 'Rossum') 2

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

>>> a=10,"KVR","Python”, True # without braces ( )

>>> print(a,type(a))---------------------------(10, 'KVR', 'Python', True) <class 'tuple'>

>>> a=10,

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

>>> a=10

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

>>> t=tuple(a)-------------TypeError: 'int' object is not iterable

>>> t=tuple(a,)------------TypeError: 'int' object is not iterable

>>> t=tuple((a)) -----------TypeError: 'int' object is not iterable

>>> t=(a,) # correct conversion

>>> print(t,type(t))----------------(10,) <class 'tuple'>

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

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

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

pre-defined functions in tuple

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

=>tuple object contains two pre-defined functions. They are

1. count()

2. index()

Examples:

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

>>> t1=(10,10,20,30,10,10,30)

>>> t1.count(10)----------------4

>>> t1.count(30)----------------2

>>> t1.count(300)--------------0

>>> t1.count("KVR")------------0

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

>>> t1=(10,10,20,30,10,10,30)

>>> t1.index(10)------------0

>>> t1.index(20)------------2

>>> t1.index(230)----------ValueError: tuple.index(x): x not in tuple

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

>>> t1=(10,10,20,30,10,10,30)

>>> for i,v in enumerate(t1):

... print(i,v)

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

Output

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

0 10

1 10

2 20

3 30

4 10

5 10

6 30

NOTE: tuple object does not contain the following pre-defined Functions bcoz tuple object belongs to immutable.

1) append()

2) insert()

3) remove()

4) pop(index)

5) pop()

6) copy()

7) clear()

8) reverse()

9) sort()

10)extend()

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

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

Inner tuple OR Tuple List

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

=>The Process of defining one tuple in another tuple is called Inner or Nested tuple

=>Syntax:

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

tupleobj=(Val1, Val2.......(Val11,Val12..) , ( Val21,Val22.....), Val-n)

=>here (Val11,Val12..) is one Inner tuple

=>Here (Val21,Val22.....) is another Inner tuple

=>(Val1, Val2......., Val-n ) is called Outer tuple

NOTE:

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

=>We can define One Tuple Inside of Another Tuple

=>We can define One List Inside of Another List

=>We can define One Tuple Inside of Another List

=>We can define One List Inside of Another Tuple

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

Examples

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

>>> t1=(10,"Rossum”, (15,18,17),(66,67,56),"OUCET")

>>> print(t1,type(t1))-------------(10, 'Rossum', (15, 18, 17), (66, 67, 56), 'OUCET') <class 'tuple'>

>>> t1[2] --------------(15, 18, 17)

>>> print(t1[2], type(t2))------------(15, 18, 17) <class 'tuple'>

>>> print(t1[-2],type(t2))------------(66, 67, 56) <class 'tuple'>

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

>>> t1=(10,"Rossum",[15,18,17],(66,67,56),"OUCET")

>>> print(t1,type(t1))------(10, 'Rossum', [15, 18, 17], (66, 67, 56), 'OUCET') <class 'tuple'>

>>> print(t1[2],type(t1[2]))-----[15, 18, 17] <class 'list'>

>>> print(t1[3],type(t1[3]))-------(66, 67, 56) <class 'tuple'>

>>> t1[2].insert(1,16)

>>> print(t1,type(t1))------(10, 'Rossum', [15, 16, 18, 17], (66, 67, 56), 'OUCET') <class 'tuple'>

>>> t1[2].sort(reverse=True)

>>> print(t1,type(t1))--------(10, 'Rossum', [18, 17, 16, 15], (66, 67, 56), 'OUCET') <class 'tuple'>

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

>>> l1=[10,"Rossum”, [15,18,17],(66,67,56),"OUCET"]

>>> print(l1,type(l1))------------[10, 'Rossum', [15, 18, 17], (66, 67, 56), 'OUCET'] <class 'list'>

>>> l1[2]. remove(18)

>>> print(l1,type(l1))----------[10, 'Rossum', [15, 17], (66, 67, 56), 'OUCET'] <class 'list'>

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

Special Case:

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

sorted():

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

=>It is one of the general pre-defined function and is used for Sorting the elements of tuple (in this case) and gives the sorted elements in th form of list(But Sorted Elements will not place in tuple bcoz tuple is immutable).

Syntax: sorted(tuple object)

(OR)

listobj=sorted(tupleobj)

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

Examples:

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

>>> t1=(12,45,-3,3,0,14)

>>> print(t1,type(t1))-------------------(12, 45, -3, 3, 0, 14) <class 'tuple'>

>>> t1. sort()-------------AttributeError: 'tuple' object has no attribute 'sort'

>>> sorted(t1) ---------- [-3, 0, 3, 12, 14, 45]

>>> print(t1,type(t1))-----------(12, 45, -3, 3, 0, 14) <class 'tuple'>

>>> x=sorted(t1)

>>> print(x,type(x))--------------[-3, 0, 3, 12, 14, 45] <class 'list'>

(OR)

>>> t1=(12,45,-3,3,0,14)

>>> print(t1, type(t1))-------------(12, 45, -3, 3, 0, 14) <class 'tuple'>

>>> l1=list(t1)

>>> print(l1,type(l1))-------------[12, 45, -3, 3, 0, 14] <class 'list'>

>>> l1.sort()

>>> print(l1,type(l1))-----------[-3, 0, 3, 12, 14, 45] <class 'list'>

>>> t1=tuple(l1)

>>> print(t1,type(t1))------(-3, 0, 3, 12, 14, 45) <class 'tuple'>

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

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

Set Categery Data Types(Collections Data Types or Data Structures)

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

=>The purpose of Set Categery Data Types is that " To store Collection or multiple values either of same type or different type or both the types with Unique Values ( No duplicates are allowed)".

=>We have 2 data types in Set Categery. They are

1. set (mutable and immutable)

2. frozenset (immutable)

1. 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 Collection or multiple values either of same type or different type or both the types with Unique Values ( No duplicatesd are allowed)".

=>The elements of set must be organized within curly braces { } and elements must separate by comma,

=>An object of set does not maintain insertion order bcoz PVM displays any order of multiple possibilities.

=>On the object of set, we can't perform Indexing and slicing Operations bcoz set object does not maintain Insertion order.

=>An object of set belongs to immutable (bcoz of 'set' object does not support item assignment) and mutable ( bcoz in the case of add() ). =>By using set class, we can two types of set objects. They are

a) empty set

b) non-empty set

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

a) empty set:

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

=>An empty set is one, which does not contain any elements and whose length is 0

=>Syntax:- setobj=set()

b) non-empty set:

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

=>A non-empty set is one, which contains elements and whose length is >0

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

=>To convert one type of object into set type object, we use set()

Syntax: setobj=set(obj)

-----------------------------------------------------------------------Examples:

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

>>> s1={10,20,30,40,50,10,10,20,75}

>>> print(s1,type(s1))-----------------{50, 20, 40, 10, 75, 30} <class 'set'>

>>> s1={10,20,25,35,10,20}

>>> print(s1,type(s1))-------------------{25, 10, 35, 20} <class 'set'>

>>> s1[0] ----------------TypeError: 'set' object is not subscriptable

>>> s1[0:3]--------------TypeError: 'set' object is not subscriptable

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

>>> s1= {10,20,30,40,50}

>>> print(s1,id(s1))-------------------------------{50, 20, 40, 10, 30} 1473821509440

>>> s1[0] =100-------------TypeError: 'set' object does not support item assignment

>>> s1.add("KVR")

>>> print(s1,id(s1))------------------{50, 20, 40, 10, 'KVR', 30} 1473821509440

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

>>> s1=set ()

>>> print(s1,type(s1))------------------{} <class 'set'>

>>> len(s1) ------------0

>>> s2= {10,20,30,10,20}

>>> print (s2, type(s2))---------------{10, 20, 30} <class 'set'>

>>> len(s2) -------------------3

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

>>> l1=[10,20,10,20,"Python",23.45]

>>> s1=set(l1)

>>> print(s1) -----------------{10, 20, 23.45, 'Python'}

>>> t1=tuple(s1)

>>> print(t1, type(t1))------------(10, 20, 23.45, 'Python') <class 'tuple'>

>>> t1=list(s1)

>>> print (t1, type(t1)) ------------[10, 20, 23.45, 'Python'] <class 'list'>

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

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

pre-defined functions in set

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

=>on the object of set, we can perform different type of Operations by using pre-defined functions in set object.

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

1) add ()

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

=>This Function is used for adding the elements to set object.

=>Syntax: setobj.add(Value)

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

Examples:

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

>>> s1={10,20,30}

>>> print(s1,type(s1),id(s1))--------------{10, 20, 30} <class 'set'> 1691649314592

>>> s1.add (12.34)

>>> print(s1,type(s1),id(s1))-------------{10, 20, 12.34, 30} <class 'set'> 1691649314592

>>> s1.add("python")

>>> print(s1,type(s1),id(s1))------------{10, 12.34, 'python', 20, 30} <class 'set'> 1691649314592

>>> s2=set()

>>> print(s2,type(s2),id(s2))----------set() <class 'set'> 1691645340672

>>> s2.add(100)

>>> s2.add("Rajesh")

>>> s2.add("Kasif")

>>> print(s2,type(s2),id(s2))--------{100, 'Kasif', 'Rajesh'} <class 'set'> 1691645340672

>>> s2.add(23.45)

>>> print(s2, type(s2),id(s2))-----{100, 23.45, 'Kasif', 'Rajesh'} <class 'set'> 1691645340672

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

2) remove()

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

=>Syntax:- setobj.remove(Value)

=>This Function is used for removing the element from set object.

=>The element / value does not exist in setobject then we get KeyError( bcoz all the elements of set are Unique and they are called Keys)

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

Examples:

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

>>> s1= {10,"Rajesh",34.56,400, True,2+3j}

>>> print(s1, type(s1))--------{400, True, 34.56, 'Rajesh', 10, (2+3j)} <class 'set'>

>>> print(s1,type(s1),id(s1))---{400, True, 34.56, 'Rajesh', 10, (2+3j)} <class 'set'> 1691649315936

>>> s1.remove(34.56)

>>> print(s1,type(s1),id(s1))----{400, True, 'Rajesh', 10, (2+3j)} <class 'set'> 1691649315936

>>> s1. remove(True)

>>> print(s1,type(s1),id(s1))---{400, 'Rajesh', 10, (2+3j)} <class 'set'> 1691649315936

>>> s1.remove("Rajesh")

>>> print(s1,type(s1),id(s1))----{400, 10, (2+3j)} <class 'set'> 1691649315936

>>> s1.remove("KVR")------KeyError: 'KVR'

>>> set().remove(10)-----------KeyError: 10

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

3) discard()

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

=>Syntax: setobj.discard(value)

=>This Function is used for removing the element from set object.

=>The element / value does not exist in setobject then we never get KeyError

Examples:

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

>>> s1={10,20,30,40,50,60,70,10}

>>> print(s1,type(s1))-------------------{50, 20, 70, 40, 10, 60, 30} <class 'set'>

>>> s1.discard(50)

>>> print(s1,type(s1))-------------------{20, 70, 40, 10, 60, 30} <class 'set'>

>>> s1.discard(10)

>>> print(s1,type(s1))-------------------{20, 70, 40, 60, 30} <class 'set'>

>>> s1.discard(100) # we never get KeyError

>>> print(s1,type(s1))-------------------{20, 70, 40, 60, 30} <class 'set'>

>>> s1.discard("Python") # we never get KeyError

>>> s1.remove("Python")-----------KeyError: 'Python'

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

4) clear()

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

=>Syntax: setobj.clear()

=>This function is used for removing all the elements of set object.

Examples:

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

>>> s1={10,20,30,40,50,60,70,10}

>>> print(s1,type(s1))---------------------{50, 20, 70, 40, 10, 60, 30} <class 'set'>

>>> len(s1)------------------------7

>>> s1.clear()

>>> print(s1,type(s1))-----------------set() <class 'set'>

>>> len(s1)------------------0

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

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

5) copy() -----------Shallow Copy

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

Syntax: setobj2=setobj1.copy()

=>This Function is used for copying the content of one set object into another set object

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

=>Examples:

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

>>> s1={10,20,30,40,50,60,70,10}

>>> print(s1, type(s1),id(s1))-------------{50, 20, 70, 40, 10, 60, 30} <class 'set'> 2424304921600

>>> s2=s1.copy()

>>> print(s2,type(s2),id(s2))----------{50, 20, 70, 40, 10, 60, 30} <class 'set'> 2424308895072

>>> s1.add(12.34)

>>> s2.add("Python")

>>> print(s1,type(s1),id(s1))------{50, 20, 70, 40, 10, 60, 12.34, 30} <class 'set'> 2424304921600

>>> print(s2,type(s2),id(s2))---{50, 20, 'Python', 70, 40, 10, 60, 30} <class 'set'> 2424308895072

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

6) isdisjoint()

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

=>Syntax: setobj1.isdisjoin(s2)

=>This Function returns True Provided there is no common eleement between setobj1 and setobj2.

=>This Function returns False Provided there is atleast common element between setobj1 and setobj2.

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

Examples:

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

>>> s1={10,20,30,40}

>>> s2={"Apple","Mango","kiwi"}

>>> s3={10,50,60}

>>> s1. isdisjoint(s2) --------------True

>>> s1. isdisjoint(s3) --------------False

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

7) issuperset()

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

Syntax: setobj1.issuperset(setobj2)

=>This Function return True provided all elements setobj2 must present setobj1

OR

=> setobj1 must contains all elements of setobj2

Examples:

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

>>> s1={10,20,30,40}

>>> s2={10,20}

>>> s3={10,20, "Apple","Mango","kiwi"}

>>> s1. issuperset(s2)----------True

>>> s1. issuperset(s3)---------False

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

8) issubset()

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

Syntax: setobj1.issubset(setobj2)

=>This Function return True provided all elements setobj1 must present setobj2

OR

=>setobj2 must contains all elements of setobj1

Examples:

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

>>> s1={10,20,30,40}

>>> s2={10,20}

>>> s3={10,20, "Apple","Mango","kiwi"}

>>> s2.issubset(s1)---------True

>>> s3.issubset(s1)----------False

>>> s3.issubset(s2)---------False

>>> s2.issubset(s3)-------True

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

9) union()

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

Syntax:- setobj1.union(setobj2)

(OR)

setobj3=setob1.union(setobj2)

=>This is used for obtaining all Unique Elements of setobj1 and setobj2 and result unique values placed in setobj3.

Examples:

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

>>> s1={10,20,30,40}

>>> s2={15,10,25}

>>> s3=s1.union(s2)

>>> print(s1)-----------{40, 10, 20, 30}

>>> print(s2)----------{25, 10, 15}

>>> print(s3)-----------{20, 40, 25, 10, 30, 15}

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

>>> print(s1.union(s2))-----------{20, 40, 25, 10, 30, 15}

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

10) intersection()

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

Syntax: setobj1.intersection(setobj2)

(OR)

setobj3= setobj1.intersection(setobj2)

=>This function is used for obtaining common elements from setobj1 and setobj2.

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

Examples:

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

>>> s1={10,20,30,40}

>>> s2={15,10,25}

>>> s3=s1.intersection(s2)

>>> print(s3) ------------{10}

>>> s3=s2.intersection(s1)

>>> print(s3) -------------{10}

>>> print(s1.intersection(s2))-------{10}

>>> s1={10,20,30,40}

>>> s2={"Apple","Mango","kiwi"}

>>> print(s1.intersection(s2))------------set()

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

11)difference()

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

=>Syntax: setobj1.difference(setobj2)

=>This obtains removes common elements from setobj1 and setobj2 and Takes remaining elements from setobj1 and place them setobj3.

Examples:

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

>>> s1={10,20,30,40}

>>> s2={10,15,25}

>>> s3=s1.difference(s2)

>>> print(s1)--------{40, 10, 20, 30}

>>> print(s2)-------------{25, 10, 15}

>>> print(s3)---------{40, 20, 30}

>>> s4=s2.difference(s1)

>>> print(s4)--------{25, 15}

>>> a = {1, 3 ,5}

>>> b = {2, 4, 6}

>>> c = {1, 2}

>>> print(a)---------{1, 3, 5}

>>> print(b)--------{2, 4, 6}

>>> print(c)-------{1, 2}

>>> d=a.difference(b).difference(c)

>>> print(d)--------{3, 5}

>>> d=a.difference(b,c)

>>> print(d)-------------{3, 5}

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

12) symmetric\_difference()

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

=>Syntax: setobj1.symmetric\_difference(setobj2)

=>This function removes common elements from both setobj1 and setobj2 and takes remaining elements from both setobj1 and setobj2 and place them setobj3.

Examples:

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

>>> s1={10,20,30,40}

>>> s2={10,15,25}

>>> s3=s1.symmetric\_difference(s2)

>>> print(s1)----------{40, 10, 20, 30}

>>> print(s2)--------{25, 10, 15}

>>> print(s3)--------{40, 15, 20, 25, 30}

>>> s3=s2.symmetric\_difference(s1)

>>> print(s3)-------------{40, 15, 20, 25, 30}

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

Use-Case:

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

>>> cp={"sachin","kohli","rohit"}

>>> tp={"rossum","saroj","rohit"}

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

>>> allcptp=cp.union(tp)

>>> print(allcptp)-----------------{'kohli', 'sachin', 'rohit', 'rossum', 'saroj'}

>>> bothcptp=cp.intersection(tp)

>>> print(bothcptp)--------------{'rohit'}

>>> onlycp=cp.difference(tp)

>>> print(onlycp)-------------{'kohli', 'sachin'}

>>> onlytp=tp.difference(cp)

>>> print(onlytp)-------------{'rossum', 'saroj'}

>>> exclcptp=cp.symmetric\_difference(tp)

>>> print(exclcptp)---------------{'sachin', 'rossum', 'kohli', 'saroj'}

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

MOST IMP Case:

>>> allcptp=cp|tp # Bitwise OR Operator ( | )

>>> print(allcptp)------{'kohli', 'sachin', 'rohit', 'rossum', 'saroj'}

>>> bothptp=cp&tp # Bitwise AND Operator ( & )

>>> print(bothcptp)---------{'rohit'}

>>> onlycp=cp-tp # Subtract Operator

>>> print(onlycp)--------{'kohli', 'sachin'}

>>> onlytp=tp-cp # Subtract Operator

>>> print(onlytp)----------{'rossum', 'saroj'}

>>> exclcptp=cp^tp # Biwise XOR Operator ( ^ )

>>> print(exclcptp)--------{'sachin', 'rossum', 'kohli', 'saroj'}

>>> print({10,20,30} & {10,25,67,34})----------{10}

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

13) update()

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

=>Syntax: setobj1.update(setobj2)

=>This Function is used for updating the values of setobj2 with setobj1.

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

Examples:

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

>>> s1={10,20,30}

>>> s2={"Python","Java"}

>>> print(s1,id(s1))----------{10, 20, 30} 2424308898432

>>> print(s2,id(s2))----------{'Java', 'Python'} 2424308895072

>>> s1. update(s2)

>>> print(s1, id(s1))----------{20, 'Java', 10, 'Python', 30} 2424308898432

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

>>> s1={10,20,30}

>>> s2={10,20,"Python"}

>>> print(s1,id(s1))---------{10, 20, 30} 2424308896416

>>> print(s2,id(s2))----------{10, 20, 'Python'} 2424308898432

>>> s1.update(s2)

>>> print(s1,id(s1))----------{20, 10, 'Python', 30} 2424308896416

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

Nested or Inner Formulas

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

Imp Points:

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

=> Set in Set Not Possible

=>Tuple in set Possible (No use bcoz we can't locate by using Indexing)

=>List in set Not Possible ( bcoz list is mutable and allows changes )

=>set in Tuple Possible (boz tuple permits to locate set object by using indexing)

=>Set in list Possible (boz tuple permits to locate set object by using indexing)

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

Examples:

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

>>> l1=[10,"Akash",{10,20,30},[23,45,23],"OUCET" ]

>>> print(l1,type(l1))

[10, 'Akash', {10, 20, 30}, [23, 45, 23], 'OUCET'] <class 'list'>

>>> print(l1[0],type(l1[0]))-------------10 <class 'int'>

>>> print(l1[1],type(l1[2]))------------Akash <class 'set'>

>>> print(l1[2],type(l1[2]))----------{10, 20, 30} <class 'set'>

>>> l1[2][0] ------------TypeError: 'set' object is not subscriptable

>>> l1[:3] ---------------[10, 'Akash', {10, 20, 30}]

>>> l1[2].add(23)

>>> print(l1)------------[10, 'Akash', {10, 20, 30, 23}, [23, 45, 23], 'OUCET']

>>> l1[-2][0]------------23

>>> l1[-3][0]--------TypeError: 'set' object is not subscriptable

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

>>> t1=(10,"Akash",{10,20,30},[23,45,23],"OUCET")

>>> print(t1,type(t1))-----------(10, 'Akash', {10, 20, 30}, [23, 45, 23], 'OUCET') <class 'tuple'>

>>> print(t1[2],type(t1[2]))----------{10, 20, 30} <class 'set'>

>>> print(t1[2],type(t1[3]))----------{10, 20, 30} <class 'list'>

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

>>> s1={10,"Akash",(10,20,30),(23,45,23),"OUCET"}

>>> print(s1,type(s1))--------{'OUCET', 'Akash', (23, 45, 23), 10, (10, 20, 30)} <class 'set'>

>>> print(s1[2],type(s1[2]))---TypeError: 'set' object is not subscriptable

>>> s1={10,"Akash",[10,20,30],(23,45,23),"OUCET"}----TypeError: unhashable type: 'list'

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

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

pre-defined functions in set

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

=>on the object of set, we can perform different type of Operations by using pre-defined functions in set object.

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

1) add()

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

=>This Function is used for adding the elements to set object.

=>Syntax: setobj.add(Value)

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

Examples:

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

>>> s1={10,20,30}

>>> print(s1,type(s1),id(s1))--------------{10, 20, 30} <class 'set'> 1691649314592

>>> s1.add (12.34)

>>> print(s1, type(s1),id(s1))-------------{10, 20, 12.34, 30} <class 'set'> 1691649314592

>>> s1.add("python")

>>> print(s1, type(s1),id(s1))------------{10, 12.34, 'python', 20, 30} <class 'set'> 1691649314592

>>> s2=set()

>>> print(s2, type(s2),id(s2))----------set() <class 'set'> 1691645340672

>>> s2.add(100)

>>> s2.add("Rajesh")

>>> s2.add("Kasif")

>>> print(s2,type(s2),id(s2))--------{100, 'Kasif', 'Rajesh'} <class 'set'> 1691645340672

>>> s2.add(23.45)

>>> print(s2,type(s2),id(s2))-----{100, 23.45, 'Kasif', 'Rajesh'} <class 'set'> 1691645340672

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

2) remove ()

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

=>Syntax:- setobj.remove(Value)

=>This Function is used for removing the element from set object.

=>The element / value does not exist in setobject then we get KeyError( bcoz all the elements of set are Unique and they are called Keys)

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

Examples:

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

>>> s1={10,"Rajesh",34.56,400,True,2+3j}

>>> print(s1,type(s1))--------{400, True, 34.56, 'Rajesh', 10, (2+3j)} <class 'set'>

>>> print(s1,type(s1),id(s1))---{400, True, 34.56, 'Rajesh', 10, (2+3j)} <class 'set'> 1691649315936

>>> s1.remove(34.56)

>>> print(s1,type(s1),id(s1))----{400, True, 'Rajesh', 10, (2+3j)} <class 'set'> 1691649315936

>>> s1.remove(True)

>>> print(s1,type(s1),id(s1))---{400, 'Rajesh', 10, (2+3j)} <class 'set'> 1691649315936

>>> s1.remove("Rajesh")

>>> print(s1,type(s1),id(s1))----{400, 10, (2+3j)} <class 'set'> 1691649315936

>>> s1.remove("KVR")------KeyError: 'KVR'

>>> set().remove(10)-----------KeyError: 10

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

3) discard()

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

=>Syntax: setobj.discard(value)

=>This Function is used for removing the element from set object.

=>The element / value does not exist in setobject then we never get KeyError

Examples:

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

>>> s1={10,20,30,40,50,60,70,10}

>>> print(s1, type(s1))-------------------{50, 20, 70, 40, 10, 60, 30} <class 'set'>

>>> s1. discard(50)

>>> print(s1,type(s1))-------------------{20, 70, 40, 10, 60, 30} <class 'set'>

>>> s1.discard(10)

>>> print(s1,type(s1))-------------------{20, 70, 40, 60, 30} <class 'set'>

>>> s1.discard(100) # we never get KeyError

>>> print(s1,type(s1))-------------------{20, 70, 40, 60, 30} <class 'set'>

>>> s1.discard("Python") # we never get KeyError

>>> s1.remove("Python")-----------KeyError: 'Python'

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

4) clear()

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

=>Syntax: setobj.clear()

=>This function is used for removing all the elements of set object.

Examples:

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

>>> s1={10,20,30,40,50,60,70,10}

>>> print(s1,type(s1))---------------------{50, 20, 70, 40, 10, 60, 30} <class 'set'>

>>> len(s1)------------------------7

>>> s1.clear()

>>> print(s1,type(s1))-----------------set() <class 'set'>

>>> len(s1) ------------------0

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

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

5) copy() -----------Shallow Copy

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

Syntax: setobj2=setobj1.copy()

=>This Function is used for copying the content of one set object into another set object

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

=>Examples:

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

>>> s1={10,20,30,40,50,60,70,10}

>>> print(s1, type(s1),id(s1))-------------{50, 20, 70, 40, 10, 60, 30} <class 'set'> 2424304921600

>>> s2=s1.copy()

>>> print(s2,type(s2),id(s2))----------{50, 20, 70, 40, 10, 60, 30} <class 'set'> 2424308895072

>>> s1.add(12.34)

>>> s2.add("Python")

>>> print(s1,type(s1),id(s1))------{50, 20, 70, 40, 10, 60, 12.34, 30} <class 'set'> 2424304921600

>>> print(s2,type(s2),id(s2))---{50, 20, 'Python', 70, 40, 10, 60, 30} <class 'set'> 2424308895072

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

6) isdisjoint()

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

=>Syntax: setobj1.isdisjoin(s2)

=>This Function returns True Provided there is no common eleement between setobj1 and setobj2.

=>This Function returns False Provided there is atleast common eleement between setobj1 and setobj2.

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

Examples:

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

>>> s1={10,20,30,40}

>>> s2={"Apple","Mango","kiwi"}

>>> s3={10,50,60}

>>> s1.isdisjoint(s2)--------------True

>>> s1.isdisjoint(s3)--------------False

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

7) issuperset()

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

Syntax: setobj1.issuperset(setobj2)

=>This Function return True provided all elements setobj2 must present setobj1

OR

setobj1 must contains all elements of setobj2

Examples:

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

>>> s1={10,20,30,40}

>>> s2={10,20}

>>> s3={10,20, "Apple","Mango","kiwi"}

>>> s1.issuperset(s2)----------True

>>> s1.issuperset(s3)---------False

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

8) issubset()

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

Syntax: setobj1.issubset(setobj2)

=>This Function return True provided all elements setobj1 must present setobj2

OR

setobj2 must contains all elements of setobj1

Examples:

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

>>> s1={10,20,30,40}

>>> s2={10,20}

>>> s3={10,20, "Apple","Mango","kiwi"}

>>> s2.issubset(s1)---------True

>>> s3.issubset(s1)----------False

>>> s3.issubset(s2)---------False

>>> s2.issubset(s3)-------True

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

9) union()

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

Syntax:- setobj1.union(setobj2)

(OR)

setobj3=setob1.union(setobj2)

=>This is used for for obtaining all Unique Elements of setobj1 and setobj2 and result unique values placed in setobj3.

Examples:

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

>>> s1={10,20,30,40}

>>> s2={15,10,25}

>>> s3=s1.union(s2)

>>> print(s1)-----------{40, 10, 20, 30}

>>> print(s2)----------{25, 10, 15}

>>> print(s3)-----------{20, 40, 25, 10, 30, 15}

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

>>> print(s1.union(s2))-----------{20, 40, 25, 10, 30, 15}

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

10) intersection()

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

Syntax: setobj1.intersection(setobj2)

(OR)

setobj3= setobj1.intersection(setobj2)

=>This function is used for obtaining common elements from setobj1 and setobj2.

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

Examples:

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

>>> s1={10,20,30,40}

>>> s2={15,10,25}

>>> s3=s1.intersection(s2)

>>> print(s3)------------{10}

>>> s3=s2.intersection(s1)

>>> print(s3)-------------{10}

>>> print(s1.intersection(s2))-------{10}

>>> s1={10,20,30,40}

>>> s2={"Apple","Mango","kiwi"}

>>> print(s1.intersection(s2))------------set()

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

11)difference()

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

=>Syntax: setobj1.difference(setobj2)

=>This obtains removes common elements from setobj1 and setobj2 and Takes remaining elements from setobj1 and place them setobj3.

Examples:

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

>>> s1={10,20,30,40}

>>> s2={10,15,25}

>>> s3=s1.difference(s2)

>>> print(s1)--------{40, 10, 20, 30}

>>> print(s2)-------------{25, 10, 15}

>>> print(s3)---------{40, 20, 30}

>>> s4=s2.difference(s1)

>>> print(s4)--------{25, 15}

>>> a = {1, 3 ,5}

>>> b = {2, 4, 6}

>>> c = {1, 2}

>>> print(a)---------{1, 3, 5}

>>> print(b)--------{2, 4, 6}

>>> print(c)-------{1, 2}

>>> d=a.difference(b).difference(c)

>>> print(d)--------{3, 5}

>>> d=a.difference(b,c)

>>> print(d)-------------{3, 5}

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

12) symmetric\_difference()

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

=>Syntax: setobj1.symmetric\_difference(setobj2)

=>This function removes common elements from both setobj1 and setobj2 and Takes remaining elements from both setobj1 and setobj2 and place them setobj3.

Examples:

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

>>> s1={10,20,30,40}

>>> s2={10,15,25}

>>> s3=s1.symmetric\_difference(s2)

>>> print(s1)----------{40, 10, 20, 30}

>>> print(s2)--------{25, 10, 15}

>>> print(s3)--------{40, 15, 20, 25, 30}

>>> s3=s2.symmetric\_difference(s1)

>>> print(s3)-------------{40, 15, 20, 25, 30}

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

Use-Case:

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

>>> cp={"sachin","kohli","rohit"}

>>> tp={"rossum","saroj","rohit"}

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

>>> allcptp=cp.union(tp)

>>> print(allcptp)-----------------{'kohli', 'sachin', 'rohit', 'rossum', 'saroj'}

>>> bothcptp=cp.intersection(tp)

>>> print(bothcptp)--------------{'rohit'}

>>> onlycp=cp.difference(tp)

>>> print(onlycp)-------------{'kohli', 'sachin'}

>>> onlytp=tp.difference(cp)

>>> print(onlytp)-------------{'rossum', 'saroj'}

>>> exclcptp=cp.symmetric\_difference(tp)

>>> print(exclcptp)---------------{'sachin', 'rossum', 'kohli', 'saroj'}

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

MOST IMP Case:

>>> allcptp=cp|tp # Bitwise OR Operator ( | )

>>> print(allcptp)------{'kohli', 'sachin', 'rohit', 'rossum', 'saroj'}

>>> bothptp=cp&tp # Bitwise AND Operator ( & )

>>> print(bothcptp)---------{'rohit'}

>>> onlycp=cp-tp # Subtract Operator

>>> print(onlycp)--------{'kohli', 'sachin'}

>>> onlytp=tp-cp # Subtract Operator

>>> print(onlytp)----------{'rossum', 'saroj'}

>>> exclcptp=cp^tp # Biwise XOR Operator ( ^ )

>>> print(exclcptp)--------{'sachin', 'rossum', 'kohli', 'saroj'}

>>> print({10,20,30} & {10,25,67,34})----------{10}

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

13) update()

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

=>Syntax: setobj1.update(setobj2)

=>This Function is used for updating the values of setobj2 with setobj1.

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

Examples:

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

>>> s1={10,20,30}

>>> s2={"Python","Java"}

>>> print(s1,id(s1))----------{10, 20, 30} 2424308898432

>>> print(s2,id(s2))----------{'Java', 'Python'} 2424308895072

>>> s1.update(s2)

>>> print(s1,id(s1))----------{20, 'Java', 10, 'Python', 30} 2424308898432

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

>>> s1={10,20,30}

>>> s2={10,20,"Python"}

>>> print(s1,id(s1))---------{10, 20, 30} 2424308896416

>>> print(s2,id(s2))----------{10, 20, 'Python'} 2424308898432

>>> s1.update(s2)

>>> print(s1,id(s1))----------{20, 10, 'Python', 30} 2424308896416

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

14) pop()

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

=>Syntax: setobj.pop ()

=>This Function is used for removing any Arbitary Element from setobject.

=>when we call pop () on empty set() then we get KeyError

Examples:

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

>>> s1={10,"Abinash","Python",45.67, True,2+3j}

>>> s1.pop ()-------------True

>>> s1.pop()------------10

>>> s1.pop()------------'Abinash'

>>> s1.pop()------------'Python'

>>> s1.pop()---------------(2+3j)

>>> s1.pop()---------------45.67

>>> s1.pop()-------------KeyError: 'pop from an empty set'

>>> set().pop()-----------KeyError: 'pop from an empty set'

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

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

2. frozenset

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

=>'frozenset' is one of the pre-defined classes and treated as set data type.

=>The purpose of frozenset data type is that To store multiple values of either of same type or different type or both types with Unique Values in single object."

=>The elements set must organized with curly braces {} and values must separate by comma and those values can converted into frozenset by using frozenset()

Syntax: - frozensetobj1=frozenset(setobj)

frozensetobj1=frozenset(listobj)

frozensetobj1=frozenset(tupleobj)

=>An object of frozenset does not maintain insertion Order bcoz PVM displays any possibility of elements of frozenset

=>Since frozenset object does not maintain insertion order, we can't perform Indexing and Slicing Operations ( frozenset' object is not subscriptable)

=>An object of frozenset belongs to immutable (in the case frozenset' object does not support item assignment and adding elements also not possible)

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

Note: -The Functionality of frozenset is similar to set but an object of set belongs to both immutable (in case of item assigment) and mutable (in the case of add()) where as an object frozenset belongs to immutable.

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

Examples:

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

l1= [10,20,30,40,10]

fs=frozenset(l1)

print(fs,type(fs))----------------frozenset({40, 10, 20, 30}) <class 'frozenset'>

fs.add(100)------------AttributeError: 'frozenset' object has no attribute 'add'

fs[0]=345------------TypeError: 'frozenset' object does not support item assignment

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

>>> t1=(10,20,30,10,40,23.45,56)

>>> print(t1,type(t1))-------------------(10, 20, 30, 10, 40, 23.45, 56) <class 'tuple'>

>>> fs1=frozenset(t1)

>>> print(fs1, type(fs1))-----------------frozenset({40, 10, 20, 23.45, 56, 30}) <class 'frozenset'>

>>> s1={10,"KVR",34.56,"Python","Java"}

>>> print(s1,type(s1))----------------{34.56, 10, 'KVR', 'Java', 'Python'} <class 'set'>

>>> fs2=frozenset(s1)

>>> print(fs2,type(fs2))------frozenset({34.56, 10, 'KVR', 'Java', 'Python'}) <class 'frozenset'>

>>> fs2[0]-----------------TypeError: 'frozenset' object is not subscriptable

>>> fs2[0:3]---------------TypeError: 'frozenset' object is not subscriptable

>>> fs2[0]=123----------TypeError: 'frozenset' object does not support item assignment

>>> fs2.add(100)------------AttributeError: 'frozenset' object has no attribute 'add'

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

Pre-defined functions in frozenset

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

1) copy()

2) union()

3) intersection()

4) difference()

5) symmetric\_difference()

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

Examples:

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

>>> s1={10,20,30,40}

>>> s2={15,25,30,40}

>>> fs1=frozenset(s1)

>>> fs2=frozenset(s2)

>>> print(fs1)

frozenset({40, 10, 20, 30})

>>> print(fs2)

frozenset({40, 25, 30, 15})

>>> fs3=fs1.union(fs2)

>>> print(fs3)

frozenset({40, 10, 15, 20, 25, 30})

>>> fs4=fs1.intersection(fs2)

>>> print(fs4)

frozenset({40, 30})

>>> fs5=fs1.difference(fs2)

>>> print(fs5)

frozenset({10, 20})

>>> fs6=fs2.difference(fs1)

>>> print(fs6)

frozenset({25, 15})

>>> fs7=fs2.symmetric\_difference(fs1)

>>> print(fs7)

frozenset({10, 15, 20, 25})

>>> fs7=fs1.symmetric\_difference(fs2)

>>> print(fs7)

frozenset({10, 15, 20, 25})

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

>>> s1={10,20,30,40}

>>> fs1=frozenset(s1)

>>> fs2=fs1.copy()

>>> print(fs1,id(fs1))-----------------frozenset({40, 10, 20, 30}) 2299638113984

>>> print(fs2,id(fs2))-----------------frozenset({40, 10, 20, 30}) 2299638113984

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

The following pre-defined functions not found in frozenset

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

1) add()

2) remove()

3) discard()

4) update()

5) pop()

6) clear()

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

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

pre-defined functions in dict data type

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

=>dict object contains the following pre-defined function to perform Various Operations.

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

1) clear ()

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

=>Syntax:- dictobj.clear()

=>This function removes all the (key,Value) from dict object

=>When we call clear () upon empty dict object then we get None

Examples:

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

>>> d1={10:1.2,20:2.3,40:5.6,50:1.2}

>>> print(d1, type(d1), id(d1))------------{10: 1.2, 20: 2.3, 40: 5.6, 50: 1.2} <class 'dict'> 1228171857856

>>> d1. clear()

>>> print(d1,type(d1), id(d1))------------{} <class 'dict'> 1228171857856

>>> print(d1.clear())---------------None

-----------------------------------------------------------------------2) copy()

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

=>This Function is used copying the content of one dict object into another dict object (implementation of shalow copy)

Examples:

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

>>> d1={10:1.2,20:2.3,40:5.6,50:1.2}

>>> print(d1, type(d1), id(d1))-----{10: 1.2, 20: 2.3, 40: 5.6, 50: 1.2} <class 'dict'> 1228176102528

>>> d2=d1.copy()

>>> print(d2, type(d2), id(d2))----{10: 1.2, 20: 2.3, 40: 5.6, 50: 1.2} <class 'dict'> 1228171857856

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

3) pop ()

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

=>Syntax: dictobj.pop(Key)

=>This Function is used removing (Key,Value) from non-dict object

=>if we call this function on empty dict object we get KeyError

Examples:

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

>>> d1={10:1.2,20:2.3,40:5.6,50:1.2}

>>> print (d1, type(d1), id(d1))----{10: 1.2, 20: 2.3, 40: 5.6, 50: 1.2} <class 'dict'> 1228176103168

>>> d1.pop (20) ---------2.3

>>> print(d1, type(d1), id(d1))------{10: 1.2, 40: 5.6, 50: 1.2} <class 'dict'> 1228176103168

>>> d1.pop(40)-----5.6

>>> print(d1,type(d1), id(d1))---{10: 1.2, 50: 1.2} <class 'dict'> 1228176103168

>>> d1.pop(10)-------1.2

>>> print(d1,type(d1), id(d1))---{50: 1.2} <class 'dict'> 1228176103168

>>> d1.pop(50)------1.2

>>> d1.pop(150)------KeyError: 150

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

4) popitem()

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

=>Syntax: dictobj.popitem()

=>This Function is used removing last entry of (Key,Value) from non-dict object

=>if we call this function on empty dict object we get KeyError

Examples:

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

>>> d1={10:1.2,20:2.3,40:5.6,50:1.2}

>>> print(d1,type(d1), id(d1))--{10: 1.2, 20: 2.3, 40: 5.6, 50: 1.2} <class 'dict'> 1228171857920

>>> d1.popitem()---(50, 1.2)

>>> print(d1,type(d1), id(d1))---{10: 1.2, 20: 2.3, 40: 5.6} <class 'dict'> 1228171857920

>>> d1. popitem()---(40, 5.6)

>>> print (d1, type(d1), id(d1))---{10: 1.2, 20: 2.3} <class 'dict'> 1228171857920

>>> d1. popitem()--(20, 2.3)

>>> print(d1,type(d1), id(d1))--{10: 1.2} <class 'dict'> 1228171857920

>>> d1.popitem()---(10, 1.2)

>>> print(d1,type(d1), id(d1))--{} <class 'dict'> 1228171857920

>>> d1. popitem()----KeyError: 'popitem(): dictionary is empty'

>>> {}.popitem()-----KeyError: 'popitem(): dictionary is empty'

>>> dict(). popitem()---KeyError: 'popitem(): dictionary is empty'

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

5) keys ()

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

=>Syntax: Varname=dictobj.keys()

(OR)

dictobj.keys()

=>This Function is used for obtaining values of Key.

Examples:

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

>>> d1={10:"Python",20:"Data Sci",30:"Django",40:"Java"}

>>> print(d1,type(d1))-----------{10: 'Python', 20: 'Data Sci', 30: 'Django', 40: 'Java'} <class 'dict'>

>>> d1.keys()------------dict\_keys([10, 20, 30, 40])

>>> kvs=d1.keys()

>>> print(kvs)--------------dict\_keys([10, 20, 30, 40])

>>> for k in kvs:

... print(k)

...

10

20

30

40

>>> for k in d1.keys():

... print(k)

...

10

20

30

40

NOTE:

>>> d1={10:"Python",20:"Data Sci",30:"Django",40:"Java"}

>>> print(d1,type(d1))------------{10: 'Python', 20: 'Data Sci', 30: 'Django', 40: 'Java'} <class 'dict'>

>>> k=d1.keys()

>>> print(k)------------------dict\_keys([10, 20, 30, 40])

>>> print(k,type(k))----------dict\_keys([10, 20, 30, 40]) <class 'dict\_keys'>

>>> l=list(k)

>>> print(l,type(l))--------------[10, 20, 30, 40] <class 'list'>

>>> print(l[0])-------------10

----------------------------OR-----------------------------------

>>> d1={10:"Python",20:"Data Sci",30:"Django",40:"Java"}

>>> print(d1,type(d1))--------{10: 'Python', 20: 'Data Sci', 30: 'Django', 40: 'Java'} <class 'dict'>

>>> list(d1.keys())[0]------------10

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

6) values()

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

Syntax: Varname=dictobj.values()

(OR)

dictobj.values()

=>This Function is used for obtaining Values of Value.

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

Examples:

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

>>> d1={10:"Python",20:"Data Sci",30:"Django",40:"Java"}

>>> print(d1,type(d1))---------------{10: 'Python', 20: 'Data Sci', 30: 'Django', 40: 'Java'} <class 'dict'>

>>> d1.values()------------dict\_values(['Python', 'Data Sci', 'Django', 'Java'])

>>> vs=d1.values()

>>> print(vs)---------------dict\_values(['Python', 'Data Sci', 'Django', 'Java'])

>>> for v in vs:

... print(v)

...

Python

Data Sci

Django

Java

>>> for v in d1.values():

... print(v)

Python

Data Sci

Django

Java

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

7) items ()

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

Syntax: - varname=dictobj.items()

(OR)

dictobj.items()

=>This Function is used for obtaing (Key,Value) from dict object in the form of list of tuples.

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

Examples

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

>>> d1= {10:"Python",20:"Data Sci",30:"Django",40:"Java"}

>>> print(d1, type(d1))----------{10: 'Python', 20: 'Data Sci', 30: 'Django', 40: 'Java'} <class 'dict'>

>>> d1. items()---dict\_items([(10, 'Python'), (20, 'Data Sci'), (30, 'Django'), (40, 'Java')])

>>> kv=d1.items()

>>> print(kv)--dict\_items([(10, 'Python'), (20, 'Data Sci'), (30, 'Django'), (40, 'Java')])

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

>>> for x in kv:

... print(x)

...

(10, 'Python')

(20, 'Data Sci')

(30, 'Django')

(40, 'Java')

>>> for k,v in kv:

... print(k,v)

...

10 Python

20 Data Sci

30 Django

40 Java

>>> for k,v in kv:

... print(k,"-->“, v)

...

10 --> Python

20 --> Data Sci

30 --> Django

40 --> Java

>>> for k,v in d1.items():

... print(k,"-->“, v)

...

10 --> Python

20 --> Data Sci

30 --> Django

40 --> Java

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

8) update()

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

Examples:

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

>>> d1={10:1.2,20:3.4}

>>> d2={30:1.5,40:5.6}

>>> print (d1, type(d1))--------------{10: 1.2, 20: 3.4} <class 'dict'>

>>> print(d2,type(d2))-------------{30: 1.5, 40: 5.6} <class 'dict'>

>>> d1. update(d2)

>>> print(d1,type(d1))---------{10: 1.2, 20: 3.4, 30: 1.5, 40: 5.6} <class 'dict'>

>>> print(d2,type(d2))---------{30: 1.5, 40: 5.6} <class 'dict'>

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

>>> d1={10:1.2,20:3.4}

>>> d2={10:6.5,20:7.6}

>>> print(d1,type(d1))-------------{10: 1.2, 20: 3.4} <class 'dict'>

>>> print(d2,type(d2))------------{10: 6.5, 20: 7.6} <class 'dict'>

>>> d1.update(d2)

>>> print(d1,type(d1))--------------{10: 6.5, 20: 7.6} <class 'dict'>

>>> print(d2,type(d2))--------------{10: 6.5, 20: 7.6} <class 'dict'>

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

>>> d1={10:1.2,20:3.4}

>>> d2={30:1.5,10:15.6}

>>> print(d1,type(d1))------------{10: 1.2, 20: 3.4} <class 'dict'>

>>> print(d2,type(d2))------------{30: 1.5, 10: 15.6} <class 'dict'>

>>> d1.update(d2)

>>> print(d1,type(d1))-----------{10: 15.6, 20: 3.4, 30: 1.5} <class 'dict'>

>>> print(d2,type(d2))---------{30: 1.5, 10: 15.6} <class 'dict'>

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

9) get()

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

=>Syntax: Varname=dictobj.get(Key)

(OR)

dictobj.get(Key)

=>This Function is used for finding value of Value by passing Value of Key

=>If Value of Key does not exist then we get None.

Examples:

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

>>> d1={"TS":"HYD","AP":"AMVT","KAR":"BANG","TAMIL":"CHE"}

>>> print(d1) ----------{'TS': 'HYD', 'AP': 'AMVT', 'KAR': 'BANG', 'TAMIL': 'CHE'}

>>> d1["TS"]--------'HYD'

>>> d1["TAMIL”] ----------'CHE'

>>> d1["AMPT”] ---------KeyError: 'AMPT'

>>> d1.get("AP")-----------'AMVT'

>>> d1.get("KAR")-----------'BANG'

>>> d1.get("SRN")--------

>>> print(d1.get("SRN"))------None

----------

NOTE:

----------

We can get Value of Value by passing Key by using the syntax also.

Syntax: varname=dictobj[Key]

=>Here if Value of Key does not Exist then we get KeyError

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

MISC Examples:

----------------------------------------------------------------------->>> d1= {10:"Python",20:"Data Sci",30:"Django",40:"Java"}

>>> print(d1, type(d1))--------{10: 'Python', 20: 'Data Sci', 30: 'Django', 40: 'Java'} <class 'dict'>

>>> kvs=d1.items()

>>> print (kvs, type(kvs))-------dict\_items([(10, 'Python'), (20, 'Data Sci'), (30, 'Django'), (40, 'Java')]) <class 'dict\_items'>

>>> l1=list(kvs)

>>> print(l1,type(l1))---[(10, 'Python'), (20, 'Data Sci'), (30, 'Django'), (40, 'Java')] <class 'list'>

>>> l1[0]------------(10, 'Python')

>>> l1[1]----------(20, 'Data Sci')

>>> l1[2] ----------(30, 'Django')

>>> l1[-1]---------(40, 'Java')

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

>>> d1={10:"Python",20:"Data Sci",30:"Django",40:"Java"}

>>> print(d1,type(d1))---{10: 'Python', 20: 'Data Sci', 30: 'Django', 40: 'Java'} <class 'dict'>

>>> lst=list(d1.items())

>>> print(lst)------[(10, 'Python'), (20, 'Data Sci'), (30, 'Django'), (40, 'Java')]

-----------------------------------------------------------------------Most Useful case:

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

>>> l1=[(10,"Rizwan"),(20,"Rossum"),(30,"Rajesh"),(40,"Ranjit")]

>>> print(l1,type(l1))------[(10, 'Rizwan'), (20, 'Rossum'), (30, 'Rajesh'), (40, 'Ranjit')] <class 'list'>

>>> d1=dict(l1)

>>> print(d1,type(d1))----{10: 'Rizwan', 20: 'Rossum', 30: 'Rajesh', 40: 'Ranjit'} <class 'dict'>

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

Most Useful case

> l1=(10,20,30,40)

>>> l2=(1.2,2.3,4.5,1.5)

>>> d1=dict(zip(l1,l2))

>>> print(d1)-------------{10: 1.2, 20: 2.3, 30: 4.5, 40: 1.5}

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

>>> l1=(10,20,30,40)

>>> l2=(1.2,2.3)

>>> d=dict(zip(l1,l2))

>>> print(d,type(d))--------------{10: 1.2, 20: 2.3} <class 'dict'>

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

>>> print(d1, type(d1))

{10: ['C', 'CPP'], 20: {'PYTHON': {'Core': 'GUI', 'ADV': 'OOPS'}, 30: 'OUCET'}} <class 'dict'>

>>>

>>> for k,v in d1.items():

... print(k,"--->“, v)

...

10 ---> ['C', 'CPP']

20 ---> {'PYTHON': {'Core': 'GUI', 'ADV': 'OOPS'}, 30: 'OUCET'}

>>> len(d1)-----2

>>> d1[20]

{'PYTHON': {'Core': 'GUI', 'ADV': 'OOPS'}, 30: 'OUCET'}

>>> type(d1[20])-----------<class 'dict'>

>>> len(d1[20])------------2

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

>> d1[100]="Guido"

>>> print(d1,type(d1),len(d1), id(d1))-----{100: 'Guido', 101: 'Ritche', 102: 'Travis', 103: 'MCKinney'} <class 'dict'> 4 2299637840384

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

NoneType data type

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

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

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

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

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

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

Examples:

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

>>> a=None

>>> print(a,type(a))------------None <class 'NoneType'>

>>> a=NoneType()---------NameError: name 'NoneType' is not defined

>>> l1=[]

>>> print(l1.clear())------------None

>>> s1=set()

>>> print(s1.clear())---------None

>>> d1=dict()

>>> print(d1.clear())-----------None

>>> d1={10:1.2,20:3.4}

>>> print(d1.get(100))---------None

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