==

History Development of Python

==

=>The Python Programming Language Conceived in the Year 1980.
=>The Python Programming Language Development or Implementation began in the year 1989
=>The Python Programming Language Officially released in the year 1991 Feb 20th
=>The Python Programming Language developed by "GUIDO VAN ROSSUM".
=>The Python Programming Language Developed at Centrum Wickenden Informatica (CWI) Institute in Nether Lands.
=>The ABC Programming Language is the predecessor of Python Programming Language

==

Versions of Pythons

==

=>Python Programming Language contains 3 types of Versions. They are

1) Python 1.x here 1 is called Major Versions and x is called minor Version and x represents 0 1 2 3 4 5 6.......so on
2) Python 2.x here 2 is called Major Versions and x is called minor Version and x represents 0 1 2 3 4 5 6, 7 only

3) Python 3.x here 3 is called Major Versions and x is called minor Version and x represents 0 1 2 3 4 5 6, 7 8 9 10 11 12
=>The Python Software maintained by a non-commercial organization called "Python Software Foundation (PSF) "

=>To download python software, we must visitwww.python.org

===

Real Time Applications / Projects Developed by Python Programming

==

=>By using Python Programming, as on today we can develop 22+ Applications / Projects / Products. They are

1.Web Applications (Web Sites)
Java Programming lang---- Servlets, JSP, etc—I

C#.net Programming Lang---ASP.Net------II

Python Programming Lang--- Django, Pyramid, Bottle, Flask

2.Games Applications Development
3.Artificial Intelligence Application

=> Machine Learning (ML)
=>Deep Learning (DL)
=>Tensor Flow

1. Desktop GUI Applications
5. Image Processing Applications.
6. Audio and Video Based Applications
7. Business Applications Development.
8. Text processing Based Applications
9. Web Scrapping / Web Harvesting Applications
10. Data Visulation
11. Complex Math Calculation
12. Scientific Applications
13. Software Development
14. Operating System Installers 15) CAD / CAM Based Applications 16) IOT Based Applications.
17) Embedded Applications
18) Console Based Persistency Applications
19) Language Development
20) Automation Testing
21) Data Analysis and Data Analytics
22) Education Programs----etc

===

Features of Python

===

=>Features of a language are nothing but Services or Facilities provided by language developers and they are available in the languages and they are used by Language programmers for developing Real Time Applications.
=>Python Programming Provides 11 features. They are

1. Simple
2. Freeware and Open Source
3. Platform Independent Lang
4. Dynamically Typed
5. Interpreted
6. High Level
7. Procedure Oriented and Object Oriented
8. Robust (Strong)
9. Extensible
10.Embedded
11. Supports for Third Party APIs like NumPy, Pandas, SciPy, Scikit, matplotlib...etc

===

1. Simple:

===

=>Python is one of the Simple Programming Language because of 3 Important Technical factors.

1) Python Programming Provides Rich set of APIs. So that Python Programmers can re-use the pre-defined code without writing our own logic.

Def. of API:(Application Programming Interface):

=>An API is a collection MODULES. =>A Module is a collection of Variables,
Functions and Class Names.
=>Learning about python is nothing but learning about Modules.

Examples: math, class math, random, calendar, os, re, threading, pickle, etc

----

2) Python Programming lang provides In-built facility called "Garbage Collector and whose role is to collect un-used memory space and improves the performance of Python Based Applications.

Def. of Garbage Collector:

=>A Garbage Collector is one of the background python Program which is running in background of our regular python Program and whose is to collect un-used memory space and improves the performance of Python Based Applications.

3) Python Programming Language provides User-Friendly Syntaxes. So that Python programmer can develop error-free program in limited span of time.

===

2. Freeware and Open Source

===

=>Python is one of the Freeware because Python Software can be Freely from www.python,org.
=>In General, if any software downloaded freely then it is called Freeware.

=>Open Source:

=>The standard name of Python Software is "CPYTHON".
=>An Open-Source Software is one, which is customized by Various Software Vendors for their in-house tool's development.
=>Now Python is also one of the Open-Source Software because Python software Customized by Various Software Vendors and used as in-house tools. These customized software's of python are called "Python Distributions".
=>Some of the "Python Distributions" are

1.Jpython or Jython------>Used for running Java Based Applications.

2.Iron Python or Ipython--->Used for running C#.net Based Applications
3. Micro Python------->Used to develop Micro Controller Based Applications
4. Ruby Python----->Used to run Ruby Based Applications
5. Anaconda Python--->Used to run and develop Bigdata / Hadoop Based Applications.
6. Stockless Python----->Used to develop Concurrency Applications....etc

==

3. Platform Independent Lang

==

Concept: -

=>A lang is said to be Platform Independent Lang if and only if whose application runs on every OS with having any restriction on memory space".

Language Comparison:

=>C, CPP are treated Platform Dependent languages because their data types are taking different memory space on different OSes.

=>Java is treated Platform Independent Lang bcoz Java Data Types takes same memory space on all Types of OSes but they take only single Value. To store multiple values of same type or different type, we must Classes and Object in Java. Java Objects Can Store Multiple values of same type or different type or both types with limited Number of Values.

Python---Platform Independent Lang

=>In Python Programming, all values are stored in the form objects with un-limited number of values and memory space restrictions and hence python object are Platform Independent

===

4. Dynamically Typed

=>In This context, we have two types programming languages. They are

1. Static Typed Programming Languages
2. Dynamically Typed Programming Languages

----

1. Static Typed Programming Languages

=>In This programming Languages, The Programmer must write Variable Declaration with Suitable Data type and Variable Name. Otherwise, we get Error.
Examples: C, CPP, Java. .Net...etc Example:

```
                     jshell> int m=10
   m ==> 10

                   jshell> int n=20
   n ==> 20
                   jshell> int k=m+n
                         k ==> 30
```

----

2. Dynamically Typed Programming Languages

=>In This programming Languages, The Programmer need not write Variables Declaration. In other words, programmers need not use data types explicitly. Internally the Python execution environment decides the type of data type based on the value assigned by Programmer.

Examples: Python

Examples: Write the instructions to write sum of two numbers in Python Lang

```
            >>> a=100
            >>> b=23.45
            >>> c=a+b
            >>> print (a, type(a)) -------------------  100    <cl
            >>> print (b, type(b)) --------------------- 23.45
<class 'float'>
            >>> print (c, type(c)) --------------------       123.45 <cl
```

Note: - In python Programming, All Values are stored in the form of Objects and to creator objects there must exist a class. Otherwise, we can't create object.

===

5. Interpreted Programming

===

=>When we develop any python program, we must give some file name with an extension .py (File Name.py).
=>When we execute python program, two process taken place internally

a) Compilation Process

b) Execution Process.

=>In COMPILATION PROCESS, the python Source Code submitted to Python Compiler and It reads the source Code, Check for errors by verifying syntaxes and if no errors found then Python Compiler Converts into Intermediate Code called BYTE CODE with an extension .pyc (Filename). If errors found in source code, then we error displayed on the console.
=>In EXECUTION PROCESS, The PVM reads the Python Intermediate Code (Byte Code) and Line by Line and Converted into Machine Under stable Code (Executable or binary Code) and it is read by OS and Processer and finally Gives Result.
=>Hence In Python Program execution, Compilation Process and Execution Process is taking place Line by Line conversion and it is one of the Interpretation Based Programming Language. --
----

Definition of PVM (Python Virtual Machine)

=>PVM is one program in Python Software and whose role is to read LINE by LINE of Byte Code and Converted into Machine Under stable Code (Executable or binary Code)

==

6. High Level Programming

==

=>In general, we have two types of Programming languages. They are

a) Low Level Programming Languages.

b) High Level Programming Languages.

----
a) Low Level Programming Languages:

=>In These Programming Languages, we represent the data in lower-level data like Binary, Octal and Hexa decimal and This type data is not by default untestable by Programmers and end users.

Examples: - a=0b1111110000111101010---binary data
b=0o23-----octal
c=0xface----Hexa Decimal
d=0xBEE e=0xacc

----
b) High Level Programming Languages.

=>In These Programming Languages, even we represent the data in lower-level data like Binary, Octal and Hexa decimal, the High-Level Programming Languages automatically converts into Decimal number System data, which is untestable by Programmers and end-users and python is one High Level Programming Language.

Example: Python

=====================================
7. Robust (Strong)
=====================================

=>Python is one of Robust (Strong) because of "Exception Handling".
=>Exception: - Runtime Errors of the Program are called Exceptions
=>Exception by default generates Technical Error Messages
=>Exception Handling: - The process of converting Technical Error Messages into User Friendly Error Messages is called Exception Handling.
=>If the Python program uses Exception Handling the Python program is Robust.

===================================
8. Extensible
===================================

=>The Python programming giving its programming facilities to other languages and hence Python is one of the extensible Programming languages.

==
9. Embedded
==

=>Python Programming can call other languages coding segments for fastest execution

Example: Python code can call C programming Code.

==
10.Extensive Support for Third Party APIs
==

=>As Python Libraries / API can do many tasks and Operations and unable perform complex operations and to solve such complex operations more easily and quickly we use Third Party APIs such as

1) NumPy----Numerical calculations
2) pandas---Analysis too
3) matplotlib-----Data Visualization
4) SciPy
5) scikit

===========================*&*================================
==
Data Types in Python
==

=>The purpose of Data Types is that " To allocate sufficient amount of memory space for storing input values or literals in main memory of the computer ".
=>In Python Programming, we have 14 data types and They classified in 6 types.

I) Fundamental Data Types
   1. int
   2. float
   3. bool
   4. complex
II) Sequence Category Data Types
   1. str
   2. bytes
   3. byte array
   4. range
III) List Category Data Types (Collection Data Types or Data Structures)
   1. list
   2. tuple
IV) Set Category Data Types (Collection Data Types or Data Structures)

1. set
2. frozen set

V) Dict Category Data Types (Collection Data Types or Data Structures)

1. dict

VI) None Type Category Data Type

1. None Type

==
I) Fundamental Data Types
==

=>The purpose of Fundamental 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

===
1. int
===

=>'int' is one of the pre-defined classes and treated as Fundamental data Type.
=>The purpose of 'int' data type is that " To store single Integer data or Whole Numbers or Integral data‟
=>Examples:

```
--------------------------------------------------------------------------
Python Instructions                                                 Output
--------------------------------------------------------------------------
>>> a=100
>>> print(a)-------------------------------------------------     100
>>> type(a)-------------------------------------------------- <class
'int'>
>>> id(a)----------------------------------------------------
1765066345808
>>> print (a, type(a), id(a)) ---------------------------    100 <class
'int'> 1765066345808
------------------------------------------------
>>> a=12
>>> b=13
>>> c=a+b
>>> print (a, type(a)) -------------------------------12 <class 'int'>
>>> print (b, type(b)) ----------------------------- 13 <class 'int'>
>>> print (c, type(c)) -------------------------------- 25 <class
'int'>
>>> print(id(a), id(b), id(c)) ---------------------- 1765066342992
1765066343024 1765066343408
--------------------------------------------------------------------------
```

=>with 'int' data type, we can store different Number Systems Data.

=>In programming Language including Python, we have 4 Types of Number Systems. They are

1. Decimal Number System
2. Binary Number System
3. Octal Number System 4. Hexa Decimal Number System.

---

1. Decimal Number System:

=>It is one of the default number systems.
=>This Number System contains the following digits

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

=>hence Base 10 Literals are called Integer Data

---

2. Binary Number System

=>Binary Number System untestable by OS and Processor during Program execution.
=>This Number System contains the following digits

```
              Digits: 0, 1------Total Digits----2
              Base: 2
```

=>hence Base 2 Literals are called Binary Data
=>In python Programming, to store Binary Number System data, The binary data must be preceded with a letter '0b' or '0B'.

```
----------------
Syntax:                Varname=0b Binary data
----------------                    (OR)
                              varname=0B Binary data
```

=>Internally, the binary data automatically converted into Decimal Number System Data. **Examples**:

```
-------------------
>>> a=0b1010
>>> print (a, type(a)) ------------------ 10 <class 'int'>
>>> b=0B1111
>>> print (b, type(b)) ------------------15 <class 'int'>
```

Examples:

```
----------------------
>>> c=0b100+0B101
>>> print (c, type(c)) ----------------9 <class 'int'>
>>> bin (15) ---------------'0b1111'
>>> bin (10) --------------'0b1010'
>>> bin (4) --------------'0b100'
>>> bin (5) --------------'0b101'
>>> a=0b1010102----------------Syntax Error: invalid digit '2' in
binary literal
```

---

3. Octal Number System

=> Octal Number System untestable by Micro Processor Kits during Program execution.

```
=>This Number System contains the following digits
              Digits: 0 1 2 3   4 5 6 7 ------Total Digits----8
              Base: 8
=>hence Base 8 Literals are called Octal Data
=>In python Programming, to store Octal Number System data, The Octal
data must be preceded with a letter '0o' or '0O'.

=>Internally, The Octal data automatically converted into Decimal
Number System Data.
```

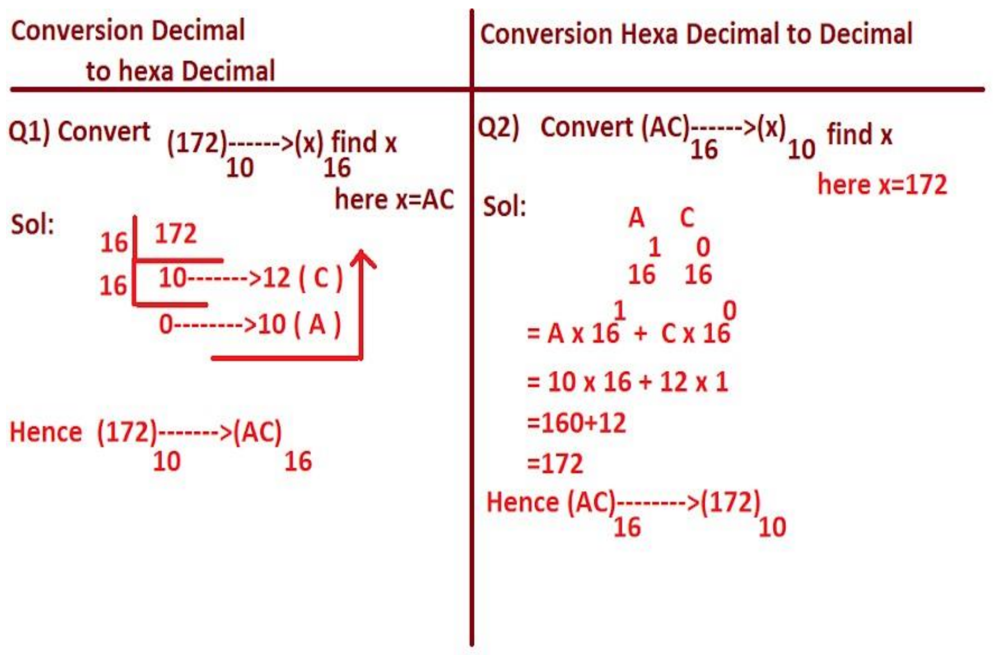
Examples:

```
-----------------
>>> a=0o23
>>> print (a, type(a)) ----------------------19 <class 'int'> >>>
a=0o123
>>> print (a, type(a)) -----------------------83 <class 'int'>
>>> b=0o23+0O123
>>> print (b, type(b)) --------------------102 <class 'int'>
>>> oct (19) ------------------------------ '0o23'
>>> oct (83) --------------------------------- '0o123'
>>> a=0o18-------------------------Syntax Error: invalid digit '8' in
octal literal
-----------------------------------------------------------------------
```

4. Hexa Decimal Number System.

```
------------------------------------------------------
```

Int Data Types


Conversion Decimal to hexa Decimal
Q1) Convert $(172)_{10}$------>$(x)_{16}$ find x
here x=AC
Sol:
16 | 172
16 | 10------->12 (C)
0-------->10 (A)
Hence $(172)_{10}$------->$(AC)_{16}$
Conversion Hexa Decimal to Decimal
Q2) Convert $(AC)_{16}$------>$(x)_{10}$ find x
here x=172
Sol:
A C
16^1 16^0
= A x 16^1 + C x 16^0
= 10 x 16 + 12 x 1
=160+12
=172
Hence $(AC)_{16}$-------->$(172)_{10}$

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

2. float

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

=>'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 "

=>This data also supports Scientific Notation. Scientific Notation is one of the alternative notations for Real Constant values or FloatingPoint Values. The advantage of Scientific Notation is that to store real constant values in less memory space

=>float data type does not supper Binary, Octal and Hexa Decimal Number System Data. It supports only default number called Decimal Number System.

Examples:

```
>>> a=12.34
>>> print (a, type(a), id(a)) -----------------------------12.34 <class
'float'> 2569509390480
>>> a=0.99
>>> print (a, type(a), id(a)) -----------------------------0.99 <class
'float'> 2569509394896
>>> a=3e2
>>> print (a, type(a)) ---------------------------------------300.0
```

```
<class 'float'>
>>> b=10e-2
>>> print(type(b)) -------------------0.1 <class 'float'> >>>
a=12.34
>>> print(type(c)) -------------- 1e-50    <class 'float'> ------------
------------------------
```

Examples:

```
-------------------------------------
>>> a=0b1111.0b1010------------Syntax Error: invalid decimal literal
>>> b=0o123.0o345------------Syntax Error: invalid decimal literal
>>> c=0xFace.0xBEE-----------Syntax Error: invalid decimal literal
>>> d=0b1010.0o123----------Syntax Error: invalid decimal literal
```

==

3. bool

==

=>'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 or False Values (Logical values) ".
=>Here True False are keywords and treated as Values for bool data type.
=>Internally True is taken as 1 and False is Taken as 0.

Examples:

```
---------------------
>>> a=True
>>> print (a, type(a)) --------------------True <class 'bool'>
>>> b=False
>>> print(type(b)) -----------------False <class 'bool'>
----------------------------------
>>> c=true---------------------Name Error: name 'true' is not defined.
Did you mean: 'True'?
>>> d=false-------------------Name Error: name 'false' is not defined.
Did you mean: 'False'?
----------------------------------------------
```

Examples:

```
-----------------------------------------------
>>> a=True
>>> b=False
>>> print (a, type(a)) -----------------------True <class 'bool'>
>>> print (b, type(b)) ----------------------False <class 'bool'>
>>> print(a+b) ---------------------------- 1
>>> print(a*b) ---------------------------- 0
>>> print (True+True+False*True) -------------2
>>> print (True-True+False) ------------------0
>>> print (4*True False*3) ------------------4
>>> c=0b1111+True*0.0
>>> print(c)---------------15.0
>>> print(type(c)) -----------<class 'float'>
>>> print(0xA+True+0b1010) -------------21
>>> print (2*True True*3) -----------------5
>>> print(2+True*True+2) ------------5
```

```
----------------------------------------------------
>>> print (True//True) --------------1
>>> print (False//True) -------------0
>>> print (False//False) ------------ZeroDivisionError: integer
division or modulo by zero
>>> print (True//False) ------------ZeroDivisionError: integer division
or modulo by zero
>>> print(23/True) -----------23.0
>>> print(0b1010/True) --------10.0
```

===

4. complex

===

=>'complex' is one of the pre-defined classes and treated as Fundamental data Type.
=>The purpose of complex data type is that "To store complex or imaginary data ".
=>The general format complex numbers are given bellow.

a+bj   or a-bj

=>Here 'a' is called Real Part and 'b' is called Imaginary Part
=>Here 'j' is representing sqrt (-1)
=>Internally the values of Real and Imaginary are treated as floating point values.
=>To extract or obtain the real and imaginary parts of Complex Object, we use two pre-defined attributes / Variables present in complex object
a) real
b) image
=>Syntax: complex. real------>Gives Real part of Complex Object.
complexobj. image----->Gives Imaginary part of Complex Object.

=>On complex data, we can perform many operations like addition, subtraction, multiplication etc

----

Examples:

```
------------------------
>>> a=2+3j
>>> print (a, type(a)) --------------------------(2+3j) <class
'complex'>
>>> b=3-4j
>>> print(type(b)) --------------------------(3-4j) <class 'complex'>
>>> c=2.3+4.5j
>>> print(type(c)) ---------------------------(2.3+4.5j) <class
'complex'>
>>> d=-2.3-4.5j
>>> print (d, type(d)) -----------------------(-2.3-4.5j) <class
'complex'>
>>> e=0+3.4j
>>> print (e, type(e)) -----------------------3.4j <class 'complex'>
-------------------------------------
>>> a=10+2.3j
>>> print (a. real) -------------------------   10.0
>>> print (a. imag) -------------------    2.3
>>> b=-2.3-3.4j
>>> print (b. real) ------------------------   -2.3
>>> print (b. imag) ----------------------   -3.4
>>> c=0+3j
>>> print (c. real) --------------------------   0.0
>>> print (c. imag) --------------------------- 3.0
------------------------------------------------------------------------
------------------------------------------
>>> a=2+3j
>>> b=3+4j
>>> print(a+b) -------------------(5+7j)
>>> print(a-b) -------------------(-1-1j)
>>> print(a*b) ------------------ (-6+17j)
------------------------------------------------------------------------
-------
>>> print(10+2+3j+True+1.2) -------------(14.2+3j)
```

====================================**====================================
===

II) Sequence Category Data Types

===

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

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

1. str
2. bytes
3. bytearray
4. range

==

1. str

== **Index**

==

1. str

==

=>'str' is one of the pre-defined classes and treated as Sequence Data Type.
=>The purpose of str data type is that "To store String or Text Data ".

=>Definition of String:

=>A String is a collection or Sequence of chars enclosed within either Double or Single Quotes or triple Double or triple Single Quotes.

=>Types of Strings:

=>In Python Programming, we have two types of Strings. They are
1. Single Line String Data
2. Multi Line String Data

-- 1.
Single Line String Data:

=>Single Line String Data must be enclosed within Single or Double Quotes.

```
=>Syntax: -     strobj="Single Line String Data"
                                     (OR)
=>Syntax: -     strobj='Single Line String Data'
```

Examples:

```
>>> s1="Python Programming Lang"
>>> print (s1, type(s1)) --------------Python Programming Lang <class
'str'>
>>> s2='Python Prog Lang'
>>> print (s2, type(s2)) ------------------Python Prog Lang <class
'str'>
>>> s3="K"
>>> print (s3, type(s3)) ----------------K <class 'str'>
>>> s4='V'
>>> print (s4, type(s4)) --------------V <class 'str'>
>>> s5="12345"
>>> print (s5, type(s5)) ---------------12345 <class 'str'>
>>> s6="Python3.10"
>>> print (s6, type(s6)) -------------------Python3.10 <class 'str'>
--------------------------------------------------
>>> addr1="Guido van Rossum

                             Syntax Error: unterminated string literal
(detected at line 1)

>>> addr2='Guido van Rossum

                             Syntax Error: unterminated string literal
(detected at line 1)
```

Note: - Single Quotes or double Quotes are used for Storing Single Line String Data but used for Multi Liner String Data.

----
2. Multi Line String Data:

=>Multi Line String Data must be enclosed within Triple Single or Double Quotes.

```
=>Syntax: -     strobj=" " "String Data-1
                               String data-2
                         ------------------
           String Data-n " " "
                               (OR)
=>Syntax: -     strobj=' ' 'String Data-1
                               String data-2
                         ------------------
           String Data-n ' ' '
```

Examples:

```
>>> addr1="""Guido van Rossum
... 3-4 Hill Side
... Python Software Foundation
... Nether lands-56 """
>>> print (addr1, type(addr1))
                                    Guido van Rossum
                                    3-4 Hill Side
                                    Python Software Foundation
                            Nether lands-56 <class 'str'> ------------
--------------------------------------------
>>> addr2=' ' 'James Gosling
... 23-4, Red Sea Side
... Sun Micro System
... USA-45 '''
>>> print (addr2, type(addr2))
                                    James Gosling
                                    23-4, Red Sea Side
                                    Sun Micro System
                                    USA-45 <class 'str'>
---------------------------------------------------------------------
>>> addr3=" " "Java Programming" " "
>>> addr4=' ' 'Python Programming' ' '
>>> print (addr3, type(addr3)) ------------------ Java Programming
<class 'str'>
>>> print (addr4, type(addr4)) --------------------Python Programming
<class 'str'>
>>> c1=' ' 'H ' ' '
>>> print (c1, type(c1)) ------------------ H <class 'str'>
>>> c1=" " " H " " "
>>> print (c1, type(c1)) ------------------H <class 'str'>
```

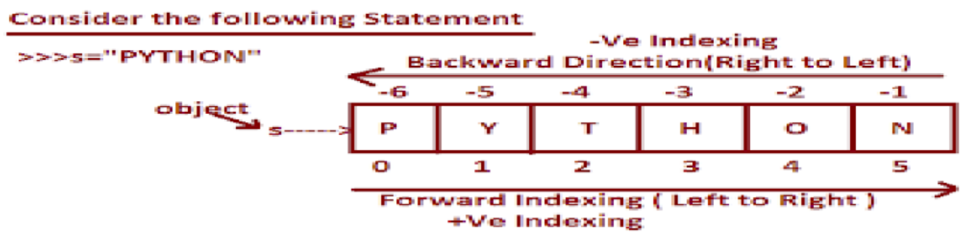
===

String Memory Management

===

=>We store String data in the object of main memory, internally it is stored in the form of two types. They are
1) With Forward Indexing
2) Backward Indexing

=>In Forward Indexing, Index starts from left to right with 0 to NumberValues-1 Indexes. (Also called +ve Indexing)
=>In Backward Indexing, Index starts from Right to Left -1 to Number Values Indexes. (Also called -ve Indexing)


Consider the following Statement
>>>s="PYTHON"
-Ve Indexing
Backward Direction(Right to Left)
-6 -5 -4 -3 -2 -1
object
s----->
P Y T H O N
0 1 2 3 4 5
Forward Indexing (Left to Right)
+Ve Indexing

==

Operations on str data

==

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

1. Indexing
2. Slicing

----

1. Indexing

=>The process of obtaining a character from a given strobj by passing valid index is called Indexing. =>Syntax: - strobj [index]
=>Here Index can be either +ve or -ve
=>if the value of Index is valid then we get Corresponding Character
=>if the value of Index is invalid then we get "Index Error" (It is called one type of error) ----------------

Example:

```
>> s="PYTHON"
>>> s [3] -----------------------'H'
>>> s [-3] ---------------------'H'
>>> s [-1] ---------------------'N'
>>> s [0] -----------------------'P'
>>> s [-6] -------------------'P'
>>> s [-5] -------------------'Y'
>>> s [3] ------------------------'H'
>>> s [13] -----------------Index Error: string index out of range
>>> s [-13] ----------------Index Error: string index out of range
```

```
2. Slicing:          strobj [begin: End] -----------------1
                            strobj [: End] --------------------2
                            strobj [Begin:] ---------------------3
                             strobj [:] ----------------------------4
                            strobj [Begin: End: Step] ---------5
```

=>The process of obtaining range of characters or sub string from Given String is called Slicing.
=>Slicing operation can be performed with 5 syntaxes.

Syntax1: - strobj [Begin: End]
=>This Syntax obtains range of characters or sub string from given str object from BEGIN INDEX to END INDEX-1 provided BEGIN<END otherwise we never get any output.

Examples:

```
>>> s="PYTHON"
>>> s [0:4] -----------------------------------'PYTH'
>>> s [0:3] ----------------------------------'PYT'
>>> s [2:4] ----------------------------------'TH'
>>> s [3:6] ---------------------------------'HON'
>>> s [2:6] ----------------------------------'THON'

>>>
```

```
>>> s [-6: -2] ------------------------------------'PYTH'
>>> s [-3: -1] ------------------------------'HO'
>>> s [-5: -2] -----------------------'YTH' s [6:0]
    ------------------------- ''(no result)
>>> s [0:6] ------------------------'PYTHON'
>>> s [0:5] ----------------------'PYTHO'
>>> s [-6: -1] ---------------------'PYTHO'
#Special Cases:
>>> s [2: -2] ------------------------'TH'
>>> s [1: -4] ------------------------'Y'
>>> s [1: -1] -------------------------'YTHO' >>>
s [-5:4] ------------------------'YTH'
>>> s [-6:6] ------------------------'PYTHON'
>>> s [0:6] ------------------------'PYTHON'
>>> print(s)----------------------PYTHON
-------------------------------------------------------------------------
```

2) Syntax-2: stromb [: End]

=>In This Syntax we are not specifying Begin Index.
=>If we don't specify Begin Index then PVM Takes First Character Index as Begin Index (Either +Vee Index or -ve Index)
=>This syntax also Generates Range of characters from First Character Index (Begin Index) to End Index-1.

Examples:

```
------------------
>>> s="PYTHON"
>>> print(s)--------------------PYTHON
>>> s [: 4] ----------------------'PYTH'
>>> s [: 3] ---------------------'PYT'
>>> s [: 6] --------------------'PYTHON'
>>> s [: -4] -------------------'PY'
>>> s [: -3] -------------------'PYT'
>>> s [: -1] --------------------'PYTHO'
-------------------------------------------------------------------------
-------------------------------------
```

Syntax-3: stromb [Begin:]

=>In This Syntax we are not specifying End Index.
=>If we don't specify End Index then PVM Takes Len(stromb)-1 as End Index OR Last
    Character Index as End Index Value.
=>This syntax also Generates Range of characters from Begin Index to End Index (
    Len(strobj)-1 OR Last Character Index as End Index Value.)

Examples:

```
----------------------
>>> s="PYTHON"
>>> print(s)-----------------------------PYTHON
>>> s [2:] ---------------------------------'THON'
>>> s [3:] --------------------------------'HON'

>>>
```

```
>>> s [0:] ------------------------------'PYTHON'
>>> s [4:] ------------------------------'ON'
>>> s [-3:] -----------------------------'HON'
>>> s [-6:] ------------------------------'PYTHON'
>>> s [-2:] ------------------------------'ON'
>>> s="PYTHON"
>>> s------------------------------------'PYTHON'
>>> s [2:5] ------------------------------'THO'
>>> s [2:] -------------------------------'THON' s
    [:4] -------------------------------'PYTH'
>>> s [2: -1] ----------------------------'THO'
------------------------------------------------------------------------
```

Syntax-4: strobj [:]

=>In This Syntax we are not specifying Begin Index and End Index. =>If we don't specify Begin Index then PVM Takes First Character Index as Begin Index (Either +Ve Index or -ve Index). If we don't specify End Index then PVM Takes Len(strobj)-1 as End Index OR Last Character Index as End Index Value.
=>This Syntax gives complex Str obj data.

Examples:

```
-------------------
>>> s="PYTHON"
>>> s-------------------------------'PYTHON'
>>> s [:] ---------------------------'PYTHON'
>>> s [0:] ----------------------------'PYTHON'
>>> s [:6] ---------------------------'PYTHON'
-------------------------------------------------------------
>>> s="Java Programming invented by James Gosling"
>>> s-----------------------'Java Programming invented by James
Gosling'
>>> s [:] --------------------'Java Programming invented by James
Gosling'
>>> s [10:] --------------'among invented by James Gosling'
>>> s [:11] ----------------------'Java Program'
------------------------------------------------------------------------
```

Syntax-5: strobj [Begin: End: Step]

Rule-1: Here Begin, End and Step can either +Vee or -Vee
Rule-2: - If STEP value is +VE then PVM Takes the characters from Begin Index to End Index-1
in forward Direction by maintaining equal Value of STEP provided BEGIN<END
Rule-3: - If STEP value is -VE then PVM Takes the characters from Begin Index to End Index+1
in backward Direction by maintaining equal Value of STEP provided BEGIN>END
Rule-4: When we extract the for data in forward direction and if the END Index is 0 then we never get any output
Rule-5: When we extract the for data in Backward direction and if the END Index is -1 then we never get any output
>>>

Examples:

```
------------------------
>>> s="PYTHON"
>>> s [0:5:2] 'PTO'
>>> s [0:6:3]
'PH'
>>>
>>>
>>> s="PYTHON" s
    [0:6:1]
```

```
>>>
```

```
'PYTHON'
>>> s [0:6:2]
'PTO'
>>> s [0:6:3]
'PH'
>>> s [2:6:3]
'TN'
>>> s [5:6:3]
'N'
>>> s [0:5: -1]
''
>>> s [5:0: -1] 'NOHTY'
>>> s [4:2: -1]
'OH'
>>> s [-1: -6: -1]
'NOHTY'
>>> s [-6: -1: -1]
''
>>> s [: -1]
'NOHTYP'
>>> s [4: -1]
'OHTYP'
>>> s [: -2] 'NHY'
>>> s [: -3] 'NT'
>>> s="JAVA"
>>> s [: -1]
'AVAJ'
>>> s [:]
'JAVA'
>>> s [:2]
'JV'
>>> s [: -2]
'AA'
>>> s="LIRIL"
>>> s [:] ==s [: -1]
True
>>> "PYTHON"[: -1]
'NOHTYP'
>>> "PYTHON"[: -1] [0:4]
'NOHT'
>>> "PYTHON"[: -1] [0:4:2]
'NH'
>>> s="PYTHON"
>>> s [: 0:1]
''
>>> s [: -1: -1]
''
>>> s="PYTHON"
>>> s [0:234]
'PYTHON'
>>> s [0:234: -1]
''
>>> s [234: -1]
'NOHTYP'
>>>
```

===

Mutable and Immutable objects in python

===

Mutable object:

=>A Mutable object is one which allows to update / modify / change the values at the same address.
=>Examples: List, set, dict... etc

Immutable object:

=>An Immutable object is one which will satisfy the following Properties.
a) Never allows us to modify to at same address
(In Otherwards Values can modified and placed in different Address)
b) Never allows us to do Item Assignment
(or)
does not support item assignment

Examples: int, float, bool, complex, str, tuple, set ...etc

===

2. bytes

===

=>'bytes' is one of the Pre-defined Data Types and treated as Sequence Data Type.
=>The purpose of bytes data type is that " To store Sequence of Numerical Positive Integer
Values ranges from (0,256) "
=>In the python programming, we don't have any symbolic notation for representing the elements by using bytes data type. But we can convert Other Type of Values into Bytes object type by using bytes ()
=> Syntax: byteobject=bytes(object)
=>An object of bytes data type always maintains Insertion Order (Whichever order we enter the data in the same order data will be displayed).
=>On the object of Bytes, we can perform Indexing and Slicing Operations.
=>An object bytes data type belongs to Immutable.

Examples:

```
>>> lst= [10,20,30,40,256]
>>> print(lst,type(lst))-------------------[10, 20, 30, 40, 256] <class
'list'>
>>> b=bytes(lst)-------------------------- ValueError: bytes must be in
range(0, 256)
>>> lst=[0,-10,20,30,40,255]
>>> print(lst,type(lst))-----------------------------[0, -10, 20, 30,
40, 255] <class 'list'>
>>> b=bytes(lst)-----------ValueError: bytes must be in range (0, 256)
>>> lst= [0,10,20,30,40,255]
>>> print (lst, type(lst)) -----------------[0, 10, 20, 30, 40, 255]
<class 'list'>
>>> b=bytes(lst)
>>> print (b, type(b)) ----------------------- b'\x00\n\x14\x1e (\xff'
<class 'bytes'>

>>> for x in b:
...           print(x)
...
                                            0
                                            10
                                            20
                                            30
                                            40
                                            255

>>> print (b [0]) ----------------------100
>>> print (b [1]) -----------------------200
>>> print (b [-11]) -------------------Index Error: index out of range
>>> print (b [-1]) ----------------255
>>> print (b [-3]) ----------------34
>>> x=b [0:4] # Slicing Operation
>>> print(type(x)) ------------b'd\xc8\x0c"' <class 'bytes'>
>>> for v in x: ...
print(v) ...
                      100
                      200
                      12
                      34

>>> for v in b [0:3]:
...           print(v)
...
                      100
                      200
                      12
>>> for v in b [: -1] [0:3]:
...           print(v)
...
                             255
                             0
                             34
>>> for v in b [: -1] [:2]:
...           print(v)
...
                      255
                 34
      200
>>> for v in b:
...           print(v)
...
                      100
                      200
                      12
                      34
                      0
                      255
```

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

===

3. byte array

===

=>'byte array' is one of the Pre-defined Data Types and treated as Sequence Data Type.
=>The purpose of byte array data type is that " To store Sequence of Numerical Positive Integer Values ranges from (0,256) "
=>In the python programming, we don't have any symbolic notation for representing the elements by using byte array data type. But we can convert Other Type of Values into array object type by using byte array ()

=> Syntax: bytearrayobject=bytearray(object)
=>An object of bytearray data type always maintains Insertion Order (Whichever order we enter the data in the same order data will be displayed).
=>On the object of Bytearray, we can perform Indexing and Slicing Operations.
=>An object of bytearray data type belongs to mutable.

Examples:

```
>>> lst=[10,20,30,40,256]
>>> print (lst, type(lst)) -------------------[10, 20, 30, 40, 256]
<class 'list'>
>>> b=byte array(lst)-------------------------- Value Error: bytes must
be in range (0, 256)
>>> lst= [0, -10,20,30,40,255]
>>> print (lst, type(lst)) -----------------------------[0, -10, 20,
30, 40, 255] <class 'list'>
>>> b=byte array(lst)-----------ValueError: bytes must be in range(0,
256)
>>> lst=[0,10,20,30,40,255]
>>> print(lst,type(lst))-----------------[0, 10, 20, 30, 40, 255]
<class 'list'>
>>> b=bytearray(lst)
>>> print(b, type(b))----------------------- b'\x00\n\x14\x1e(\xff'
<class 'bytearray'>

>>> for x in b:
...           print(x)
...
                                            0
                                            10
                                            20
                                            30
                                            40
                                            255

>>> print(b[0])---------------------100
>>> print(b[1])----------------------200
>>> print(b[-11])-------------------IndexError: index out of range
>>> print(b[-1])-----------------255
>>> print(b[-3])-----------------34
>>> x=b[0:4]  # Slicing Operation
>>> print(x,type(x))------------b'd\xc8\x0c"' <class 'bytes'>
>>> for v in x: ...
print(v) ...
                      100
                      200
                      12
                      34

>>> for v in b[0:3]:
...          print(v)
...
                      100
                      200
                      12
>>> for v in b[::-1][0:3]:
...          print(v)
...
                             255
                             0
                             34
>>> for v in b[::-1][::2]:
...          print(v)
...
                      255
                 34
      200
>>> for v in b:
...          print(v)
...
                      100
                      200
                      12
                      34
                      0
                      255

>>> b[0]=123----------------valid >>>
for v in b:
...          print(v)
...
                      123
                      200
                      12
                      34
                      0
                      255
```

===
4. range
===

=>"range" is one of the pre-defined class and treated as Sequence Data Type.
=>The purpose of range data type is that " To store Sequence of Numerical Integer Values by
maintaining equal Interval of value" =>range data type is one of the immutable object
=>on the object of range data type, we can perform Both Indexing and Slicing Operations.
=>range data type contains 3 syntaxes. They are
1) range(value)
2) range (Begin, End)
3) range (Begin, End, Step)

Syntax-1: varname=range(value)

=>This Syntax generates range of values from 0 to value-1 =>Here varname is an object of <class, "range">

Examples:

```
>>> r=range (6)
>>> print (r, type(r)) -------------------range (0, 6) <class 'range'>
>>> for Val in r:
...     print (Val)
                      ...
                      0
                      1
                      2
                      3
                      4
                      5
>>> print(r[0])---------------------0
>>> print(r[-1])--------------------5
>>> for val in r[::-1]:
...     print(val) ...
                                            5
                                            4
                                            3
                                            2
                                            1
                                            0
>>> for val in r[0:3]:
...     print(val) ...
                                            0
                                            1
                                            2
>>> for val in r: ...
print(val) ...
                                            0
                                            1
                                            2
                                            3
```

```
                                             4
                                             5

>>> r[0]=10-------------------------TypeError: 'range' object does not
support item assignment
-----------------------------------------------------------------------
```

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

```
-------------------------
=>This Syntax generates range of values from Begin to End-1
=>Here varname is an object of <class,"range">

Examples:
------------------
>>> r=range(10,21)
>>> print(r,type(r))-----------------range(10, 21) <class 'range'>
>>> for val in r: ...
print(val) ...
                              10
                              11
                              12
                              13
                              14
                              15
                              16
                              17
                              18
                              19
                              20
>>> for val in range(100,106):
...     print(val) ...
                                     100
                                     101
                                     102
                                     103
                                     104
                                     105
>>> for val in range(1000,1006):
...     print(val,end=" ")--------------   1000    1001   1002   1003
1004   1005

=>Syntax-1 and Syntax-2, the default value of Step is 1 which is
nothing but equal Interval of value.
-----------------------------------------------------------------------
```

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

```
-------------------------
=>This Syntax generates range of values from Begin to End-1 by
maintaining Equal Interval of value with Step Value (Interval value )
=>Here varname is an object of <class,"range">
```

Examples:

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

```
>>> for v in r:
...     print(v)
                             ...
                             10
                             12
                             14
                             16
                             18
                             20
>>> for v in range(100,121,5):
...     print(v)
                             ...
                             100
                             105
                             110
                             115
                             120
>>> for v in range(100,151,10):
...     print(v) ...
                             100
                             110
                             120
                             130
                             140
                             150
>>> for v in range(100,160,10):
...     print(v) ...
                             100
                             110
                             120
                             130
                             140
                             150
========================================================================
```

Examples:

```
----------
Q 1)Generate   0   1     2     3     4    5    6    7    8    9 -----
range(10)
------------------------------------------------------------------------
>>> for  val in range(10):
...     print(val) ...
                                     0
                                     1
                                     2
                                     3
                                     4
                                     5
                                     6
                                     7
                                     8
                                     9
```

```
-----------------------------------------------------------------------
Q 2) Generate  10   11   12   13   14   15   16   17   18  19  20--
range(10,21)
-----------------------------------------------------------------------
>>> for val in range(10,21):
...     print(val) ...
10
11
12
13
14
15
16
17
18
19
20

-----------------------------------------------------------------------
Q3) Generate 100    102     104    106   108   110-----range(100,111,2)
-----------------------------------------------------------------------
>>> for val in range(100,111,2):
...     print(val)
                                    ...
                                    100
                                    102
                                    104
                                    106
                                    108
                                    110
-----------------------------------------------------------------------
Q4) Generate     -1    -2     -3   -4   -5    -6   -7  -8   -9  -10---
range(-1, -11, -1)
-----------------------------------------------------------------------
>>> for val in range(-1, -11, -1):
...     print(val) ...
                                    -1
                                    -2
                                    -3
                                    -4
                                    -5
                                    -6
                                    -7
                                    -8
                                    -9
                                    -10
-----------------------------------------------------------------------
Q5) Generate   120   115   110   105  100  ------- range(120, 99 , -5)
-----------------------------------------------------------------------
>>> for val in  range(120, 99 , -5):
...     print(val) ...
                      120
                      115
                      110
                      105
                      100
------------------------------------------------------------------------
6) 10   9   8  7   6     5    4   3   2   1-----range(10,0,-1)
------------------------------------------------------------------------
>>> for val in range(10,0,-1):
...     print(val) ...
                                     10
                                     9
                                     8
                                     7
                                     6
                                     5
                                     4
                                     3
                                     2
                                     1
------------------------------------------------------------------------
7) -10   -9   -8  -7   -6    -5------------------ range(-10, -4, 1)
------------------------------------------------------------------------
>>> for val in range(-10, -4, 1):
...     print(val) ...
                              -10
                              -9
                              -8
                              -7
                              -6
                              -5
------------------------------------------------------------------------
Q8) Generate   -10  -15   -20   -25  -30 -35---------------range(-10,-
36, -5)
>>> for val in range(-10,-36, -5):
...     print(val) ...
                                            -10
                                            -15
                                            -20
                                            -25
                                            -30
                                            -35

------------------------------------------------------------------------
Q9) generate 100   80   60   40   20  0-----range(100,-1,-20)
>>> for val in range(100,-1,-20):
...     print(val) ...
                      100
                      80
                      60
                      40
                      20
                      0

------------------------------------------------------------------------
Q 10) Generate   1000  750   500    250    0----range(1000,-1,-250)
```

```
------------------------------------------------------------------------
>>> for val in range(1000,-1,-250):
...     print(val) ...
                             1000
                             750
                             500
                             250
                             0
------------------------------------------------------------------------
Q 11) Generate   -5   -4   -3   -2  -1   0    1    2    3    4  5-----
--  range(-5, 6, 1)
------------------------------------------------------------------------
>>> for val in range(-5, 6, 1):
...     print(val) ...
                             -5
                             -4
                             -3
                             -2
                             -1
                             0
                             1
                             2
                             3
                             4
                             5
------------------------------------------------------------------------
```

MiSc. Examples:

```
----------------------------
>>> for val in r[::2][::-1]:
...     print(val) ...
                      120
                      118
                      116
                      114
                      112
                      110
                      108
                      106
                      104
                      102
                      100
>>> print(r[::2][::-1][0])---------------------120
>>> for val in r[::2][::-1][0:3]:
...     print(val) ...
                             120
                             118
                             116

================================**====================================
```

==
Type Casting techniques in Python
==

=>The Process of Converting One type of Possible value into another Type of value is Called Type Casting.
=>In Python Programming, we have 5 types of Fundamental Type Casting Techniques. They are

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

==
1) int()
== =>int() is used for Converting One Type of Possible Value into int type value.
=>Syntax:- varname=int(float / bool / complex / str)

Example1: float type----> int type----->Possible

```
>>> a=12.34
>>> print(a,type(a))-------------12.34 <class 'float'>
>>> b=int(a)
>>> print(b,type(b))-------------12 <class 'int'>
>>> a=0.0009
>>> print(a,type(a))--------------0.0009 <class 'float'>
>>> b=int(a)
>>> print(b,type(b))-------------0 <class 'int'>
```

Example2: bool type----> int type----->Possible

```
>>> a=True
>>> print(a, type(a))------------------------True <class 'bool'>
>>> b=int(a)
>>> print(b,type(b))-------------------------1 <class 'int'> >>>
a=False
>>> print(a, type(a))-----------------------False <class 'bool'>
>>> b=int(a)
>>> print(b,type(b))----------------------- 0 <class 'int'>
```

Example3: Complex type----> int type----->Not Possible

```
>>> a=2+3j
>>> print(a, type(a))------------------(2+3j) <class 'complex'> >>>
b=int(a)-----TypeError  not possible convert 'complex' value into
int type.
```

Example-4: Strings------>Int

Case-1:

```
>>> a="10" # str int------>int----> Possible
>>> print(a, type(a))-----10 <class 'str'>
>>> b=int(a)
>>> print(b,type(b))--------10 <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"
>>> 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-4: # Pure Str-----int------->Not Possible
```
-----------------------------------
>>> a="Python"
>>>  print(a,  type(a))-------------------Python  <class  'str'>  >>>
b=int(a)-------------ValueError: invalid literal for int() with base
10: 'Python'
>>> a="THREE"
>>> print(a, type(a))-----------THREE <class 'str'>
>>> b=int(a)------------ValueError: invalid literal for int() with base
10: 'THREE'
```

===

2. float()

===

=>float() is used for Converting One Type of Possible Value into float type value.

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

Example1:   int type----> float type----->Possible
```
-----------------------------------------------------------------------
>>> a=100
>>> print(a, type(a))-------------100 <class 'int'>
>>> b=float(a)
>>> print(b,type(b))--------------100.0 <class 'float'>
-----------------------------------------------------------------------
```
Example2: bool type----> float type----->Possible
```
-----------------------------------------------------------------------
>>> a=True
>>> print(a, type(a))-------------------True <class 'bool'>
>>> b=float(a)
>>> print(b,type(b))-------------------1.0 <class 'float'>
>>> a=False
>>> print(a, type(a))------------------False <class 'bool'>
>>> b=float(a)
>>> print(b,type(b))-------------------0.0 <class 'float'>
------------------------------------------------------------------------
```

Example3: complex type----> float type----->Not Possible

```
------------------------------------------------------------------------
>>> a=2+3j
>>> print(a, type(a))------------------(2+3j) <class 'complex'> >>>
b=float(a)-----------------TypeError: float() argument must be a
string or a real number, not 'complex'
------------------------------------------------------------------------
```

Example4: Str data ---> float

```
------------------------------------------------------------------------
```

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" # str float
>>> 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+3j" # Str Complex
>>> print(a, type(a))--------------2+3j <class 'str'>
>>> b=float(a)-----------ValueError: could not convert string to float:
'2+3j'
--------------------------------------------------------------------
```

Case-5: Pure Str ----->float---->Not Possible

```
-----------------
>>> a="Python"
>>> print(a, type(a))---------------Python <class 'str'> >>>
b=float(a)------------ValueError: could not convert string to
float: 'Python'
```

```
===========================================
                3. bool()
===========================================
```

=>bool() is used for Converting One Type of Possible Value into bool type value.

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

=>ALL NON-ZERO VALUES ARE CONSIDERED AS TRUE

=>ALL ZERO VALUES ARE CONSIDERED AS FALSE

Example1: int type----> bool type----->Possible

```
>>> a=123
>>> print(a, type(a))-------------------123 <class 'int'>
>>> b=bool(a)
>>> print(b,type(b))-------------------True <class 'bool'>
>>> a=-234
>>> print(a, type(a))-----------------234 <class 'int'>
>>> b=bool(a)
>>> print(b,type(b))----------------True <class 'bool'>
>>> a=0
>>> print(a, type(a))---------------0 <class 'int'>
>>> b=bool(a)
>>> print(b,type(b))----------------False <class 'bool'>
```

Example2: float type----> bool type----->Possible

```
>>> a=12.34
>>> print(a, type(a))------------12.34 <class 'float'>
>>> b=bool(a)
>>> print(b,type(b))-------------True <class 'bool'>
>>> a=0.00000000000000000000000000000000000000000001
>>> print(a, type(a))-------------1e-44 <class 'float'>
>>> b=bool(a)
>>> print(b,type(b))------------True <class 'bool'>
>>> a=0.00000000000000
>>> print(a, type(a))--------------0.0 <class 'float'> >>>
b=bool(a)
>>> print(b,type(b))--------------False <class 'bool'>
```

Example3: complex type----> bool type----->Possibility

```
>>> a=2+3j
>>> print(a, type(a))---------(2+3j) <class 'complex'>
>>> b=bool(a)
>>> print(b,type(b))-------------True <class 'bool'>
>>> a=0+0j
>>> print(a, type(a))--------------0j <class 'complex'>
>>> b=bool(a)
>>> print(b,type(b))-----------False <class 'bool'>
```

Example4: str type----> bool type

Case-1: str int-------->bool----Possible

```
>>> a="1234"
>>> print(a, type(a))--------------------1234 <class 'str'>
>>> b=bool(a)
>>> print(b,type(b))--------------------------True <class 'bool'>
>>> a="0"
>>> print(a, type(a))---------------------0 <class 'str'>
>>> b=bool(a)
>>> print(b,type(b))----------------------True <class 'bool'>
>>> len(a)------------------------------1
>>> a="     "
>>> print(a, type(a))--------------            <class 'str'>
>>> len(a)-----------------------5
>>> b=bool(a)
>>> print(b,type(b))---------------True <class 'bool'>
>>> a=""
>>> print(a, type(a))--------------     <class 'str'>
>>> len(a)-----------------------  0
>>> b=bool(a)
>>> print(b,type(b))---------------False <class 'bool'>
----------------------------------------------------------------
```

Case-1: str float-------->bool----Possible

```
--------------
>>> a="12.34"
>>> print(a, type(a))----------------12.34 <class 'str'>
>>> b=bool(a)
>>> print(b,type(b))---------------True <class 'bool'>
>>> a="0.0"
>>> print(a, type(a))------------0.0 <class 'str'>
>>> b=bool(a)
>>> print(b,type(b))-------------True <class 'bool'>
>>> len(a)---------------3
----------------------------------------------------------------
```

Case-3: str complex-------->bool----Possible

```
--------------
>>> a="2+3j"
>>> print(a, type(a))---------------2+3j <class 'str'>
>>> b=bool(a)
>>> print(b,type(b))---------------True <class 'bool'>
>>> a="0+0j"
>>> print(a, type(a))------------0+0j <class 'str'>
>>> b=bool(a)
>>> print(b,type(b))----------------True <class 'bool'>
----------------------------------------------------------------
```

Case-4: str bool-------->bool----Possible

```
--------------
>>> a="True"
>>> print(a, type(a))--------------True <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'>
----------------------------------------
```

Case-5: pure str -------->bool----Possible

```
--------------
```

```
>>> a="Python"
>>> print(a, type(a))-------------Python <class 'str'>
>>> b=bool(a)
>>> print(b,type(b))------------True <class 'bool'>
```

==

4. complex()

==

=>complex() is used for Converting One Type of Possible Value into complex type value.

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

Example1: int type----> complex type----->Possible

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

Example2: float type----> complex type----->Possible

```
>>> a=12.34
>>> print(a, type(a))------------12.34 <class 'float'>
>>> b=complex(a)
>>> print(b,type(b))---------------(12.34+0j) <class 'complex'>
```

Example3: bool type----> complex type----->Possible

```
>>> a=True
>>> print(a, type(a))-------------------True <class 'bool'>
>>> b=complex(a)
>>> print(b,type(b))-----------------(1+0j) <class 'complex'>
>>> a=False
>>> print(a, type(a))-----------------False <class 'bool'>
>>> b=complex(a)
>>> print(b,type(b))------------------0j <class 'complex'>
```

Example4: Str type----> complex type

Case-1: str int------complex type-----Possible

```
>>> a="12"
>>> print(a, type(a))-----------------12 <class 'str'>
>>> b=complex(a)
>>> print(b,type(b))-------------(12+0j) <class 'complex'>
```

Case-2: str float------complex type-----Possible

```
>>> a="1.2"
>>> print(a, type(a))---------------1.2 <class 'str'>
>>> b=complex(a)
>>> print(b,type(b))--------------(1.2+0j) <class 'complex'>
```

Case-3: str bool------complex type-----Not Possible

```
>>> a="True"
>>> print(a, type(a))----------------True <class 'str'>
>>> b=complex(a)-----------ValueError: complex() arg is a malformed
string
```

Case-4: pure str ------complex type-----Not Possible

```
>>> a="Python"
>>> print(a, type(a))---------------Python <class 'str'>
>>> b=complex(a)------------ValueError: complex() arg is a malformed
string
```

Misc Examples:

```
>>> b=complex(bool("Python"))
>>> print(b,type(b))--------------(1+0j) <class 'complex'>
>>> b=complex(bool(float(int(345))))
>>> print(b,type(b))----------------(1+0j) <class 'complex'>
----------------------------------------------------------------------
>>> b=complex(0b1010)
>>> print(b,type(b))-------------------------(10+0j) <class 'complex'>
>>> b=complex(float(0b1010))
>>> print(b,type(b))---------------------------(10+0j) <class
'complex'>
>>> b=complex(float(0b1010.0xA))----------SyntaxError: invalid decimal
literal
```
===

5. str()

===

=>str() is used for Converting All types Values into str type value.
=>Syntax:- varname=str(int / float / bool / complex)

```
>>> a=123
>>> print(a, type(a))------------------123 <class 'int'> >>>
b=str(a)
>>> print(b,type(b))-------------------123 <class 'str'>

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

>>> a=True
>>> print(a, type(a))-------------True <class 'bool'>
>>> b=str(a)
>>> print(b,type(b))---------------True <class 'str'>
>>> a=2+3.4j
>>> print(a, type(a))------------(2+3.4j) <class 'complex'>
>>> b=str(a)
>>> print(type(b)) ----------(2+3.4j) <class 'str'>
```

=================================**==================================

==
List Category Data Types (Collection Data Types or Data Structures)
==

=>The purpose of List Category Data Types is that " To store Multiple Values either of same type or different type or both the types with duplicate and Unique values".

=>List Category contains two data types. They are
1) list
2) tuple

==
1) list
==

Index:

Properties of list:

=>"list" is one of the pre-defined class and treated as list data type.
=>The purpose of list data type is that " To store Multiple Values either of same type or different type or both the types with duplicate and Unique values".
=>The elements or Values of list must be written (or) enclosed within Square Brackets.
=>An object of list maintains Insertion order.
=>On the object of list object we can perform Both Indexing and Slicing Operations.
=>An object of list belongs to Mutable.
=>We can create 2 types of list objects. They are
1. Empty List
2. Non-Empty List

1.Empty List:

=>An empty list is one which does not contain any elements and whose length is 0

```
=>Syntax:-          listobj=[ ]
                              (OR)
                          listobject=list()
```

2.Non-Empty List:

```
----------------------
=>An non-empty list is one which  contains  elements and whose length
is > 0
=>Syntax:-        listobj=[Val1,Val2,.............Val-n]

-------------------------------------------------------------------------
Examples:
-----------------
>>> l1=[10,20,30,40,-23,15]
>>> print(l1,type(l1))-----------------------------------[10, 20, 30,
40, -23, 15] <class 'list'>
>>> l2=[10,"Rossum",22.22,"Python",True]
>>> print(l2,type(l2))------------------------------------[10, 'Rossum',
22.22, 'Python', True] <class 'list'>
>>> l3=[10,10,10,"Python","Python",34.56]
>>> print(l3,type(l3))-----------------------[10, 10, 10, 'Python',
'Python', 34.56] <class 'list'>
>>> len(l1)------------6
>>> len(l2)---------5
>>> len(l3)----------6
---------------------------------
>>> l4=[]
>>> print(l4,type(l4))----------------[] <class 'list'>
>>> len(l4)---------------------------- 0
>>> l5=list()
>>> print(l5,type(l5))------------------[] <class 'list'>
>>> len(l5)------------------------------- 0
-------------------------------------------------------------------------
>>> l2=[10,"Rossum",22.22,"Python",True]
>>> print(l2,type(l2),id(l2))----[10, 'Rossum', 22.22, 'Python', True]
<class 'list'> 2974731208512 >>>
l2[0]-------------10
>>> l2[1]-----------'Rossum'
>>> l2[-1]-------------True
>>> l2[-3]---------------22.22
>>> l2[1:4]---------------['Rossum', 22.22, 'Python']
>>> l2[::]---------------[10, 'Rossum', 22.22, 'Python', True]
>>> l2[::-1]-------------[True, 'Python', 22.22, 'Rossum', 10]
>>> l2[::2]-------------[10, 22.22, True]
>>> l2[1::2]----------------['Rossum', 'Python']
>>> l2[::-2]-----------------[True, 22.22, 10]
>>> l2[::-2][0]-------------True
>>> l2[::-2][10]------------IndexError: list index out of range
-----------------------------------------------------
>>> l1=[10,20,30]
>>> ba=bytearray(l1)
>>> print(ba,type(ba))-----------------bytearray(b'\n\x14\x1e') <class
'bytearray'>
>>> l2=list(ba)
>>> print(l2,type(l2))-----------------[10, 20, 30] <class 'list'> ----
---------------------------------------------------------
>>> l2=[10,"Rossum",22.22,"Python",True]
>>> print(l2,type(l2),id(l2))--[10, 'Rossum', 22.22, 'Python', True]
<class 'list'> 2974731208512
```

```
>>> l2[0]=100
>>> print(l2,type(l2),id(l2))--[100, 'Rossum', 22.22, 'Python', True]
<class 'list'> 2974731208512
```

```
                 ========================================
                       Pre-defined Functions in list
                 ========================================
```

=>Along with the operations of indexing and slicing on list object, we can also perform
   various operations on list object by using predefined functions present in list object.

```
------------------------
1) append():
------------------------
```

=>This function is used for adding the value to the list object always at end.
=>Syntax:-   listobj.append(Value) ----------------
Examples:

```
----------------
>>> l1=[]
>>> print(l1,type(l1),id(l1))----------------------[] <class 'list'>
2974735360640
>>> len(l1)----------------------------0
>>> l1.append(10)
>>> print(l1,type(l1),id(l1))-------------------[10] <class 'list'>
2974735360640
>>> l1.append("KVR")
>>> l1.append("Python")
>>> l1.append("Java")
>>> print(l1,type(l1),id(l1))-----[10, 'KVR', 'Python', 'Java'] <class
'list'> 2974735360640
```

```
2)   insert() :
----------------------------------------------------------------------
```

=>This Function is used for inserting the specifed value in list object at perticular Index.
=>Syntax: listobj.insert(Index,Value)
=>Here Index Can be either +Ve Index ort -ve Index.
Examples:

```
-----------------
>>> l1=[10,"Rossum",34.45]
>>> print(l1,type(l1),id(l1))---------[10, 'Rossum', 34.45] <class
'list'> 2974731208512
>>> l1.insert(2,"Python")
>>> print(l1,type(l1),id(l1))----------[10, 'Rossum', 'Python', 34.45]
<class 'list'> 2974731208512
>>> l1[-1]=55.55  # Updated
>>> print(l1,type(l1),id(l1))--------[10, 'Rossum', 'Python', 55.55]
<class 'list'> 2974731208512
--------------------------------------------------------------------
3)   remove()  :---based on Value
-----------------------------------
```

=>This Function is used for removing the First Occurence of specified value from list object.

```
=>If the specfied value does not exist then we get ValueErrort.
=>Syntax:-    listobj.remove(value)

Examples:
-------------------
>>> l1=[10,"Rossum",34.45]
>>> print(l1,type(l1),id(l1))-----------[10, 'Rossum', 34.45] <class
'list'> 2974731208512
>>> l1.remove(10)
>>> print(l1,type(l1),id(l1))-------------['Rossum', 34.45] <class
'list'> 2974731208512
>>> l1.remove(34.45)
>>> print(l1,type(l1),id(l1))---------['Rossum'] <class 'list'>
2974731208512
>>> l1.remove(120)-------------ValueError: list.remove(x): x not in
list
-----------------------------------------------------------------
>>> l1=[10,"Rossum",10,"Python",10]
>>> print(l1,type(l1),id(l1))--------[10, 'Rossum', 10, 'Python', 10]
<class 'list'> 2974735360832
>>> l1.remove(10)
>>> print(l1,type(l1),id(l1))-------['Rossum', 10, 'Python', 10] <class
'list'> 2974735360832
>>> l1.remove(10)
>>> print(l1,type(l1),id(l1))--------['Rossum', 'Python', 10] <class
'list'> 2974735360832
>>> l1.remove(10)
>>> print(l1,type(l1),id(l1))----['Rossum', 'Python'] <class 'list'>
2974735360832
>>> l1.remove(10)-----------ValueError: list.remove(x): x not in list
---------------------------------------------------------------------
--------------------------------------------
4) pop(index):---based on Index
------------------------------------------------
=>This function is used for removing an element of list based on Index.
=>Syntax:-   listobj.pop(Index)
=>here Index can be either  +ve or -ve
=>If the index is invalid then we get IndexError. -------------------
Examples:
--------------------
>>> print(l1,type(l1),id(l1))------------[10, 'Rossum', 'Python', 10]
<class 'list'> 2974731208512
>>> l1.pop(-1)----------------10
>>> print(l1,type(l1),id(l1))----------[10, 'Rossum', 'Python'] <class
'list'> 2974731208512
>>> l1.pop(-2)-----------'Rossum'
>>> print(l1,type(l1),id(l1))---------[10, 'Python'] <class 'list'>
2974731208512
>>> l1.pop(11)------------IndexError: pop index out of range
========================
>>> l1=[10,20,30]
>>> print(l1)-----------------[10, 20, 30]
>>> l1.pop(0)---------------10
>>> l1.pop(0)-------------20
>>> l1.pop(1)----------IndexError: pop index out of range
>>> l1.pop(0)----------30
>>> l1.pop(1)----------IndexError: pop from empty list
======================
>>> l=list()
>>> l.pop(0)------------IndexError: pop from empty list
>>> list().pop(2)-------------IndexError: pop from empty list
>>> list().remove(0)--------ValueError: list.remove(x): x not in list
-----------------------------------------------------------------------
5)   pop():
------------------------
=>This function is used for removing last element of list but latest
inserted element. =>Syntax:-   listobj.pop()

Examples:
--------------------
>>> l1=[10,"Rossum",10,"Python",10]
>>> print(l1,type(l1),id(l1))-----------[10, 'Rossum', 10, 'Python',
10] <class 'list'> 2974735360768
>>> l1.pop()--------------10
>>> print(l1,type(l1),id(l1))---------[10, 'Rossum', 10, 'Python']
<class 'list'> 2974735360768
>>> l1.pop()--------------'Python'
>>> print(l1,type(l1),id(l1))--------[10, 'Rossum', 10] <class 'list'>
2974735360768
>>> l1.pop()----------10
>>> print(l1,type(l1),id(l1))---------[10, 'Rossum'] <class 'list'>
2974735360768
>>> l1.pop()------------'Rossum'
>>> print(l1,type(l1),id(l1))----------[10] <class 'list'>
2974735360768
>>> l1.pop()----------10
>>> print(l1,type(l1),id(l1))-------------[] <class 'list'>
2974735360768
>>> l1.pop()---------------------IndexError: pop from empty list
>>> list().pop()---------------IndexError: pop from empty list
-------------------------------------------------------
>>> l1=[10,"Rossum",10,"Python",10]
>>> l1.pop()-----------------10
>>> print(l1,type(l1),id(l1))---------[10, 'Rossum', 10, 'Python']
<class 'list'> 2974735360960
>>> l1.insert(2,"Java")
>>> print(l1,type(l1),id(l1))-----[10, 'Rossum', 'Java', 10, 'Python']
<class 'list'> 2974735360960
>>> l1.pop()-----------'Python'
>>> print(l1,type(l1),id(l1))--------[10, 'Rossum', 'Java', 10] <class
'list'> 2974735360960
>>> l1.pop()-----------10
>>> print(l1,type(l1),id(l1))-----------[10, 'Rossum', 'Java'] <class
'list'> 2974735360960
```

```
a'] <class 'list'> 2974735360960
-----------------------------------------------------------------------
Note: del operator
-----------------------------
```

```
=>'del' operator is used for deleting the element(s) of any object
either based on indexing or slicing or complete object.
Syntax:-             del   objectname[index]   # Based on Indexing

Syntax:-             del   objectname[ Begin: End ]  # Based On
Slicing

Syntax:-             del   objectname  # Complete object Removal ----
-----------------
Examples:
--------------------
>>> l1=[10,20,30,40,50,60,70,80]
>>> print(l1,type(l1),id(l1))---------[10, 20, 30, 40, 50, 60, 70, 80]
<class 'list'> 2967238884672
>>> del l1[0]
>>> print(l1,type(l1),id(l1))------------[20, 30, 40, 50, 60, 70, 80]
<class 'list'> 2967238884672
>>> del l1[-1]
>>> print(l1,type(l1),id(l1))--------------[20, 30, 40, 50, 60, 70]
<class 'list'> 2967238884672
>>> del l1[2:5]
>>> print(l1,type(l1),id(l1))------[20, 30, 70] <class 'list'>
2967238884672
--------------------------------------------------------------
>>> l1=[10,20,30,40,50,60,70,80]
>>> print(l1,type(l1),id(l1))---------[10, 20, 30, 40, 50, 60, 70, 80]
<class 'list'> 2967238870848
>>> del l1[::2]
>>> print(l1,type(l1),id(l1))-------------[20, 40, 60, 80] <class
'list'> 2967238870848
>>> del l1
>>> print(l1,type(l1),id(l1))------------NameError: name 'l1' is not
defined
----------------------------------------------------------
>>> a=10
>>> print(a,type(a))-----------------10 <class 'int'>
>>> del a
>>> print(a,type(a))--------------NameError: name 'a' is not defined
--------------------------------------------------------------------------
6) count():
------------------
=>This function is used finding number of occurences of elements in
list
=>Syntax:-    listobj.count() -------------------
Examples:
-------------------
>>> l1=[10,20,30,10,40,20,50,60,10,20]
>>> print(l1)-------------------------[10, 20, 30, 10, 40, 20, 50, 60,
10, 20]
>>> l1.count(10)-----------3
>>> l1.count(20)-----------3 >>>
l1.count(40)----------1
>>> l1.count(400)----------0
>>> l1=["apple","orange","kiwi","apple","apple"]
>>> l1.count("apple")----------------3
>>> l1.count("sberry")------------0
-----------------------------------------------------------------------
7) index()  :
----------------------
=>This Function is used for finding Index of first occurence of
specified value from list object.
=>if a specified value is not present list object then we get
ValueError
=>Syntax:    listobj.index(Value) --------------------
Examples:
--------------------
>>> l1=[10,20,30,10,40,20,50,60,10,20]
>>> print(l1)-----------------[10, 20, 30, 10, 40, 20, 50, 60, 10, 20]
>>> l1.index(10)-----------0
>>> l1=["Python","Java","Data Sci","Django"]
>>> print(l1)--------------['Python', 'Java', 'Data Sci', 'Django']
>>> l1.index("Java")-----------1
>>> l1.index("Data Sci")--------2
>>> l1.index("PHP")----------ValueError: 'PHP' is not in list
>>> l1.index("python")----------ValueError: 'python' is not in list
-----------------------------------------------------------------------
8)  copy():
----------------------------
=>This Function is used for copying the content of one object into
another object ( implements Shallow Copy)
=>Syntax:-      listobj2=listobj1.copy()

Examples:---Shallow Copy
--------------------
>>> l1=[10,"RS"]
>>> print(l1,id(l1))---------------------[10, 'RS'] 2967238870848
>>> l2=l1.copy() # Shallow Copy
>>> print(l2,id(l2))--------------------[10, 'RS'] 2967238884672
>>> l1.append("Python")
>>> l2.append("DS")
>>> print(l1,id(l1))----------------[10, 'RS', 'Python'] 2967238870848
>>> print(l2,id(l2))---------------[10, 'RS', 'DS'] 2967238884672
-----------------------------------------------------------------------
Examples:  # Slicce Based Copy----Shallow Copy
-----------------------------------------------------------------------
>>> l1=[10,20,30,40,50]
>>> print(l1,id(l1))-----------------[10, 20, 30, 40, 50] 2967238884672
>>> l2=l1[::]  # Slicce Based Copy----Shallow Copy
>>> print(l2,id(l2))--------------------[10, 20, 30, 40, 50]
2967238870848
>>> l2.remove(10)
>>> l1.pop()-------------------50
>>> print(l1,id(l1))------------------[10, 20, 30, 40] 2967238884672
>>> print(l2,id(l2))----------------[20, 30, 40, 50] 2967238870848
------------------------------------------ Examples:--Deep
Copy
------------------------------------------
>>> l1=[10,"RS"]
>>> print(l1,id(l1))------------------[10, 'RS'] 2967243016192
>>> l2=l1 # Deep Copy
>>> print(l2,id(l2))----------------[10, 'RS'] 2967243016192
>>> l1.append("Python")
>>> print(l1,id(l1))-----------------[10, 'RS', 'Python'] 2967243016192
>>> print(l2,id(l2))----------------[10, 'RS', 'Python'] 2967243016192
>>> l2.insert(2,"DS")
>>> print(l1,id(l1))-----------------[10, 'RS', 'DS', 'Python']
2967243016192
>>> print(l2,id(l2))-------------[10, 'RS', 'DS', 'Python']
2967243016192
------------------------------------------------------------------------
9) extend():
----------------------------------
=>This function is used for extending the functionality of Source list
object with Destination list object.
=>Syntax:            Sourcelistobject.extend(Destinationlistobject)

Examples:
-----------------
>>> l1=[10,20,30,40] # ----Source List
>>> l2=["Python","Data Sci","Django"] # Destination list object
>>> l1.extend(l2)
>>> print(l1)-----------[10, 20, 30, 40, 'Python', 'Data Sci',
'Django']
>>> print(l2)-----------['Python', 'Data Sci', 'Django']
-------------------------------------------------------
>>> l1=[10,20,30,40]
>>> l2=["Python","Data Sci","Django"]
>>> l3=["Oracle","MySQL","MongoDB"]
>>> l1.extend(l2,l3)---------TypeError: list.extend() takes exactly one
argument (2 given)
>>>
#__________________OR_______________________________________________
>>> l1=l1+l2+l3   # Here we are using + operator instead of extend()
>>> print(l1)---[10, 20, 30, 40, 'Python', 'Data Sci', 'Django',
'Oracle', 'MySQL', 'MongoDB']
>>> print(l2)------['Python', 'Data Sci', 'Django']
>>> print(l3)------['Oracle', 'MySQL', 'MongoDB']
------------------------------------------------------------------------
10) reverse()
------------------------------------------------------------------------
=>This Function is used for obtaining Reverse of elements of List ( front
to back and back to front)
=>Syntax:-    listobj.reverse() -------------------
Examples:
--------------------
>>> l1=[10,20,30,40]
>>> print(l1)-------------[10, 20, 30, 40]
>>> l1.reverse()
>>> print(l1)--------------[40, 30, 20, 10]

>>> l2=["Python","Data Sci","Django"]
>>> print(l2)---------------['Python', 'Data Sci', 'Django'] >>>
l2.reverse()
>>> print(l2)---------------['Django', 'Data Sci', 'Python']
```

```
----------------------------------------------------------------------
11) sort():
----------------------------------------------------------------------
=>Syntax1:-     listobj.sort()  (or) listobj.sort(reverse=False)
=>Syntax2:-     listobj.sort(reverse=True)
=>Syntax-1 makes list of Homogeneous values in Ascending Order(by
defaultreverse=False).
=>Syntax-2 makes list of Homogeneous values in Decending Order

Examples:
--------------------
>>> l1=[10,2,56,23,12,-5,0,34,12]
>>> print(l1)-------------------------[10, 2, 56, 23, 12, -5, 0, 34,
12]
>>> l1.sort()
>>> print(l1)--------------------------[-5, 0, 2, 10, 12, 12, 23, 34,
56]
>>> l1.reverse()
>>> print(l1)---------------------[56, 34, 23, 12, 12, 10, 2, 0, -5]
-------------------------------------------
>>> l1=[10,2,56,23,12,-5,0,34,12]
>>> print(l1)
[10, 2, 56, 23, 12, -5, 0, 34, 12]
>>> l1.sort(reverse=True)
>>> print(l1)-------------------------------[56, 34, 23, 12, 12, 10, 2,
0, -5]
-------------------------------------------------------------
>>> l1=[10,2,56,23,12,-5,0,34,12]
>>> print(l1)----------------------------------[10, 2, 56, 23, 12, -5,
0, 34, 12]
>>> l1.sort(reverse=False)
>>> print(l1)----------------------------------[-5, 0, 2, 10, 12, 12,
23, 34, 56]
----------------------------------------------------------------------
>>> l1=["Trump","Modiji","Putin","Sachin","Rohit","Rossum"]
>>> print(l1)-------------['Trump', 'Modiji', 'Putin', 'Sachin',
'Rohit', 'Rossum']
>>> l1.sort()
>>> print(l1)-----------------['Modiji', 'Putin', 'Rohit', 'Rossum',
'Sachin', 'Trump']
----------------------------------------------------------------------
>>> l1=["Trump","Modiji","Putin","Sachin","Rohit","Rossum"]
>>> print(l1)-------------------------------['Trump', 'Modiji',
'Putin', 'Sachin', 'Rohit', 'Rossum']
>>> l1.sort(reverse=True)
>>> print(l1)-------------['Trump', 'Sachin', 'Rossum', 'Rohit',
'Putin', 'Modiji']
----------------------------------------------------------------------
>>> l1=[10,"Rossum",34.56,2+3j,True]-----------Heterogenous values
>>> print(l1)-------------------[10, 'Rossum', 34.56, (2+3j), True] >>>
l1.sort()-----------TypeError: '<' not supported between instances of
'str' and 'int'
```

==
2) tuple
==

Properties of tuple:

=>"tuple" is one of the pre-defined class and treated as list data type.
=>The purpose of tuple data type is that " To store Multiple Values either of same type or different type or both the types with duplicate and Unique values".
=>The elements or Values of tuple must be written (or) enclosed within Braces ( ) .
=>An object of tuple maintains Insertion order.
=>On the object of tuple we can perform Both Indexing and Slicing Operations.
=>An object of tuple belongs to immutable.
=>We can create 2 types of tuple objects. They are
1. Empty tuple
2. Non-Empty tuple
---------------------- 1.Empty tuple:

=>An empty tuple is one which does not contain any elements and whose length is 0
```
=>Syntax:-          ltupleobj=()
                              (OR)
                         tupleobject=tuple()
```

---------------------- 2.Non-Empty tuple:

=>An non-empty tuple is one which contains elements and whose length is > 0
```
=>Syntax:-        tupleobj=( Val1,Val2,.............Val-n )
```

Note:- The Functionality of tuple is extactly similar to list but list object belongs to mutable and tuple belongs to immutable.
```
------------------------------------------------------------------------
>>> t1=(10,20,30,10,20)
>>> print(t1,type(t1))------------------------------(10, 20, 30, 10, 20) <class 'tuple'>
>>> t2=(10,"Rossum",44.44,"Python",3+4j,True)
>>> print(t2,type(t2))----------------(10, 'Rossum', 44.44, 'Python', (3+4j), True) <class 'tuple'>
>>> t2[0]---------------10
>>> t2[2]-----------44.44
>>> t2[-1]-----------True
>>> t2[::-1]---------(True, (3+4j), 'Python', 44.44, 'Rossum', 10)
>>> len(t1)-----------5
>>> len(t2)---------6
----------------------------------------------------------
>>> t3=()
>>> t4=tuple()
>>> print(t3,type(t3))-----------() <class 'tuple'>
>>> print(t4,type(t4))-------------() <class 'tuple'>
>>> len(t3)----------0
>>> len(t4)---------0
-----------------------------------------------------------------------
>>> t2=(10,"Rossum",44.44,"Python",3+4j,True)
>>> print(t2)--------------------(10, 'Rossum', 44.44, 'Python',
(3+4j), True)
>>> t2[0]=100-----------TypeError: 'tuple' object does not support item
assignment
-----------------------------------------
>>> l1=[10,20,34.56,67]
>>> print(l1)-------------------[10, 20, 34.56, 67]
>>> t1=tuple(l1)
>>> print(t1)-----------------(10, 20, 34.56, 67)
```

==
pre-defined function in tuple
==

=>tuple object contains 2 pre-defined functions. They are
1. count()
2. index()

Examples:

```
>>> t1=(10,10,10,20,30,40,20)
>>> t1.count(10)----------3
>>> t1.count(20)---------2
>>> t1.count(40)--------1
>>> t1.count(400)---------0
>>> t1=(10,"Ram",44.44)
>>> t1.index("Ram")----------1
>>> t1.index(10)-------------0
>>> t1.index(100)----------ValueError: tuple.index(x): x not in tuple
```

NOTE: tuple object does not contain the following Function, bcoz tuple is immutable

append(val) insert(index, val) remove(val), pop (index), pop() copy()

==
Inner or nested List
==

=>The Process of defining one list inside of another list is called Inner or Nested List.
=>Syntax: listobj= [val1, val2.....[val11,val12,...val1n] ,......val-n]

=>Here Val1,Val2...Val-n are present in Outer List.
=>Here val11,val12,...val1n are present in inner List.
=>On inner list we can perform both Indexing and slicing Operations.
=>On inner list we can apply all types of Pre-defined functions of list.

```
----------------------
Examples:
----------------------
>>> lst=[10,"Ram", [15,17,14], [70,67,72], "OUCET" ]
>>> print(lst)------------[10, 'Ram', [15, 17, 14], [70, 67, 72],
'OUCET']
>>> lst[0]-------------10
>>> lst[1]-----------'Ram'
>>> lst[2]----------[15, 17, 14]
>>> lst[3]-------------[70, 67, 72]
>>> lst[4]----------'OUCET'
>>> lst[-3]-----------[15, 17, 14]
>>> lst[-2]----------[70, 67, 72]
>>> lst[2][0]----------15
>>> lst[2][-3]------------15
>>> lst[-3][-1]---------14
-----------------------------------------------------------
>>> print(lst)---------[10, 'Ram', [15, 17, 14], [70, 67, 72], 'OUCET']
>>> lst[3][1:]-----------[67, 72]
>>> lst[3][::-1]-------------[72, 67, 70]
----------------------------------------------------------------------
--
>>> print(lst)----------------[10, 'Ram', [15, 17, 14], [70, 67, 72],
'OUCET']
>>> lst[2].append(16)
>>> print(lst)---------[10, 'Ram', [15, 17, 14, 16], [70, 67, 72],
'OUCET']
>>> lst[-2].insert(1,60)
>>> print(lst)------------[10, 'Ram', [15, 17, 14, 16], [70, 60, 67,
72], 'OUCET']
>>> lst[2].sort()
>>> print(lst)---------[10, 'Ram', [14, 15, 16, 17], [70, 60, 67, 72],
'OUCET']
>>> lst[-2].sort(reverse=True)
>>> print(lst)---------[10, 'Ram', [14, 15, 16, 17], [72, 70, 67, 60],
'OUCET']
>>> lst[2].pop()----------17
>>> lst[3].remove(67)
>>> print(lst)-------------[10, 'Ram', [14, 15, 16], [72, 70, 60],
'OUCET']
>>> lst.pop(2)------------[14, 15, 16]
>>> print(lst)-----------[10, 'Ram', [72, 70, 60], 'OUCET']
>>> del lst[2]
>>> print(lst)---------[10, 'Ram', 'OUCET']
```

======================================&&===================================
==
Set Category Data Types (Collection Data Types or Data Structures)
==
=>=>The purpose of Set Category Data Types is that " To store Multiple Values either of same type or different type or both the types with Unique values".
=>In Python programming, we have two data types in Set Category. They are
1. set (Both Immutable and Mutable)

```
2. frozenset (immutable)

                ================================================
                                      set
                ================================================
=>Properties of set:
---------------------------------
=>'set' is one of the pre-defined class and treated as Set Data Types.
=>The purpose of set data type is that To store Multiple Values either
of same type or different type or both the types with  Unique values".
=>The elements of set must organized within curly braces { } and whose
elements separated semi colon.
=>An object of set never maintains insertion order because PVM display
any possibility of elements of set.
=>Since object of set never maintains insertion order and hence we
can't perform Indexing and slicing Operations.
=>An object of set belongs to Both immutable because set' object does
not support item assignment and mutable because we can add elements to
set object by using add().
=>We have two types of set objects. They are
a) empty set
b) non-empty set
------------------ a)
empty set -----------
-------
=>An empty set is one, which contains elements and whose length is
equal to 0
=>Syntax:-            setobj=set()      #    setobj={}---is invalid

---------------------------
b) non-empty set
---------------------------
=>An non-empty set is one, which contains elements and whose length is
>0
=>Syntax:-            setobj={v1,v2....vn}
-----------------------------------------------------------------------
Examples:
--------------------
>>> s1={10,20,30,40,50,60,10,10}
>>> print(s1,type(s1))-------------------{50, 20, 40, 10, 60, 30}
<class 'set'>
>>> len(s1)---------------6
>>> s2={"Python","Java","Django","Data Sci","Python"}
>>> print(s2,type(s2))------------------{'Python', 'Data Sci', 'Java',
'Django'} <class 'set'>
>>> s3={10,"Rossum",34.56,True,2+3j}
>>> print(s3,type(s3))--------------{True, 34.56, 'Rossum', 10, (2+3j)}
<class 'set'>
>>> len(s2)-----------------4
>>> len(s3)-----------------5
--------------------------------------------------
>>> s1={}
>>> print(s1,type(s1))-----------{} <class 'dict'>
>>> s1=set()
>>> print(s1,type(s1))-----------set() <class 'set'>
>>> len(s1)--------------------0
--------------------------------
>>> s3={10,"Rossum",34.56,True,2+3j}
>>> s3[0]--------------------TypeError: 'set' object is not
subscriptable
>>> s3[0:4]--------------TypeError: 'set' object is not subscriptable
```

==

Pre-defined Functions in set

==

=>On the object of set, we can perform various operations by using the pre-defined functions of set.
=>The following are the function of set.

---

1) add()

=>This Function is used for adding the elements to set object
=>Syntax:-   setobj1.add(Value)
=>Examples:

```
>>> s1=set()
>>> print(s1,type(s1),id(s1))-----------------set() <class 'set'>
2175600857536
>>> len(s1)-------------0
>>> s1.add(10)
>>> s1.add("Python")
>>> s1.add(12.34)
>>> print(s1,type(s1),id(s1))-------------{'Python', 10, 12.34} <class
'set'> 2175600857536
>>> len(s1)------------3
------------------------------------------------------
```

2) clear():

=>This function is used removing all the elements of set.
=>Syntax: setobj.clear() -------------------
Examples:

```
>>> s3={10,"Rossum",34.56,True,2+3j}
>>> print(s3,id(s3))------------------{True, 34.56, 'Rossum', 10,
(2+3j)} 2175600858432
>>> len(s3)---------------5
>>> s3.clear()
>>> print(s3,id(s3))-----------set() 2175600858432
>>> len(s3)----------------0
----------------------------------------------------------------------
```

3) remove() :

=>This function is used for removing an element from set object. =>If element does not exist in set object then we get KeyError(bcoz elements of set are unique and unique elements are called Keys)
=>Syntax: setobj.remove(Element)

```
------------------
Examples:
------------------
>>> s3={10,"Rossum",34.56,True,2+3j}
>>> print(s3,id(s3))-------------{True, 34.56, 'Rossum', 10, (2+3j)}
2175600856640
>>> s3.remove(34.56)
>>> print(s3,id(s3))--------------{True, 'Rossum', 10, (2+3j)}
2175600856640
>>> s3.remove(True)
>>> print(s3,id(s3))-----------{'Rossum', 10, (2+3j)} 2175600856640
>>> s3.remove("Python")-----------KeyError: 'Python'
-------------------------------------------------------------------------
4) discard():
-------------------------------------------------------------------------
=>This Function is used  removing the element from set object
=>If element does not exist in set object then we never get KeyError
=>Syntax:-     setobj.discard(element)


Examples:
-------------------
>>> s3={10,"Rossum",34.56,True,2+3j}
>>> print(s3,id(s3))--------------{True, 34.56, 'Rossum', 10, (2+3j)}
2175600858432
>>> s3.discard("Rossum")
>>> print(s3,id(s3))---------------{True, 34.56, 10, (2+3j)}
2175600858432
>>> s3.discard(2+3j)
>>> print(s3,id(s3))--------------{True, 34.56, 10} 2175600858432
>>> s3.discard("Rossum")-----No Error
>>> s3.discard("Java")-----No Error
>>> s3.remove("Java")-------KeyError: 'Java'
-------------------------------------------------------------------------
5) pop():
-------------------------------------------------------------------------
=>This Function is used for removing any arbitrary Element from set
object.
=>if we use pop() on empty set then we get KeyError
=>Syntax:     setobj.pop() ---------------
Examples:
---------------
>>> s3={10,"Rossum",34.56,True,2+3j}
>>> s3.pop()-------------------True
>>> s3.pop()---------------34.56
>>> s3.pop()-------------'Rossum'
>>> s3.pop()------------10
>>> s3.pop()-------------(2+3j)
>>> print(s3,id(s3))---------set() 2175600857536
>>> s3.pop()----------KeyError: 'pop from an empty set'
>>> set().pop()-----------KeyError: 'pop from an empty set'
>>> s1={10,20,30,40,50,60,10}
>>> s1.pop()-----------50 >>>
s1.pop()---------20
>>> s1.pop()-----------40
>>> s1.pop()------------10 >>>
s1.pop()----------60
>>> s1.pop()------------30
>>> s1.pop()--------KeyError: 'pop from an empty set'
-------------------------------------------------------------------------
Note:
>>> {10,20,30,40,50,60,10}.pop()----------50
>>>s1=set()
>>>s1.pop()-----------KeyError: 'pop from an empty set'
>>>set().pop()-------KeyError: 'pop from an empty set'
-------------------------------------------------------------------------

6)  copy():
---------------------------------
=>This Function is used for copying the content of one set object into
another set object ( shallow copy) =>Syntax:
setobj2=setobj1.copy()
Examples:
----------------
>>> s1={"Apple","Mango","kiwi","Sberry","Guava","Orange"}
>>> s2=s1.copy()  # shallow Copy
>>> print(s1,id(s1))-----------{'Mango', 'kiwi', 'Sberry', 'Apple',
'Guava', 'Orange'} 2175600858432
>>> print(s2,id(s2))----------{'Mango', 'kiwi', 'Sberry', 'Apple',
'Guava', 'Orange'} 2175600857984
>>> s1.add("Java")
>>> s2.add("python")
>>> print(s1,id(s1))---{'Mango', 'kiwi', 'Java', 'Sberry', 'Apple',
'Guava', 'Orange'}
>>> print(s2,id(s2))--------{'Mango', 'kiwi', 'Sberry', 'Apple',
'python', 'Guava', 'Orange'}
Note:
------------
>>> s1=set()
>>> s2=s1.copy()
>>> print(s1,id(s1))----------set() 2175600856192
>>> print(s2,id(s2))---------set() 2175600856640
-------------------------------------------------------------------------
7) issuperset():---
-------------------------
Syntax:   setobj1.issuperset(setobj2)
=>This function Returns True provided all elements of setobj2 present
in setobj1. Otherwise it return False.
Examples:
-----------------
>>> S1={10,20,30,40,50}
>>> S2={10,20}
>>> S3={10,60,70}
>>> S1.issuperset(S2)---------True
>>> S1.issuperset(S3)-----------False Note:
----------
>>> S1.issuperset(set())----------------True
>>> set().issuperset(set())-------------True
```

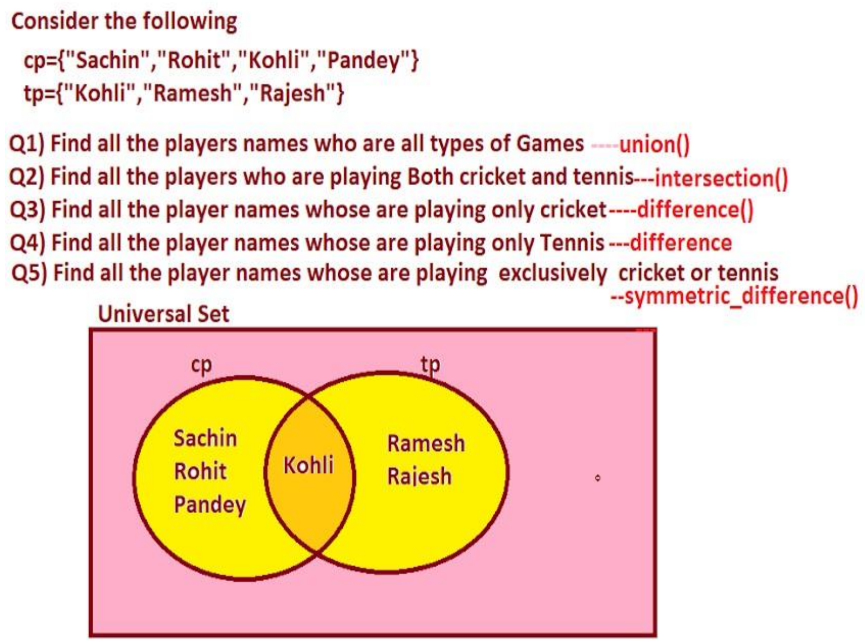
```
-----------------------------------------------------------------------
8) issubset():---
-------------------------
Syntax:   setobj1.issubset(setobj2)

Examples:
---------------
>>> s1={10,20,30,40,50}
>>> s2={40,50}
>>> s3={10,40,70}
>>> s2.issubset(s1)----------------True >>>
s3.issubset(s1)-------------False
Note:
---------
>>> set().issubset(s1)--------True
>>> set().issubset(set())-------True
-----------------------------------------------------------------------
9) isdisjoint():
-------------------------
Syntax:       setobj1.isdisjoint(setobj2)
=>This Function Returns True provided setobj1 and setobj2 does not
contains any common element(s). Otherwise it return False.

Examples:
----------------
>>> s1={10,20,30,40,50}
>>> s2={40,60,70}
>>> s3={"Python","Django"}
>>> s1.isdisjoint(s3)---------------True
>>> s1.isdisjoint(s2)-----------False
----------------------------------------------
>>> s1={"Sachin","Rohit","Kohli"}
>>> s2={"Rossum","Ritche","Travis"}
>>> s3={"Sachin","MC"}
>>> s1.isdisjoint(s2)-------------True
>>> s1.isdisjoint(s3)-----------False
----------------------------------------------------
>>> s1={"Sachin","Rohit","Kohli"}
>>> s1.isdisjoint(set())-----------True
>>> set().isdisjoint(set())-----------True
-----------------------------------------------------------------------
 10) union():
 -------------------------------------
=>Syntax:-      setobj3=setobj1.union(setobj2)
=>This Function is used for obtaining all the elements from both
setobj1 and setobj2 uniquely.

Examples:
-----------------
>>> cp={"Sachin","Rohit","Kohli","Pandey"}
>>> tp={"Kohli","Ramesh","Rajesh"}
>>> cptp=cp.union(tp)
>>> print(cptp)---------------{'Rajesh', 'Pandey', 'Ramesh', 'Kohli',
'Rohit', 'Sachin'}
               (OR)
```

```
>>> cptp=tp.union(cp)
>>> print(cptp)---------------{'Rajesh', 'Pandey', 'Rohit', 'Kohli',
'Ramesh', 'Sachin'}
```



Consider the following

cp={"Sachin","Rohit","Kohli","Pandey"}
tp={"Kohli","Ramesh","Rajesh"}

Q1) Find all the players names who are all types of Games ----union()
Q2) Find all the players who are playing Both cricket and tennis---intersection()
Q3) Find all the player names whose are playing only cricket----difference()
Q4) Find all the player names whose are playing only Tennis ---difference
Q5) Find all the player names whose are playing exclusively cricket or tennis --symmetric_difference()

```
-----------------------------------------------------------------------
11)   intersection() :
 -------------------------------------
=>Syntax:-      setobj3= setobj1. intersection (setobj2) =>This
Function is used for obtaining common  elements from both setobj1
and setobj2
Examples:
--------------------
>>> cp={"Sachin","Rohit","Kohli","Pandey"}
>>> tp={"Kohli","Ramesh","Rajesh"}
>>> bothcptp=cp.intersection(tp)
>>> print(bothcptp)---------------{'Kohli'}
>>> bothcptp=tp.intersection(cp)
>>> print(bothcptp)------------------{'Kohli'}
-----------------------------------------------------------------------
12) difference():
------------------------------------
=>Syntax:     setobj3=setobj1.difference(setobj2)
=>This Function removes the common elements from setobj1 and setobj2
and takes and display remain elements of setobj1.

Examples:
------------------
>>> cp={"Sachin","Rohit","Kohli","Pandey"}
>>> tp={"Kohli","Ramesh","Rajesh"}
>>> onlycp=cp.difference(tp)
>>> print(onlycp)-----------{'Pandey', 'Rohit', 'Sachin'}
>>> onlytp=tp.difference(cp)
>>> print(onlytp)----------------{'Rajesh', 'Ramesh'}
```

```
13) symmetric_difference()
-----------------------------------------------------------------------
=>Syntax:    setobj3=setobj1.symmetric_difference(setobj2)
=>=>This Function removes the common elements from setobj1 and setobj2
and takes and display remain elements from both setobj1 and setobj2 and
place them in setobj3.
Examples:
--------------------
>>> cp={"Sachin","Rohit","Kohli","Pandey"}
>>> tp={"Kohli","Ramesh","Rajesh"}
>>> exclcptp=cp.symmetric_difference(tp)
>>> print(exclcptp)--------------------{'Rohit', 'Sachin', 'Rajesh',
'Pandey', 'Ramesh'}
------------------------OR-----------------------------------------------
--------
>>> exclcptp=cp.union(tp).difference(cp.intersection(tp))
>>> print(exclcptp)-------------{'Rohit', 'Sachin', 'Rajesh', 'Pandey',
'Ramesh'}
-----------------------------------------------------------------------
14)update():
---------------------------
=>Syntax:-    setobj1.update(setobj2)
=>This function updates or adding all the unique values of setobj2 to
setobj1.

Examples:
-------------------------
>>> s1={10,"Rajesh"}
>>> s2={"Python","Data Science"}
>>> s1.update(s2)
>>> print(s1)------------------{10, 'Rajesh', 'Data Science', 'Python'}
>>> print(s2)----------------{'Data Science', 'Python'}
>>> s3={"Oracle","MySQL","MongoDB"}
>>> s1.update(s3)
>>> print(s1)-------------------{'Oracle', 'MongoDB', 'Rajesh',
'Python', 10, 'Data Science', 'MySQL'}
```

```
>>> s1={10,"Rajesh"}

>>> s2={10,20}
>>> s1.update(s2)
>>> print(s1)------------------{10, 'Rajesh', 20}
```

==
2. frozenset
==

=>Properties of frozenset:

=>'frozenset' is one of the pre-defined class and treated as Set Data Type.
=>The purpose of frozenset data type is that To store Multiple Values either of same type or different type or both the types with Unique values".
=>The elements of frozenset always represented by converting other type of elements (set, tuple,list..etc) by using frozenset() bcoz frozenset set does not cointain any symbolic notation.

=>Syntax:-   frozensetobj=frozenset(object)

=>An object of frozenset never maintains insertion order bcoz PVM display any possibility of elements of frozenset.
=>Since object of frozenset never maintains insertion order and hence we can't perform Indexing and slicing Operations.
=>An object of frozenset belongs to immutable bcoz frozenset' object does not support item assignment and never allows to add the elements externally.
=>We have two types of frozenset objects. They are
a) empty frozenset
b) non-empty frozenset
------------------ a)
empty frozenset -----

=>An empty frozenset is one, which contains elements and whose length is equal to 0
=>Syntax:-   frozensetobj=frozenset()

----
b) non-empty frozenset

=>An non-empty frozenset is one, which contains elements and whose length is >0
=>Syntax:-   frozensetobj=frozenset({v1,v2....vn})
        frozensetobj=frozenset((v1,v2....vn))
frozensetobj=frozenset([v1,v2....vn])
Note:

=>The Functionality of frozenset is similar to set and set object belongs to both mutable and immutable where as frozenset belongs to only immutable.
==
Examples:

```
>>> s1={10,20,30,40,50}
>>> print(s1,type(s1))---------------{50, 20, 40, 10, 30} <class 'set'>
>>> fs=frozenset(s1)
>>> print(fs,type(fs))-----------------frozenset({50, 20, 40, 10, 30})
<class 'frozenset'>
>>> t1=(10,"Rossum","Python",34.56)
>>> print(t1,type(t1))----------------(10, 'Rossum', 'Python', 34.56)
<class 'tuple'>
>>> fs1=frozenset(t1)
>>> print(fs1,type(fs1))--------------frozenset({10, 'Rossum', 34.56,
'Python'}) <class 'frozenset'>
>>> l1=[10,10,20,20]
>>> print(l1,type(l1))------------------[10, 10, 20, 20] <class 'list'>
>>> fs2=frozenset(l1)
>>> print(fs2,type(fs2))-----------------frozenset({10, 20}) <class
'frozenset'>
>>> len(fs)--------------5
>>> len(fs1)------------4
>>> len(fs2)-------------2
>>> fs3=frozenset()
>>> print(fs3,type(fs3))--------------frozenset() <class 'frozenset'>
>>> len(fs3)----------------0
>>> fs[0]----------------TypeError: 'frozenset' object is not
subscriptable
>>> fs[0:3]----------------TypeError: 'frozenset' object is not
subscriptable
>>> fs[0]=100------------TypeError: 'frozenset' object does not support
item assignment
>>> fs1.add(100)----------AttributeError: 'frozenset' object has no
attribute 'add'
```

===

pre-defined function in frozenset

===

=>Frozenset contains the following Functions.

1) copy
2) isuperset() 3) issubset()
4) isdisjoint()
5) union()
6) intersection()
7) difference()
8) symmetric_diffence()

Examples:

```
>>> fs2=fs1.copy()
>>> print(fs1,id(fs1))-------------   frozenset({40, 10, 20, 30})
2547261088128
>>> print(fs2,id(fs2))---------------frozenset({40, 10, 20, 30})
2547261088128
------------------------------------------
>>> fs1=frozenset({10,20,30,40})
>>> fs2=frozenset((100,20,300,400))
>>> fs1----------------frozenset({40, 10, 20, 30})
>>> fs2----------------frozenset({400, 100, 20, 300})
>>> fs1.issuperset(fs2)-------------False
>>> fs1.issubset(fs2)----------------False
>>> fs1.issubset({10,20})------------False
>>> fs1.issubset(frozenset({10,20,30,40,50,60}))-----------True
```

```
-----------------------------------------------------------------------
>>> fs1-----------------------frozenset({40, 10, 20, 30})
>>> fs2-----------------------frozenset({400, 100, 20, 300})
>>> fs3=fs1.union(fs2)
>>> print(fs3)--------------------frozenset({100, 40, 10, 300, 400, 20,
30})
>>> fs4=fs1.intersection(fs2)
>>> print(fs4)---------------frozenset({20})
>>> fs5=fs1.difference(fs2)
>>> print(fs5)---------------------frozenset({40, 10, 30})
>>> fs6=fs2.difference(fs1)
>>> print(fs6)---------------------frozenset({400, 100, 300})
>>> fs7=fs2.symmetric_difference(fs1)
>>> print(fs7)------------------frozenset({100, 40, 10, 300, 400, 30})
------------------------------------------------------------------------
```

Note: Frozenset does not contain the following Functions.
add() clear() remove() pop() discard() update()

============================**==

===

Dict Categeory Data Types (Collection Data Types or Data Structures)

===

=>'dict' is one of the pre-defined class and treated as Dict Data Type.
=>The purpose of dict data type is that "To store (Key,value) in single variable"
=>In (Key,Value), the value of Key is Unique and Value of Value may or may not be unique.
=>The (Key,value) must be organized or stored in the object of dict within Curly Braces {} and they separated by comma.
=>An object of dict does not support Indexing and Slicing bcoz Values of Key itself considered as Indices.
=>In the object of dict, Values of Key are treated as Immutable and Values of Value are treated as mutable.
=>Originally an object of dict is mutable bcoz we can add (Key,Value) extrenally.
=>We have two types of dict objects. They are
a) Empty dict
b) Non-empty dict

----
a) Empty dict

=>Empty dict is one, which does not contain any (Key,Value) and whose length is 0

```
=>Syntax:-          dictobj1= { }
                              or
                           dictobj=dict()
```

=>Syntax for adding (Key,Value) to empty dict:

```
    ----------------------------------------------------------------
            dictobj[Key1]=Val1
dictobj[Key2]=Val2
            ---------------------------
dictobj[Key-n]=Val-n
```

```
Here Key1,Key2...Key-n are called Values of Key and They must Unique
Here Val1, Val2...Val-n are called Values of Value and They may or may
not be unique.
----------------------------------------------------------------------
------------------------------------------------
b) Non-Empty dict
-------------------------
=>Non-Empty dict is one, which contains  (Key,Value) and whose length
is >0
=>Syntax:-          dictobj1= { Key1:Val1,Key2:Val2......Key-n:Val4}

Here Key1,Key2...Key-n are called Values of Key and They must Unique
Here Val1, Val2...Val-n are called Values of Value and They may or may
not be unique.
======================================================================
Examples:
-----------------
>>> d1={10:"Python",20:"Data Sci",30:"Django"}
>>> print(d1,type(d1))----------{10: 'Python', 20: 'Data Sci', 30:
'Django'} <class 'dict'>
>>> d2={10:3.4,20:4.5,30:5.6,40:3.4}
>>> print(d2,type(d2))------------{10: 3.4, 20: 4.5, 30: 5.6, 40: 3.4}
<class 'dict'>

>>> d4[10]="Guava"
>>> print(d4,type(d4),id(d4))----{10: 'Guava', 20: 'Mango', 30: 'Kiwi',
40: 'Sberry', 50: 'Orange'} <class 'dict'> 2090754532032
----------------------------------------------------------------------
```

```
>>> len(d1)--------------3
>>> len(d2)------------4
-------------------------------------------------
>>> d3={}
>>> print(d3,type(d3))------------{} <class 'dict'>
>>> len(d3)-------------0
>>> d4=dict()
>>> print(d4,type(d4))-------------{} <class 'dict'>
>>> len(d4)---------------0
-----------------------------------------------------------------------
>>> d2={10:3.4,20:4.5,30:5.6,40:3.4}
>>> print(d2)-----------------------------------{10: 3.4, 20: 4.5, 30:
5.6, 40: 3.4}
>>> d2[0]---------------------------------------KeyError: 0
>>> d2[10]------------------------------------3.4
>>> d2[10]=10.44
>>> print(d2)--------------------------------{10: 10.44, 20: 4.5, 30:
5.6, 40: 3.4}
-----------------------------------------------------------------------
>>> d2={10:3.4,20:4.5,30:5.6,40:3.4}
>>> print(d2,type(d2),id(d2))----{10: 3.4, 20: 4.5, 30: 5.6, 40: 3.4}
<class 'dict'> 2090750380736
>>> d2[50]=5.5
>>> print(d2,type(d2),id(d2))---{10: 3.4, 20: 4.5, 30: 5.6, 40: 3.4,
50: 5.5} <class 'dict'>
-----------------------------------------------------------------------
>>> d3={}
>>> print(d3,type(d3),id(d3))-------------------{} <class 'dict'>
2090750332992
>>> d3["Python"]=1
>>> d3["Java"]=3
>>> d3["C"]=2
>>> d3["GO"]=1
>>> print(d3,type(d3),id(d3))-----{'Python': 1, 'Java': 3, 'C': 2,
'GO': 1} <class 'dict'>
------------------------------------------------------------- >>>
d4=dict()
>>> print(d4,type(d4),id(d4))---------------{} <class 'dict'>
2090754532032
>>> d4[10]="Apple"
>>> d4[20]="Mango"
>>> d4[30]="Kiwi"
>>> d4[40]="Sberry"
>>> d4[50]="Orange"
>>> print(d4,type(d4),id(d4))---{10: 'Apple', 20: 'Mango', 30: 'Kiwi',
40: 'Sberry', 50: 'Orange'}                                          <cl
>>> d2={10:3.4,20:4.5,30:5.6,40:3.4}
>>> print(d2,type(d2),id(d2))---{10: 3.4, 20: 4.5, 30: 5.6, 40: 3.4}
<class 'dict'> 2090754531520
>>> d2[50]=1.2
>>> print(d2,type(d2),id(d2))---{10: 3.4, 20: 4.5, 30: 5.6, 40: 3.4,
50: 1.2} <class 'dict'> 2090754531520
```

==

Pre-defined Functions in dict

==

=>On the object of dict, we can perform some additional operations by using the pre-defined functions present in dict object.

----

1) clear()

=>This function is used for removing all the elements from dict object.

=>**Syntax:-** dictobj.clear()

Examples:

```
>>> d1={10:3.4,20:4.5,30:5.6,40:3.4}
>>> print(d1,len(d1))-------------------{10: 3.4, 20: 4.5, 30: 5.6, 40:
3.4}    4
>>> d1.clear()
>>> print(d1,len(d1))-----------------{}  0
>>> print({}.clear())----------------None
>>> print(dict().clear())---------------None
```

----

2) copy():

=>This function is used for copying the content of one dict object into another dict object.

=>**Syntax:** dictobj2=dictobj1.copy()

Examples:

```
>>> d1={10:3.4,20:4.5,30:5.6,40:3.4}
>>> print(d1,id(d1))----------{10: 3.4, 20: 4.5, 30: 5.6, 40: 3.4}
2090750382720
>>> d2=d1.copy()  # shallow Copy
>>> print(d2,id(d2))-----{10: 3.4, 20: 4.5, 30: 5.6, 40: 3.4}
2090750332864
>>> d1[50]=1.2
>>> d2[45]=11.2
>>> print(d1,id(d1))-----{10: 3.4, 20: 4.5, 30: 5.6, 40: 3.4, 50: 1.2}
2090750382720
>>> print(d2,id(d2))----{10: 3.4, 20: 4.5, 30: 5.6, 40: 3.4, 45: 11.2}
2090750332864
-------------------------------
>>> d1={}.copy()
>>> print(d1,len(d1))----------{}   0
>>> d2=dict().copy()
>>> print(d2,len(d2))-------------{}   0
```

----

3) pop():

=>This Function is used for removing (Key,value) from dict object provided Value of Key must exist in dict object otherwise we get KeyError.

=>Syntax: dictobj.pop(Key)

Examples:

```
-----------------
>>> d1={10:3.4,20:4.5,30:5.6,40:3.4}
>>> d1.pop(20)-------------4.5
>>> print(d1)------------------{10: 3.4, 30: 5.6, 40: 3.4}
>>> d1.pop(30)------------5.6
>>> print(d1)--------------{10: 3.4, 40: 3.4}
>>> d1.pop(40)-----------3.4
>>> print(d1)-------------{10: 3.4}
>>> d1.pop(60)--------KeyError: 60
-----------------------------------------------------------------------
```

4) popitem():

```
-----------------------------------------------------------------------
=>This function is used for removing last (key,value) from non-dict
object otherwise we get
    KeyError
=>Syntax:-    dictobj.popitem()
```

Examples:

```
-----------------
>>> d1={10: 'Guava', 20: 'Mango', 30: 'Kiwi', 40: 'Sberry', 50:
'Orange'}
>>> print(d1)
{10: 'Guava', 20: 'Mango', 30: 'Kiwi', 40: 'Sberry', 50: 'Orange'}
>>> d1.popitem()------------(50, 'Orange')
>>> print(d1)------------{10: 'Guava', 20: 'Mango', 30: 'Kiwi', 40:
'Sberry'}
>>> d1.popitem()----------(40, 'Sberry')
>>> print(d1)-----------{10: 'Guava', 20: 'Mango', 30: 'Kiwi'}
>>> d1.popitem()-----------(30, 'Kiwi')
>>> print(d1)---------------{10: 'Guava', 20: 'Mango'}
>>> d1.popitem()------------(20, 'Mango')
>>> print(d1)--------------{10: 'Guava'}
>>> d1.popitem()-----------(10, 'Guava')
>>> print(d1)--------------{}
>>> d1.popitem()------------KeyError: 'popitem(): dictionary is empty'
>>> {}.popitem()-----------KeyError: 'popitem(): dictionary is empty'
>>> dict().popitem()---------KeyError: 'popitem(): dictionary is empty'
---------------------------------------
```

Special Note:

```
----------------------------------------
>>> d1=dict( [(10,1.2),(20,2.3),(30,4.5)] ) # Converting List of tuples
into dict object
>>> d1-----------------{10: 1.2, 20: 2.3, 30: 4.5}
-----------------------------------------------------------------------
>>> dict( [(10,1.2),(20,2.3),(30,4.5)] ).pop(60)---------KeyError: 60
>>> dict( [(10,1.2),(20,2.3),(30,4.5)] ).popitem()---------(30, 4.5)
>>> d1-----------------------------------{10: 1.2, 20: 2.3, 30: 4.5}
>>> d1.popitem()---------------(30, 4.5)
>>> d1---------------------{10: 1.2, 20: 2.3}
>>> d1.popitem()-------------(20, 2.3)
>>> d1---------------{10: 1.2}
>>> d1.popitem()---------------(10, 1.2)
>>> d1-----------------------------{}
>>> d1.popitem()-----------------KeyError: 'popitem(): dictionary is
empty'
```
----

5) get()

=>This Function is used for obtaining Value of Value by passing Value of Key from Non-Empty dict object otherwise we get None

=>Syntax: dictobj.get(Key)

Examples

```
>>> d1={10: 'Guava', 20: 'Mango', 30: 'Kiwi', 40: 'Sberry', 50:
'Orange'}
>>> print(d1)
{10: 'Guava', 20: 'Mango', 30: 'Kiwi', 40: 'Sberry', 50: 'Orange'}
>>> d1[20]----------------'Mango'
>>> d1[10]-------------'Guava'
>>> d1[30]----------------'Kiwi'
============================OR=========================
>>> d1.get(20)------------------'Mango'
>>> d1.get(10)-----------------'Guava'
>>> d1.get(30)-----------------'Kiwi'
>>> d1.get(50)-------------------'Orange'
>>> print(d1.get(500))----------None
>>> print(dict().get(10))----------None
>>> print({}.get(10))-----------None
>>> dict([[10,1.2],[20,3.4]]).get(20)----------3.4
>>> dict([[10,1.2],[20,3.4]]).get(10)--------1.2
>>> print(dict([[10,1.2],[20,3.4]]).get(30))-------None
```
----

6) keys()

=>This Functrion is used for obtaining Value of Keys from non-empty dict object.

```
=>Syntax:         varname=dictobj.keys()
                             (OR)
                        dictobj.keys()
```

Examples:

```
>>> d1={10: 'Guava', 20: 'Mango', 30: 'Kiwi', 40: 'Sberry', 50:
'Orange'}
>>> print(d1)---------{10: 'Guava', 20: 'Mango', 30: 'Kiwi', 40:
'Sberry', 50: 'Orange'}
>>> k=d1.keys()
>>> print(k)----------dict_keys([10, 20, 30, 40, 50])
>>> for kv in k:
...     print(kv)
                             ...
                             10
                             20
                             30
                             40
                             50
```

```
======================OR====================
>>> for kv in  d1.keys():
...     print(kv)
                              ...
                              10
                              20
                              30
                              40
                              50

>>> d2={}.keys()
>>> d2------------dict_keys([])
>>> d3=dict().keys()
>>> d3----------------dict_keys([])
```

---

7) values():

```
------------------
=>This Function is used for obtaining Values of Value from non-empty dict
object.
=>Syntax:         varname=dictobj.values()
                                (OR)
                           dictobj.values()
```

Examples:

```
--------------------
>>> d1={10: 'Guava', 20: 'Mango', 30: 'Kiwi', 40: 'Sberry', 50: 'Orange'}
>>> print(d1)-----------------{10: 'Guava', 20: 'Mango', 30: 'Kiwi', 40:
'Sberry', 50: 'Orange'}
>>> v=d1.values()
>>> print(v,type(v))--dict_values(['Guava', 'Mango', 'Kiwi', 'Sberry',
'Orange']) <class
>>> for vv in v:
...     print(vv)
                              ...
                              Guava
                              Mango
                              Kiwi
                              Sberry
                              Orange
>>> for v in d1.values():
...     print(v)
                              ...
                              Guava
                              Mango
                              Kiwi
                              Sberry
                              Orange
Note:
-----------
>>> d1={10: 'Guava', 20: 'Mango', 30: 'Kiwi', 40: 'Sberry', 50: 'Orange'}
>>> print(d1)
{10: 'Guava', 20: 'Mango', 30: 'Kiwi', 40: 'Sberry', 50: 'Orange'}
>>> for x in d1:
```

```
...     print(x)
                       ...
                       10
                       20
                       30
                       40
                       50
```
----

8) items():

=>This Function is used for obtaining (Key,value) from non-empty dict object.

```
=>Syntax:         varname=dictobj.items()
                              (OR)
                             dictobj.items()
```

Examples:

```
>>> d1={10: 'Guava', 20: 'Mango', 30: 'Kiwi', 40: 'Sberry', 50: 'Orange'}
>>> print(d1)----{10: 'Guava', 20: 'Mango', 30: 'Kiwi', 40: 'Sberry', 50:
'Orange'}
>>> kvs=d1.items()
>>> print(kvs,type(kvs))---dict_items([(10, 'Guava'), (20, 'Mango'), (30,
'Kiwi'), (40, 'Sberry'),                                                   (50, 'Orang
------------------------------------
>>> for kv in kvs:
...     print(kv)
                       ...
                       (10, 'Guava')
                       (20, 'Mango')
                       (30, 'Kiwi')
                       (40, 'Sberry')
                       (50, 'Orange')
=====OR=======
>>> for kv in d1.items():
...     print(kv) ...
                       (10, 'Guava')
                       (20, 'Mango')
                       (30, 'Kiwi')
                       (40, 'Sberry')
                       (50, 'Orange')
=======OR=================
>>> for k,v in d1.items():
...     print(k,"--->",v) ...
                       10 ---> Guava
                       20 ---> Mango
                       30 ---> Kiwi
                       40 ---> Sberry
                       50 ---> Orange
```
----

9) update():

=>This Function is used for updating(Newly Inserting or Changing the old value of Value with New Value with Same Key) One dict object with another dict object

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

Examples:

```
----------------
>>> d1={10:"Python",20:"Data Science"}
>>> d2={30:"Django",40:"R"}
>>> d1.update(d2)------------>>> print(d1)
{10: 'Python', 20: 'Data Science', 30: 'Django', 40: 'R'}
>>> print(d2)------------{30: 'Django', 40: 'R'}
---------------------------------------------------------
>>> d1={10:"Python",20:"Data Science"}
>>> d2={30:"Django",40:"R",10:"Python Prog"}
>>> d1.update(d2)
>>> print(d1)----{10: 'Python Prog', 20: 'Data Science', 30: 'Django', 40:
'R'}
>>> print(d2)----{30: 'Django', 40: 'R', 10: 'Python Prog'}
=================================X=====================================
```

Misc Cases with Dict:

```
--------------------------------
>>> d1={1:["Python","Data
Scienece","Django"],2:("C","DS"),3:{"Java","Servlets","JSP","JDBC","Spring"
}  }
>>> print(d1,type(d1))
            {1: ['Python', 'Data Scienece', 'Django'], 2: ('C', 'DS'), 3:
{'Servlets', 'Java', 'JDBC', 'JSP', 'Spring'}} <class 'dict'>
>>> for k,v in d1.items():
...     print(k,"-->",v) ...
1 --> ['Python', 'Data Scienece', 'Django']
2 --> ('C', 'DS')
3 --> {'Servlets', 'Java', 'JDBC', 'JSP', 'Spring'}
-------------------------------------------------
>>> d1={10:["Tel","Hindi","Eng","Maths","Sci","Soc"],
... 12:["Maths","Phy","Che"],
... 13:["C","CPP","JAVA","PYTHON"] }
>>> print(d1,type(d1))
      {10: ['Tel', 'Hindi', 'Eng', 'Maths', 'Sci', 'Soc'], 12: ['Maths',
'Phy', 'Che'], 13: ['C', 'CPP', 'JAVA', 'PYTHON']} <class 'dict'>
>>> fork in d1.keys(): ...
print(k)
               ...
               10
               12
               13
>>> for v in d1.values():
...     print(v)
                       ...
                       ['Tel', 'Hindi', 'Eng', 'Maths', 'Sci', 'Soc']
                  ['Maths', 'Phy', 'Che']
      ['C', 'CPP', 'JAVA', 'PYTHON']
>>> for k,v in d1.items():
...     print(k,"<--->",v)
                               ...
                               10 <---> ['Tel', 'Hindi', 'Eng', 'Maths',
'Sci', 'Soc']
                               12 <---> ['Maths', 'Phy', 'Che']
                               13 <---> ['C', 'CPP', 'JAVA', 'PYTHON']
```

Misc Case with Inner list, Inner tuple, list of tuples converted into dict:

```
>>> l1=[[10,"RS"],[20,"JG"],[30,"TR"]]
>>> print(l1,type(l1))------------------[[10, 'RS'], [20, 'JG'], [30, 'TR']]
<class 'list'>
>>> d1=dict(l1)
>>> print(d1,type(d1))--------{10: 'RS', 20: 'JG', 30: 'TR'} <class 'dict'>
-------------------------------
>>> t1=( (10,"RS"),(20,"JG"),(30,"TR") )
>>> print(t1,type(t1))-----------((10, 'RS'), (20, 'JG'), (30, 'TR'))
<class 'tuple'>
>>> d1=dict(t1)
>>> print(d1,type(d1))-----------{10: 'RS', 20: 'JG', 30: 'TR'} <class
'dict'>
--------------------------------------------------------
>>> l1=[(10,"RS"),(20,"JG"),(30,"TR")]
>>> print(l1,type(l1))------------[(10, 'RS'), (20, 'JG'), (30, 'TR')]
<class 'list'>
>>> d1=dict(l1)
>>> print(d1,type(d1))-------------{10: 'RS', 20: 'JG', 30: 'TR'} <class
'dict'>
===============================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
```

====================================*&*===================================

==

Display the result of python program

==

=>To display the result of python program, we use a pre-defined function called print().
=>In otherwords print() is of the pre-defined function used for display the result of Python program.
=>print() can be used in various ways.

Syntax-1:

```
print(val1,val2,...val-n)
        (OR)
print(var1,var2.....var-n)
```

=>This syntax displays values directly or variable values

Examples:

```
-----------------
>>> a=10
>>> print(a,type(a))-----------------------10 <class 'int'>
>>> a=10
>>> b=20
>>> c=a+b
>>> print(a,b,c)---------------------------10 20 30
>>> print(100,200,300,400)-------------------100 200 300 400
-----------------------------------------------------------------------
```

Syntax-2: print(message1,message2,message3.....message-n)
here message1,message2,message3.....message-n are of type <class, 'str'> =>This syntax displays only messages.

Examples:

```
---------------
>>> print("Hyd")-----------Hyd
>>> print('Hello Python')-----------Hello Python
>>> print(s1+s2+s3)-----------------PythonMLDL
>>> print(s1+10)--------TypeError: can only concatenate str (not "int")
to str
>>> print(s1+str(10) )-----Python10
-----------------------------------------------------------------------
```

Syntax-3: print(msg cum value)
(or)
print(value cum message)

=>This syntax displays Values cum Messages OR Messages cum values

Examples:

```
----------------------
>>> a=10
>>> b=20
>>> c=a+b
>>> print("Val of a=",a)------------Val of a= 10
>>> print("Val of b=",b)----------Val of b= 20
>>> print("sum=",c)--------------sum= 30
----------------------------------------
>>> a=10
>>> b=20
>>> c=a+b
>>> print(a," is the value")-----------10  is the value
>>> print(b," is the value of b")-------20  is the value of b
>>> print(c," is the sum")------30  is the sum
-----------------------------------------------------------------------
>>> a=10
>>> b=20
>>> c=30
>>> d=a+b+c
>>> print("Sum of ",a,",",b," and ",c,"=",d)---Sum of  10 , 20  and  30
= 60
-----------------------------------------------------------------------
```

Syntax-4: print(msg cum value with format())
(or)

```
                 print(value cum message with  format() )
```

=>This syntax displays Values cum Messages OR Messages cum values with format()

Examples:

```
---------------------
>>> a=10
>>> b=20
>>> c=a+b
>>> print("Val of a=",a)-------------Val of a= 10
>>> print("Val of a={}".format(a))--------Val of a=10
>>> print("{} is the value of a".format(a))--------10 is the value of a
>>> print("sum={}".format(c))--------sum=30
>>> print("Sum of {} and {}={}".format(a,b,c))----Sum of 10 and 20=30
>>> a=10
>>> b=20
>>> c=30
>>> d=a+b+c
>>> print("Sum of {},{} and {}={}".format(a,b,c,d))---Sum of 10,20 and
30=60
-------------------------------------------------------------------------
```

Syntax-5:
```
                print(msg cum value with format specifiers )
                      (or)
                 print(value cum message with  format specifiers  )
```

=>This syntax displays Values cum Messages OR Messages cum values with format specifiers.

Examples:

```
-------------------
>>> a=10
>>> b=23
>>> c=a+b
>>> print("Val of a=",a)---------Val of a= 10
>>> print("Val of a={}".format(a))---------Val of a=10
>>> print("Val of a=%d" %a)--------Val of a=10
>>> print("%d is the val of a" %a)----------10 is the val of a
>>> print("Sum of %d and %d=%d" %(a,b,c) )-------Sum of 10 and 23=33
------------------------------------------------------------------
>>> a=10
>>> b=20
>>> c=a+b
>>> print("sum of %f and %f=%f" %(a,b,c))------sum of 10.000000 and
20.000000=30.000000
>>> print("sum of %0.2f and %0.3f=%0.2f" %(a,b,c))---sum of 10.00 and
20.000=30.00
>>> print("sum of %0.2f and %0.3f=%d" %(a,b,c))-------sum of 10.00 and
20.000=30
---------------------
>>> stno=10
>>> sname="Rossum"
>>> print("My number is %d and Name is %s " %(stno,sname))---
```

```
                                            My number is 10 and Name is
Rossum
>>> print("My number is %d and Name is '%s' " %(stno,sname))
                                            My number is 10 and Name is
'Rossum'
```

==================================**==================================

===

Reading the data from Keyboard in Python Program

===

=>To read the data dynamically from Keyboard, we have two pre-defined Functions. They are

1) input()
2) input(message)

----

1) input():

=>This function is used for reading the data dynamically from Key board in the form of str always.

=>Syntax:- varname=input()

=>here varname is an object of <class,'str'>. We can convert str value into any other data type by using Type Casting Techniques.

Examples:

```
#Program for accepting two numerical values and add them
#sumex3.py   print("Enter
Two   Value:")   a=float(
input()      )   b=float(
input()  )
print("Sum of {} and {}={}".format(a,b,a+b))
```

==

2) input(message):

=>This Function is used for reading the data from Key Board by displaying User-Prompting Message which is of type str.

=>Syntax:- var name=input("Message")

=>here varname is an object of <class,'str'>. We can convert str value into any other data type by using Type Casting Techniques.

=>"Message" represents User-Prompting Message.

===================================**===================================

==

Flow Control statement

==

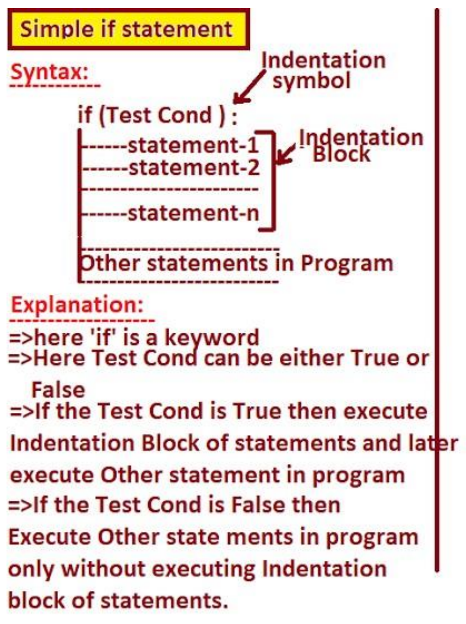
==

Conditional statements

==

Simple if statement

------------------ex



Simple if statement

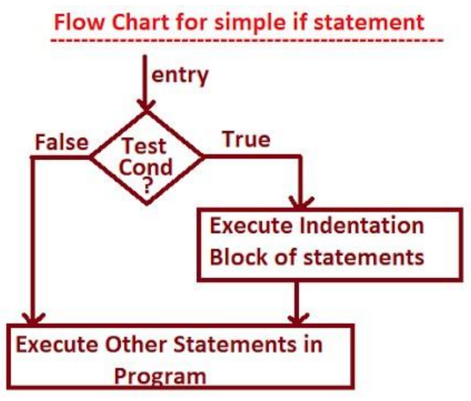
Syntax:

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

Indentation symbol
Indentation Block

Explanation:

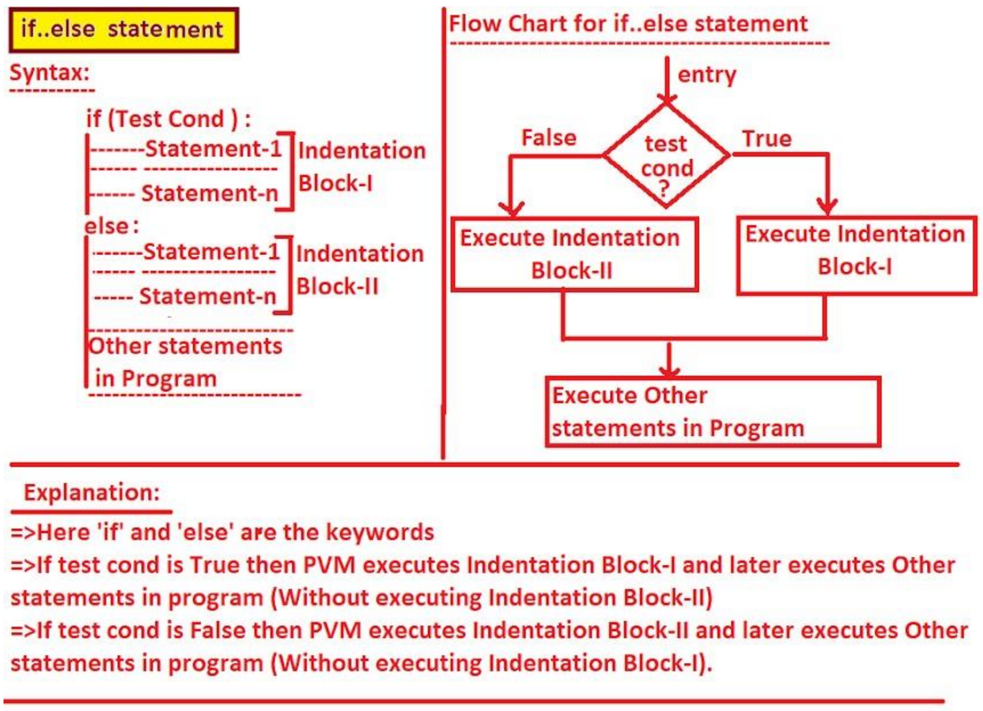
=>here 'if' is a keyword
=>Here Test Cond can be either True or False
=>If the Test Cond is True then execute Indentation Block of statements and later execute Other statement in program
=>If the Test Cond is False then Execute Other state ments in program only without executing Indentation block of statements.


Flow Chart for simple if statement
entry
False
Test Cond ?
True
Execute Indentation Block of statements
Execute Other Statements in Program

```
Ex: #PosNegZero.py
    n=int(input("Enter Value of n:"))
if(n>0):    print("{} is
POSSITIVE".format(n))     if(n<0):
print("{} is NEGATIVE".format(n))
if(n==0):
      print("{} is ZERO".format(n))
print("Program execution completed")
```

==

If else statement:



if..else statement

Syntax:

```
if (Test Cond ) :
|-------Statement-1 ]Indentation
|------ ----------------- ]Block-I
|----- Statement-n  ]
else:
|------Statement-1 ]Indentation
|----- ----------------- ]Block-II
|----- Statement-n  ]
|---------------------------
|Other statements
|in Program
---------------------------
```

Flow Chart for if..else statement
entry
test cond ?
False
True
Execute Indentation Block-II
Execute Indentation Block-I
Execute Other statements in Program

Explanation:

=>Here 'if' and 'else' are the keywords

=>If test cond is True then PVM executes Indentation Block-I and later executes Other statements in program (Without executing Indentation Block-II)

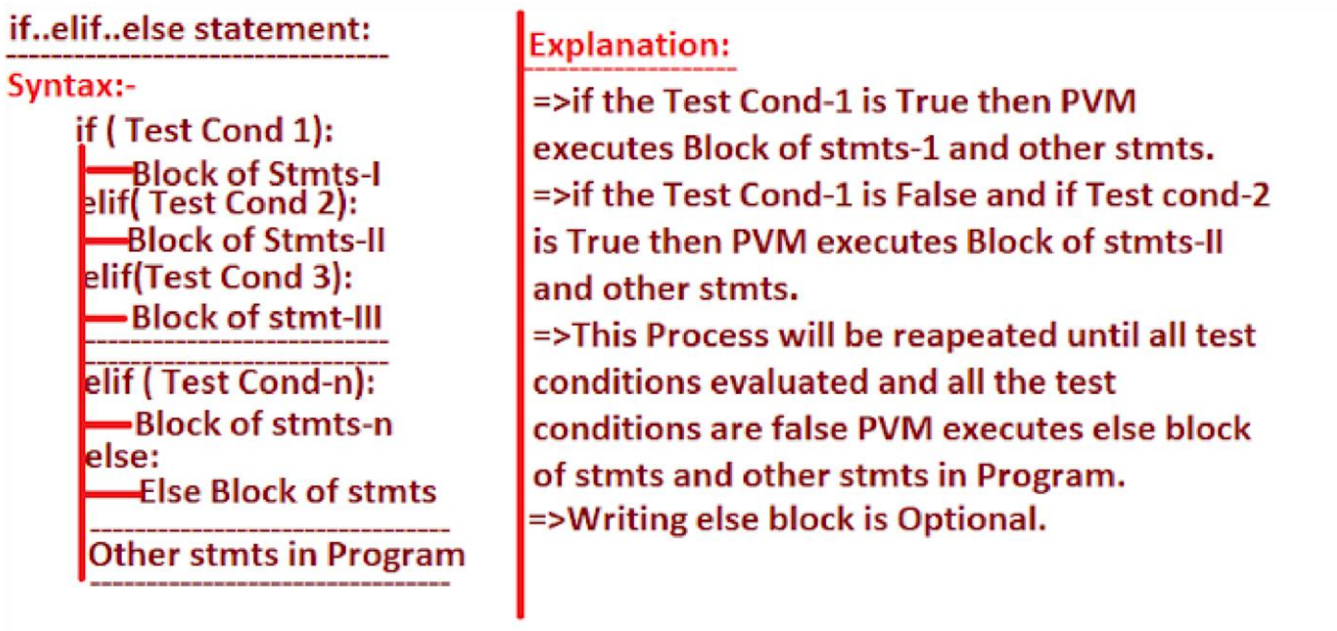
=>If test cond is False then PVM executes Indentation Block-II and later executes Other statements in program (Without executing Indentation Block-I).

Ex:

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

==

if elif else statement ------------------------



if..elif..else statement:

Syntax:-

```
if ( Test Cond 1):
    Block of Stmts-I
elif( Test Cond 2):
    Block of Stmts-II
elif(Test Cond 3):
    Block of stmt-III
--------------------------
--------------------------
elif ( Test Cond-n):
    Block of stmts-n
else:
    Else Block of stmts
--------------------------
Other stmts in Program
--------------------------
```

Explanation:

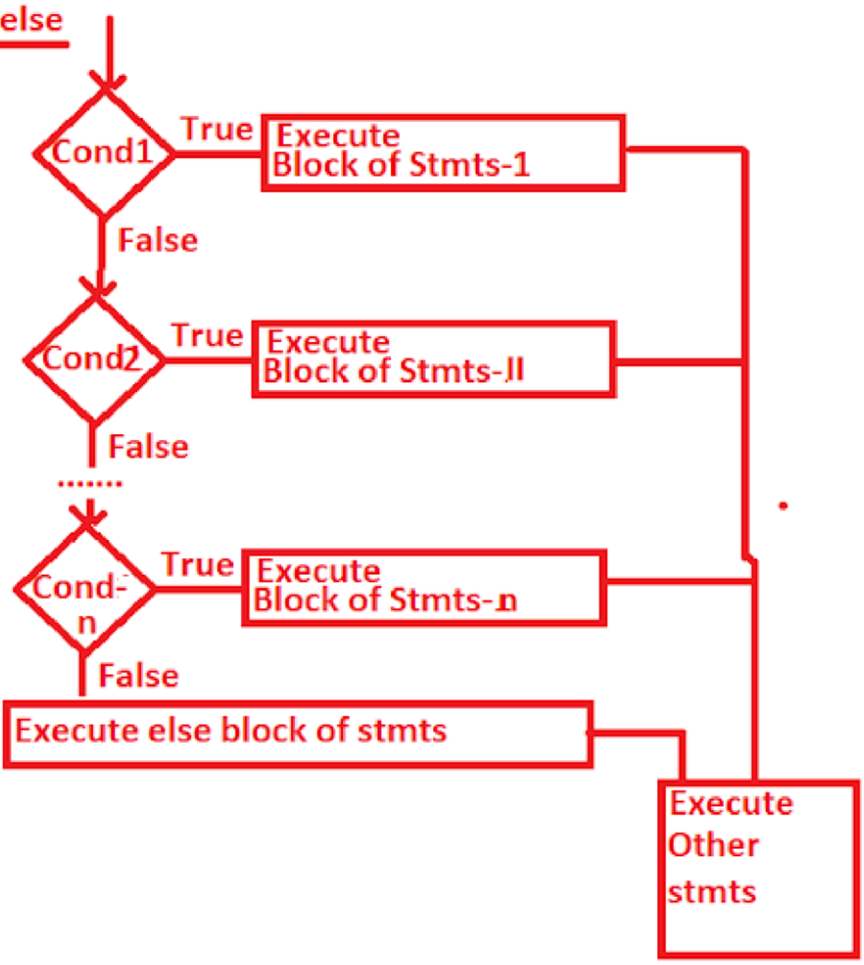
=>if the Test Cond-1 is True then PVM executes Block of stmts-1 and other stmts.

=>if the Test Cond-1 is False and if Test cond-2 is True then PVM executes Block of stmts-II and other stmts.

=>This Process will be reapeated until all test conditions evaluated and all the test conditions are false PVM executes else block of stmts and other stmts in Program.

=>Writing else block is Optional.

flow chart for if..elif..else



Ex:

```
d=int(input("Enter any digit:")) # d=0 1 2 3 4 5 6 7 8 9
if(d==0):   print("{} is ZERO:".format(d)) elif(d==1):
print("{} is ONE:".format(d)) elif(d==2):     print("{}
is TWO:".format(d)) elif(d==4):    print("{} is
FOUR:".format(d)) elif(d==3):      print("{} is
THREE:".format(d)) elif(d==6):     print("{} is
SIX:".format(d)) elif(d==5):       print("{} is
FIVE:".format(d)) elif(d==7):      print("{} is
SEVEN:".format(d)) elif(d==8):     print("{} is
EIGHT:".format(d)) elif(d==9):     print("{} is
NINE:".format(d)) elif(d>9):
      print("{} is NUMBER:".format(d)) else:
      print("{} is -ve number".format(d))
```

==

Match case statement.

==

=>here "match case" is one the new feature in Python 3.10 Version onwards
=>match case statement is recommended to take in deciding Predesigned Conditions.

=>**Syntax:**

```
                 match(Choice Expr):
       case Choice Label1:
                             Block of Stements-1
       case Choice Label2:
                             Block of Stements-2
       case Choice Label3:
                                     Block of Stements-3
                         ----------------------------
             case Choice Label-n:
     Block of Stamens-n           case _:
                                     default Block of Statements
```

Other Statements in Program

Explanation:

=>here "match" and "case" are the keywords
=>"Choice Expr" represents either int or str or bool
=>If "Choice Expr" is matching with "case label1" then PVM executes Block of Staements-1 and later executes Other statements in program.
=>If "Choice Expr" is matching with "case label2" then PVM executes Block of Staements-2 and later executes Other statements in program.
=>In General "Choice Expr" is trying match with case label-1, case label-2,....case label-n then PVM executes corresponding block of statements and later executes Other statements in program.
=>>If "Choice Expr" is not matching with any of the specified case labels then PVM executes Default Block of Statements which are written default case block (case _) and later executes Other statements in program.
=>Writing default case block is optional and If we write then it must be written at last (Otherwise we get SyntaxError)

==

Looping Statements

==

=>The Purpose of Looping or Iterative or Repetative Statements is that Performs ceratin operation Repeatedly for Finite Number of Times until Condition Becomes False.

=>In Python Programming, we have Two types of Looping statements. They are

1. while loop or   while...else loop
2. for loop   or   for ..else loop

=>At the writing Looping level programs, we must ensure that there must 3 parts. They are

1. Initlization Part (Where to start)
2. Conditional Part (After How many times to stop)
3. Updation Part (How much step to move forawrd or backward)

while loop or while...else loop:
===================================

a) while loop (or) while .. else loop

Syntax:

```
while (Test Cond ) :
|-----Block of statements
|
|
Other Statements in Program
----------------------------------------
```

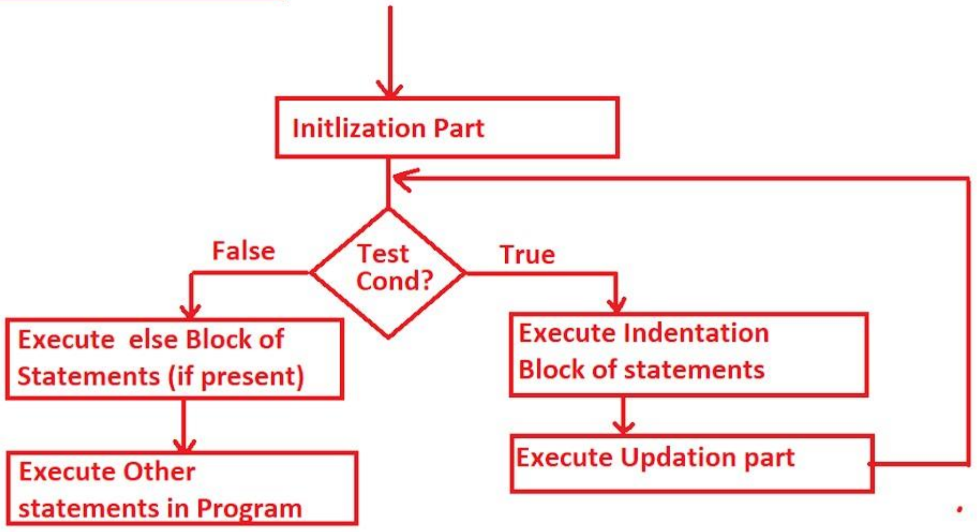
OR

```
while (Test Cond ) :
|-----Block of statements
|-----
|-----
else:
|-----Else Block of statements
|-----
|
Other statements in Program
```

Explanation:-

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

Flow Chart for while loop


Initlization Part
False
Test Cond?
True
Execute else Block of Statements (if present)
Execute Indentation Block of statements
Execute Other statements in Program
Execute Updation part

2. for loop or for ...else loop:

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

Syntax1:-

```
-----------
                  for varname  in  Iterable_object:
----------------------------------------
                              Indentation block of stmts
                        ----------------------------------------
      -------------------------------------------------
                      Other statements in Program
                      -------------------------------------------------
```

Syntax2:

```
---------------
                  for   varname   in Iterable_object:
----------------------------------------
                              Indentation block of stmts
                        ----------------------------------------
      else:
                        ----------------------------------------
            else block of statements
                        ----------------------------------------
      -------------------------------------------------
                      Other statements in Program
                      -------------------------------------------------
```

Explanation:

=>Here 'for' and 'else' are keywords
=>Here Iterable_object can be Sequence(bytes,bytearray,range,str), list(list,tuple),set(set,frozenset) and dict.
=>The execution process of for loop is that " Each of Element of Iterable_object selected,placed in varname and executes Indentation block of statements".This Process will be repeated until all elements of Iterable_object completed.
=>After execution of Indentation block of statements, PVM executes else block of statements which are written under else block and later PVM executes Other statements in Program.
=>Writing else block is optional.

Transfer control statements

==

Break Statement

=>break is a key word
=>The purpose of break statement is that "To terminate the execution of loop logically when certain condition is satisfied and PVM control comes of corresponding loop and executes other statements in the program".
=>when break statement takes place inside for loop or while loop then PVM will not execute corresponding else block of statementys but it always executes other statements in the program.

=>Syntax:

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

=>Syntax:

```
                      while(Test Cond-1):
                        ------------------------------
             if (test cond-2):
break
                           ------------------------------
                          ------------------------------
```

Continue Statement:

=>continue is a keyword
=>continue statement is used for making the PVM to go to the top of the loop without executing the following statements which are written after continue statement for that current Iteration only. =>continue statement to be used always inside of loops. =>when we use continue statement inside of loop then else part of corresponding loop also executes provided loop condition becomes false. -----------------

=>Syntax:-

```
                      for varname   in Iterable-object:
                       ------------------------------------------
           if ( Test Cond):                            continue
                       statement-1  # written after continue statement
                       statement-2
     statement-n
                      -----------------------------------------
                      -----------------------------------------
```

=>Syntax:-

```
                      while (Test Cond):
```

```
                      ------------------------------------------
          if ( Test Cond):                              continue
                      statement-1  # written after continue
statement
                      statement-2
     statement-n
                      ------------------------------------------
                      ------------------------------------------
==================================**====================================
```

Functions in Python

=>Purpose of Functions is that " To Perform Certain Operation and Provdes Code Re-Usability".

--- =>**Definition of Function:**

=>Sub Program of Main Program is called Function

(OR)

=>A part of main program is called Function.

Parts of Functions

=>When we define a function, we must ensure there must exist 2 parts. They are

1) Function Definition
2) Function Call.

=>For Every Function Call , there must exist Function Definition otherwise we get NameError.

=> Function Definition will execute by calling though the function call otherwise Function Definition will not execute.

=>The Perticular Function Definition will exist one time where we can have multiple Function calls for one function Definition

===

Types of Languages in the context of Functions

===

=>In the context of Functions, we have two types of languages. They are

a) Un-Structured Programming Lang

b) Structured Programming Lang

----

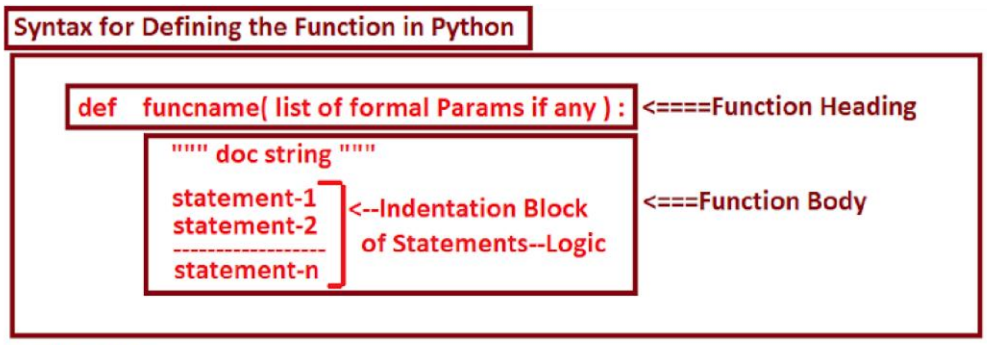
a) Un-Structured Programming Lang

--- =>In These Programming Languages, we don't have the concept of Functions.

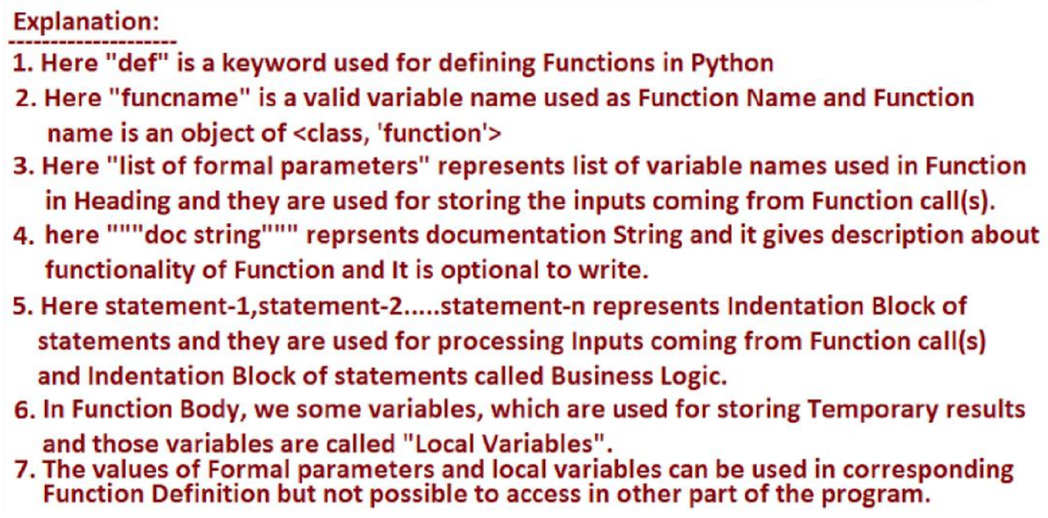
=>Since Un-Structured Programming Languages does not contain the concept of Functions, so that they the following Limitations.

1. Application Development time is More
2. Application Memory Space is More
3. Application Execution time is More (slow)
4. Application Performance is degraded
5. Redundancy (Duplication or replication) of the code is more.

=>Example: GW-BASIC


Syntax for Defining the Function in Python

```
def   funcname( list of formal Params if any ) :   <====Function Heading
        """ doc string """
        statement-1  <--Indentation Block           <===Function Body
        statement-2     of Statements--Logic
        -----------------
        statement-n
```


Explanation:
----
1. Here "def" is a keyword used for defining Functions in Python
2. Here "funcname" is a valid variable name used as Function Name and Function name is an object of <class, 'function'>
3. Here "list of formal parameters" represents list of variable names used in Function in Heading and they are used for storing the inputs coming from Function call(s).
4. here """doc string""" reprsents documentation String and it gives description about functionality of Function and It is optional to write.
5. Here statement-1,statement-2.....statement-n represents Indentation Block of statements and they are used for processing Inputs coming from Function call(s) and Indentation Block of statements called Business Logic.
6. In Function Body, we some variables, which are used for storing Temporary results and those variables are called "Local Variables".
7. The values of Formal parameters and local variables can be used in corresponding Function Definition but not possible to access in other part of the program.

==

b) Structured Programming Languages

=>In These Programming Languages, we have the concept of Functions.
=>Since Structured Programming Languages contains the concept of Functions, so that they the following Advantages.

1. Application Development time is Less
2. Application Memory Space is Less
3. Application Execution time is Less (fast)
4. Application Performance is enhanced(Improved) 5. Redundancy (Duplication or replication) of the code is Minimized.

Examples: C, CPP, JAVA, .NET, PYTHON...etc

==

Arguments and parameters

==

Parameters

=>Parameters are the Varaibles used in Function Heading and they are also called Formal Parameters.
=>The purpose of formal Parameter is that To store the inputs coming from function calls. ----------------------

Arguments:

=>Arguments(Actual Arguments) are also called Variables used in Function Calls.
=>The Relation between Arguments and parameters is that All the values of Arguments are passing to Formal Parameters.

=>The data passed from Function Calls to Function definition in the form arguments or Argument values from Function calls to Function Definition in the form Formal Parameters.
==

Types of Arguments or Parameters(In Functions)

== =>Based on the values of arguments from function call passing to the parameters of the function definition, the arguments are classified into 5 types. They are

1. Positional Arguments
2. Default Arguments
3. Key Arguments
4. Variable Length Arguments
5. Key Word Variable Length Arguments

Positional Arguments

==

=>It is one the default arguments passing mechanism.
=>The concept of Positional Arguments is that Number of arguments in Function call must be equal to Number of Formal Parameters in Function Definition.
=>The concept of Positional Arguments also recommends to maintain order and meaning for higher accuracy for Data .
=>Positional Arguments passing mechanism given by PVM as First Priority.

=>Syntax for Function Definition

```
---------------------------------------------------------
            def      function name(param1,param2,...param-n):
--------------------------------------------
                        Block of statement---Processing Logic
                        --------------------------------------------
```

=>Syntax for Function Call:

```
--------------------------------------------- function
name(arg1,arg2,....arg-n)
```

=>Here the values of arg1,arg2,....arg-n are passing param1, param2....param-n respectively.

2) Default Parameters (or) arguments

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

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

Syntax: for Function Definition with Default Parameters

```
------------------------------------------------------------------------
----------------- def   functionname(param1,param2,....param-n-
1=Val1, Param-n=Val2):
-------------------------------------------------------------
--------
-------------------------------------------------------------
------
```

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

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

3) Keyword Parameters (or) arguments

==

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

Syntax for function definition:-

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

Syntax for function call:-

```
-------------------------------------------------
      functionname(param-n=val-n,param1=val1,param-n-1=val-n-1,......)
```

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

4) Variables Length Parameters (or) arguments

==

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

Syntax for function definition with Variables Length Parameters: ------

```
------------------------------------------------------------------   def
functionname(list of formal params,  *param1,param2=value) :
       --------------------------------------------------
      --------------------------------------------------
```

=>Here *param1 is called Variable Length parameter and it can hold any number of argument values (or) variable number of argument values and *param1 type is <class,'tuple'>

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

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

5)Key Word Variables Length Parameters (or) arguments
===

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

Syntax for function definition with Keyword Variables Length Parameters:

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

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

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

Final Syntax:

```
def  funcname(PosFormal parms, *Varlenparams, default params,
**kwdvarlenparams):
       -------------------------------------------------
       -------------------------------------------------
```

===
Global variables and Local Variables
===

=>Local Variables are those which are defined / used Inside of Function Body.
=>Local Variables can be used for storing temporary result of Function.
=>The Values of Local Variables can be used inside of same Function Definition but not
=>possible to access in other part of the program and in other Function Definition.

=>Global variables are those which are used for Representing Common

values for Multiple Different Function calls and Saves the Memory Space.
=>Global variables must be defined before all function calls. So that we can access the global variable values in all the function definitions. Otherwise we can't access.

=>Syntax:

```
                      Var1=Val1
                      Var2=Val2
                 -------------
Var-n=Val-n

                      def     functionname1(.....):
                            var22=val22
         var23=val23
                                  ------------------
                      def     functionname2(.....):
                            var32=val32
         var33=val33
```

=>here Var1,Var2...Var-n are called Global variables for functiondefinition1() and func
=>here Var22,Var23... are called Local Varbales variables functiondefinition1()
=>here Var32,Var33... are called Local Varbales variables in functiondefinition2()
==
Examples:

```
#globallocalvarex3.py def
learnML():
       sub1="Machine Learning"  # here sub1 is called Local Variable
       print("\nTo Learn and Code in  '{}' , we use '{}' Programming
".format(sub1,lang))
 #print(sub2,sub3)---Error bcoz sub2 and subj3 are local variables in
other Functions
 def
learnDL():
      sub2="Deep Learning"  # here sub2 is called Local Variable
print("\nTo Learn and Code in  '{}' , we use '{}' Programming
".format(sub2,lang))
      #print(sub1,sub3)---Error bcoz sub1 and subj3 are local
variables in other Functions def   learnIOT():
       sub3="IOT"  # here sub3 is called Local Variable
       print("\nTo Learn and Code in  '{}' , we use '{}' Programming
".format(sub3,lang))
 #print(sub1,sub2)---Error bcoz sub1 and subj1 are local variables in
other Functions
#main program
lang="PYTHON"  # Global Variable
learnML() learnDL() learnIOT()
```
=================================X===================================

Examples:

```
----------------------
#globallocalvarex4.py def
learnML():
 sub1="Machine Learning"  # here sub1 is called Local Variable
print("\nTo Learn and Code in  '{}' , we use '{}' Programming
".format(sub1,lang))
 def
learnDL():
      sub2="Deep Learning"  # here sub2 is called Local Variable
print("\nTo Learn and Code in  '{}' , we use '{}' Programming
".format(sub2,lang))
 def
learnIOT():
       sub3="IOT"  # here sub3 is called Local Variable
       print("\nTo Learn and Code in  '{}' , we use '{}' Programming
".format(sub3,lang))

#main program
learnML() learnDL()
learnIOT()
lang="PYTHON"  # Global Variable  --here we can' t access Variable lang
in learnML(), learnDL() and LearnIOT() bcoz It is defined after Function
Call.
```

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

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

Syntax:

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

```
Examples:
-----------------------
#globalvarex1.py a=10
def  access1():
     print("Val of a=",a) # Here we are accessing the global variable
'a' and No Need to use global kwd.

#main program access1()
---------------------------------------
#globalvarex2.py a=10 def  access1():  global a  # refering global
Varaible before its updation / Modification  a=a+1 # Here we are
modifying the global variable value then we need to use global keyword.
       print("Val of a inside of access1()=",a) # 11
#main program
print("Val of a in main before access1():",a) # 10 access1()
print("Val of a in main after access1():",a) # 11
-----------------------------------------------------------------------
--------- Examples:
------------------    #globalvarex3.py    def
update1():   global  a,b  #  refering  global
Variables.  a=a+1  #updating global Variable a
b=b+1  #updating global Variable b
def update2():  global a,b  # refering global
Variables.
      a=a*10 #updating global Variable a
b=b*10 #updating global Variable b

#main program
a,b=1,2  # here a and b are called Global Variables
print("Val of a={} and Value of b={} in main program before update
functions :".format(a,b))
#  Val of a=1 and Value of b=2 in main program before update functions
: update1()
print("Val of a={} and Value of b={} in main program after
update1():".format(a,b))
#Val of a=2 and Value of b=3 in main program after update1(): update2()
print("Val of a={} and Value of b={} in main program after
update2():".format(a,b))
#Val of a=20 and Value of b=30 in main program after update1():
```

==

global and local variables and globals()

==

=>When we come acrosss same global Variable names and Local Vraiable Names in same function definition then PVM gives preference for local variables but not for global variables.
=>In this context, to extract / retrieve the values of global variables names along with local variables, we must use globals() and it returns an object of <class,'dict'> and this dict object stores all global variable Names as Keys and global variable values as values of value.

=>Syntax:-

```
            var1=val1
var2=val2
               --------------
  var-n=val-n  # var1, var2...var-n are called global Variables
            def    functionname():
       ------------------------
var1=val11                    var2=val22
             -----------------
                   var-n=val-nn  #  var1, var2...var-n are called local
Variables
                      # Extarct  the global variables values
         dictobj=globals()          -------------------
-----              globalval1=dictobj['var1']   #   or
dictobj.get("var1") or globals()['var1']
                      globalval2=dictobj['var2']  # or
dictobj.get("var2") or globals()['var2']
---------------------------------------------------
---
-----------------------------------------------------
====================================================================
```

Examples:

```
============
#globalsfunex3.py
a=10 b=20 c=30 d=40
def   operations():
      obj=globals()    for gvn,gvv in obj.items():
      print("\t{}---->{}".format(gvn,gvv))
print("="*50)
       print("\nProgrammer-defined Global Variables")
      print("="*50)    print("Val
of a=", obj['a'])      print("Val
of b=", obj['b'])      print("Val
of c=", obj['c'])      print("Val
of d=", obj['d'])
print("="*50)
       print("\nProgrammer-defined Global Variables")
      print("="*50)    print("Val of
a=", obj.get('a'))     print("Val of
b=", obj.get('b'))     print("Val of
c=", obj.get('c'))     print("Val of
d=", obj.get('d'))
       print("="*50)
      print("\nProgrammer-defined Global Variables")
print("="*50)
      print("Val of a=", globals().get('a'))
print("Val of b=", globals().get('b'))       print("Val
of c=", globals().get('c'))       print("Val of d=",
globals().get('d'))    print("="*50)
       print("\nProgrammer-defined Global Variables")
       print("="*50)
      print("Val of a=", globals()['a'])
print("Val of b=", globals()['b'])      print("Val
of c=", globals()['c'])      print("Val of d=",
globals()['d'])
```

```
       print("="*50)
================================================
#main program operations()
=================================================
```

Examples:

```
-----------------------
#Program for demonstrating globals()
#globalsfunex2.py
a=10 b=20 c=30
d=40  # Here  a,b,c,d are called Global Variables def
operation():
      a=100
b=200      c=300
      d=400   # Here  a,b,c,d are called Local Variables
res=a+b+c+d+globals()['a']+globals().get('b')+globals()['c']+glo
bals()['d']
       print(res)

#main program operation()
```

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

Anonymous Functions OR Lambda Functions

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

=>Anonymous Functions are those which does not contain any name.
=>The purpose of Anonymous Functions is that " To Perform Instant Operations".
=>Instant Operations are those which are used at that point of time only and not longer interested to re-use in other part of program.
=>Anonymous Functions contains single executable statement only.
=>To define Anonymous Functions, we use a keyword "lambda". Hence Anonymous
Functions are called Lambda Functions
=>Anonymous Functions returns its result automatically (No need to use return statement).

=>Syntax:

```
                      varname=lambda params-list : expression
```

Explanation:

=>Here varname is an object of <class,'function'> and it itself can be for calling Anonymous
Functions.
=>lambda is a keword used for defining Anonymous Functions.
=>params-list represents list of variable names used for holding input values coming from function calls.
=>expression represents single executable statement.

By Using Normal Function Def   Normal
Function Call

```
--------------------------------------------------------------------------
def     addop(a,b):
res=addop(10,20)
      c=a+b                                              print(res)----
--30
       return c
--------------------------------------------------------------------------
By Using Anonymous Function Def                            Anonymous
Function Call
------------------------------------------------------------------
sumop=lambda a,b : a+b
       res=sumop(10,20)

print(res) # 30
```

==

Special Functions in Python

==

=>In Python Programming, we have 3 types of special Functions. They are
1) filter()
2) map()
3) reduce()

----

1) filter():

=>filter() is used for "Filtering out some elements from list of elements by applying to function".
=>Syntax:- varname=filter(FunctionName, Iterable_object)

Explanation:

=>here 'varname' is an object of type <class,'filter'> and we can convert into any iteratable object by using type casting functions.
=>"FunctionName" represents either Normal function or anonymous functions.
=>"Iterable_object" represents Sequence, List, set and dict types.
=>The execution process of filter() is that " Each Value of Iterable object sends to Function Name. If the function return True then the element will be filtered. if the Function returns False then that element will be neglected/not filtered ". This process will be continued until all elements of Iterable object completed.

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

2) map()

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

=>map() is used for obtaining new Iterable object from existing iterable object by applying old iterable element to the function. =>In otherwords, map() is used for obtaining new list of elements from existing list of elements by applying old list elements to the function.

=>Syntax:- varname=map(FunctionName,Iterable_object)

=>here 'varname' is an object of type <class,map'> and we can convert into any iteratable object by using type casting functions.
=>"FunctionName" represents either Normal function or anonymous functions.
=>"Iterable_object" represents Sequence, List, set and dict types.
=>The execution process of map() is that " map() sends every element of iterable object to the specified function, process it and returns the modified value (result) and new list of elements will be obtained". This process will be continued until all elements of Iterable_object completed.

================================
reduce()
================================

=>reduce() is used for obtaining a single element / result from given iterable object by applying to a function.
=>Syntax:-

varname=reduce(function-name,iterable-object)

=>here varname is an object of int, float,bool,complex,str only =>The reduce() belongs to a pre-defined module called" functools".
--------------------------------------- **Internal Flow of reduce()**
----
step-1:- Initially,reduce() selects two First values of Iterable object and place them in First var and Second var . step-2:- The function-name(lambda or normal function) utilizes the values of First var and
Second var applied to the specified logic and obtains the result.
Step-3:- reduce () places the result of function-name in First variable and reduce()
selects the succeeding element of Iterable object and places in second variable.
Step-4: repeat Step-2 and Step-3 until all elements completed in Iterable object and returns the result of First Variable

---

===================================*&*===================================
==
Modules in Python
==
=>We know that Functions concept makes us understand How to perform operations and we can re-use within the same program but not able to re-use the functions across the programs.
=>To reuse the functions and global variables across the programs, we must use the concept of MODULES.

=>Definition of Modules:

=>A Module is a collection of variables (global variables) , Functions and Classes.

=>Types of Modules:

=>In Python Programming, we have two types of Modules. They are
1) Pre-defined (or) Built-in Modules
2) Programmer or user or custom-defined modules.
----

1) Pre-defined (or) Built-in Modules:

=>These modules are developed by Python Language Developers and they are avialable in Python Software (APIs) and they are used python programmers for dealing with Universal Requirements.

Examples: math cmath functools sys calendar os re threading pickle random.......etc =>Out of many pre-defined modules, in python programming one implicit pre-defined module imported to every python program called "builtins" . --
--

2) Programmer or user or custom-defined modules:

=>These modules are developed by Python Programmers and they are avialable in Python Project and they are used by other python programmers who are in project development to deal with common requirements.
=>Examples:- aop mathsinfo icici ......etc

===

Development of Programmer-Defined Module

===

=>To develop Programmer-Defined Modules, we must use the following steps

Step-1 : Define Variables (Global variables)
Step-2: Define Functions
Step-3: Define Classes

=>After developing step-1, step-2 and step-3 , we must save on some file name with an extension .py (FileName.py) and it is treated as module name.
=>When a file name treated as a module name , internally Python execution environment creates a folder automatically on the name of __pycache__ and it contains module name on the name "filename.cpython310.pyc ".
------------------- **Examples:**

__pycache__ <-----Folder Name
-- aop.cpathon-310.pyc <--------------------Module Name mathsinfo.cpython-310.pyc <--------------Module Name icici.cpython-310.pyc <----------------------Module Name

==
Number of approaches to re-use Modules
==

=>We know that A Module is a collection of variables, Functions and Classes.
=>To re-use the features(Variable Names, Function Names and Class Names) of module, we have 2 approaches.They are
1) By using import statement
2) By using from.... import statement.

---

1) By using import statement:

=>'import' is a keyword
=>The purpose of import statement is that "To refer or access the variable names, function names and class names in current program"
=>we can use import statement in 4 ways. -------------------
=>Syntax-1: import module name

=>This syntax imports single module

Example: import icici
import aop
import mathsinfo

=>Syntax-2: import module name1, module name2....Module name-n
----
=>This syntax imports multiple modules

Example: import icici,aop,mathsinfo

=>Syntax-3: import module name as alias name

=>This syntax imports single module and aliased with another name
----
Example: import icici as i
import aop as a
import mathsinfo as m

=>Syntax-4: import module name1 as alias name, module name2 as alias name......module name-n as alias name
----
=>This syntax imports multiple modules and aliased with another names
----
Example: import icici as i, aop as a , mathsinfo as m

=>Hence after importing all the variable names, Function names and class names by using "import statement" , we must access variable names, Function names and class names w.r.t Module Names or alias names.

```
Module Name.Variable Name
Module Name.Function Name
Module Name.Class  Name
```

```
                        (OR)
                Alias Name.Variable Name
                Alias Name.Function Name
                Alias Name.Class Name
```

==

2) By using from.... import statement.

==

=>Here "form" "import" are the key words
=>The purpose of from.... import statement is that " To refer or access the variable names, function names and class names in current program directly without writing module name." => we can use from.... import statement in 3 ways.

Syntax-1: from module name import Variable Names,Function Names, Class Names

----

=>This syntax imports the Variable Names,Function Names, Class Names of a module.

Example:

```
from calendar  import  month
from aop import addop,subop
from mathinfo   import pi,e
from icici import bname,addr
```

Syntax-2: from module name import Variable Names as alias name,Function Names as alias name , Class Names as alias names.

=>This syntax imports the Variable Names,Function Names, Class Names of a module with alias Names

Example:

```
from calendar  import  month as m
from aop import addop as a,subop as s, mulop as m
from mathinfo   import pi as p ,e as k
from icici import bname as b, addr as a , simpleint as si
```

Syntax-3: from module name import *

=>This syntax imports ALL Variable Names,Function Names, Class Names of a module.
=>This syntax is not recommmended to use bcoz it imports required Features of Module and also import un-interrested features also imported and leads more main memory space.

Example:

```
from calendar   import  *
from aop import  *
from mathsinfo  import  *
```

=>Hence after importing all the variable names, Function names and class names by using "from ....import statement" , we must access variable names, Function names and class names Directly without using Module Names or alias names.

```
                    Variable Name
                    Function Name
                    Class  Name
```

=>Hence with "import statement" we can give alias name for module names only but not for Variables Names, Function Names and Class Names. Where as with "from ... import statement " we can give alias names for Variables Names, Function Names and Class Names but not for Module Name

==

realoding a modules in Python

==

=>To reaload a module in python , we use a pre-defined function called reload(), which is present in imp module and it was deprecated in favour of importlib module.

```
=>Syntax:-    imp.reload(module name)
                                    (OR)
                    importlib.reload(module name) -----recommended
```

=>Purpose / Situation:

---------------------------------- =>reaload() reloads a previously imported module.

=>if we have edited the module source file by using an external editor and we want to use the changed values/ updated values / new version of previously loaded module then we use reload().

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

```
#shares.py---file  and  treated  as  module  name  def
sharesinfo():
d={"Tech":19,"Pharma":11,"Auto":1,"Finance":00}
       return d

#main program
#sharesdemo.py
import shares
import time import
importlib def
disp(d):
     print("-"*50)
print("\tShare Name\tValue")
print("-"*50)    for sn,sv in
d.items():
          print("\t{}\t\t:{}".format(sn,sv))
else:          print("-"*50)
#main program
d=shares.sharesinfo()
disp(d) time.sleep(15)
importlib.reload(shares)  # relodaing previously imported module
d=shares.sharesinfo() # obtaining changed / new values of previously
imported   module
disp(d)
```

===*&*==========================

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

Packages in Python

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

Index:

===

Package in Python

===

=>The Function concept is used for Performing some operation and provides code re-usability within the same program and unable to provide code re-usability across programs.

=>The Modules concept is a collection of Variables, Functions and classes and we can re-use the code across the Programs provided Module name and main program present in same folder but unable to provide code re-usability across the folders / drives / enviroments.

=>The Package Concept is a collection of Modules.
=>The purpose of Packages is that to provide code re-usability across the folders / drives / enviroments.

=>To deal with the package, we need to the learn the following.
a) create a package
b) re-use the package

----

a) create a package:

=>To create a package, we use the following steps.
i) create a Folder
ii) place / write an empty python file called __init__.py iii) place / write the module(s) in the folder where is it considered as Package Name

Example:

```
                      bank           <-----Package Name
                     -----------
                           __init__.py    <----Empty Python File
                  simpleint.py  <--- Module Name
aopmenu.py-----Module Name                    aoperations.py---
Module Name                        runappl.py  <--- Module Name
```

== **b) re-use the package**

=>To the re-use the modules of the packages across the folders / drives / enviroments, we have to two approaches. They are
i) By using sys module

```
       ii) by using PYTHONPATH Environmental Variable Name -----------
          ------------------------------------------------------------
```

i) By using sys module:

```
-------------------------------------- Syntax:
-----------  sys.path.append("Absolute Path of Package")

=>sys is pre-defined module
=>path is a pre-defined object / variable present in sys module
=>append() is pre-defined function present in path and is used for
locating the package name of python( specify the absolute path)
```

Example:

```
sys.path.append("E:\\KVR-PYTHON-7AM\\ACKAGES\\BANK")
                      (or)
sys.path.append("E:\KVR-PYTHON-7AM\ACKAGES\BANK")
               (or)
sys.path.append("E:\KVR-PYTHON-7AM/ACKAGES/BANK")
------------------------------------------------------------------------
```

ii) by using PYTHONPATH Enviromental Variables:

```
------------------------------------------------------------------------
-
=>PYTHONPATH is one of the Enviromental Variable =>Search
for Enviromental Variable
Steps for setting :
------------------------------
                         Var name :   PYTHONPATH
                       Var Value : E:\KVR-PYTHON-7AM\PACKAGES\BANK

The overall path
                             PYTHONPATH= E:\KVR-PYTHON-
11AM\PACKAGES\BANK
------------------------------------------------------------------------
```

Core Python Completed

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