===
Getting started with Python
===

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

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

=>Versions of Python
====================
=>Python Programming Contains two Versions. They are
1) Python 2.x----- Here x ---> 1 2 3 4 5 6 7 -----outdated---
2) Python 3.x----> here x 1 2 3 4 5 4 6 7 8 9 10
=>Python 3.x does not contain backward compatability with Python 2.x
=>To down load Python 3.x software , we use www.python.org
=>Python Software and its updations are maintained by a Non-Commerical Organization called " Python Software Foundation(PSF) "

===
Python Programming Inspired from
===

=>Python Programming Inspired from 4 programming language

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

==
Real Time Applications of Python
==

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

1) Web Applications Development.---->
   a) Java------>Servlets , JSP
   b) C#.net---->ASP.net
   c) Python---->Django,Falsk, Bottle...etc
2) Gaming Application Development.
3) Artificial Intelligence-----Machine Learning and Deep Learning
4) Desk top GUI Applications
5) Image Processing applications.
6) Text Processing Applications
7) Business Applications.
8) Audio and Video Based Applications

9) Web Scrapping Applications / Web Harvesting Applications
10) Data Visulization.
11) Complex Math Calculations.
12) Scientific Applications
13) Software Development
14) Operating System
15) CAD and CAM based Applications
16) Embedded Applications
17) IOT Based Applications
18) Language Applications
19) Automation of Testing
20) Animation Applications
21) Data Analysis and Data Analystics
22) Education Sector
23) Computer Vision

Features of Python Programming

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

=>Python Programming Provides 11 features.

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

Features of Python Programming

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

=>Python Programming Provides 11 features.

1. Simple
2. Freeware and Open Source
3. Platform Independent
4) Dynamically Typed
5) Portable
6) Interpreted
7) High Level
8) Robust(Strong)
9) Extensible
10) Embedded

11) Extensive Third Party Library / API support
(Numpy,Pandas, Matplotlib,scikit, scipy...etc)
12) Both Procedure oriented(Core Python) and Object Oriented (Adv Python)

==========================
1. Simple
==========================

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

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

Definition of API (Application Programming Interface):

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

----

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

Def of Garbage Collector:

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

----

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

==
Freeware and Open Source
==

=>Freeware:

=> If any software is available Freely Downlodable then it called FreeWare.
Examples:-   PYTHON   and JAVA

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

=>Open Source:

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

=>Some of the Python Distributions are :

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

======================================
3. Platform Independent
======================================

Concept / Definition:

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

Property :

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

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

==
Portable
==

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

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

===
4) Dynamically Typed
===

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

1. Static Typed Programming Languages:

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

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

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

----
2. Dynamically Typed Programming Languages:

=>In This Programming Languages, Data type of the values need not specify by the programmer and more over data type of the value is implicitly decided by Python Execution Environment.
=>In Python Programming , all values are stored in the form of Objects and to cerate objects we need classes.

Examples: PYTHON

Examples:

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

6) Interpreted

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

1) Compilation Process:

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

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

2) Execution Phase:

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

7) High Level

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

Extensible and Embedded

Extensible:

=>Since Python Programming Provides its services (Programming Segments / snippets) to other languages for fullfillung its requiements easily.

Examples:-C Programs-can call The coding segments of PYTHON.

Embedded:

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

11) Extensive Third Party Library (or) API support

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

11) Extensive Third Party Library (or) API support

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

Data Representation in Python (or) Literals in Python

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

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

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

Rules for Using Variables in Python

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

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

Examples:

```
                ----------------
                            12abc=10-----invalid
                            -abc=20-------invalid
                            abc=123-----valid
                            a123=34----valid
                            _abc=34---valid
                            _sal_=2.3--valid
                            _123=2.3---valid
                            _=23----valid
      3) Within in the Variable Name , special symbols are not allowed
except
          Under Score (_)
                Examples:
                ----------------
                            tot  sal=2.3---invalid
                            tot$sal=2.3--invalid
                            tot_sal=2.3--valid
      4) All the Variables in Python are Case Sensitive.
           Examples:
           ----------------
                            age=99---valid
                            AGE=89---valid
                            Age=79---valid
      5) Keywords can't be used as Variables Names bcoz all the Key words
are  Reserved Words they have some specfic meaning to the language
Compilers.
           Examples:
           -----------------
                            if=12---invalid
                            while=23---invalid
                            else=45---invalid
                            if123=56---valid
                            _while=34----valid
                            IF=45----valid
                            int=12.34---valid
                            float=45----valid

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

      Examples:-
                >>> sal_of_an_employee=1.2--Valid--Not Recommended
                >>> emp_sal=1.2--Valid--Recommended
===========================X===================================
```

```
               ============================================
                        Variables (or) Identifiers in Python
               ============================================
```

=>All types of Literals are stored in Main memory in the created memory space. To process the values stored in main memory, as programmer, we must give distinct names to the cerated memory space. So that distinct names makes us to identify the values and hence they are called Identifiers.

=>Identifier values are changing / Varying during the program execution ands hence Identifier are called Variables.

=>In Python all types of Literals / Values are stored in Main Memory in the form Variables / Identifiers and all types of Variables / Identifiers are called objects.

=>Def. of Variable:-

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

==
Data Types in Python
==

=>The purpose of data types is that To allocate sufficient amount of memory space in main memory of the computer.
=>In Python Programming, we have 14 data types and they are classfied into 6 types.

I. Fundamental Category Data Types
    1) int
    2) float
    3) bool
    4) complex
II. Sequence Category Data Types
    1) str
    2) bytes
    3) bytearray
    4) range
III. List Category Data Types(Collection data types)
    1) list
    2) tuple
IV. Set Category Data Types(Collection data types)
    1) set
    2) frozenset

V) Dict Category Data Types(Collection data types)

    1) dict
Vi) NoneType Category Data Type
    1) None

==
I. Fundamental Category Data Types
==

=>The main purpose of Fundamental Category Data Types is that "To store Single Value".
=> In Python Programming, we have 4 types in Fundamental Category. They are

1) int
2) float
3) bool
4) complex

==
1) int
==

=>'int' is one of the pre-defined class and treated as Fundamental Data Type.
=>The purpose of 'int' is that "To store Integral / whole numbers / Integer data (
data without decimal values)".

=>Examples:

Python Statements   Outputs
>>> a=19
>>> print(a, type(a))------------------19 <class 'int'>
>>> a=999
>>> print(a,type(a), id(a))-----------999 <class 'int'> 1802394655120

=>with 'int' data, we can also store Different Types of Values Like Decimal Numbers, Binary Numbers, Octal and Hexa Decimal Numbers.
=>In Python Programming, We have 4 Number Systems and whose values can be stored by using int data type. They are

----

a) Deciaml Number System (default):

=>The Digits in Decimal Number are: 0 1 2 3 4 5 6 7 8 9
=>The total Number of digits= 10
=>The Base of Decimal Number System is 10

----

b) Binary Number System :

=>The Digits in Binary Number System are: 0 1
=>The total Number of digits= 2
=>The Base of Binary Number System is 2

----

c) Octal Number System :

=>The Digits in Octal Number System are: 0 1 2   3   4 5   6   7
=>The total Number of digits= 8
=>The Base of Octal Number System is 8

----

d) Hexa Deciaml Number System :

=>The Digits in Hexa Decimal Number System are: 0 1 2   3   4 5   6
7 8 9

A(10) B(11) C(12) D(13) E(14) F(15)
=>The total Number of digits= 16
=>The Base of Hexa Decimal Number System is 16

=>Storing Binary Number System data in Python Programming

=>To store binary number system data in python Environment, the binary data must be preceded with either '0b' or '0B'.

=>Syntax:-   varname=0b Binary Data
(OR)
=>Syntax:-   varname=0B Binary Data

=>Even though we represent the binary data , internally, it is converted into default number system (Decimal)

Examples:

```
>>> a=0b1010
>>> print(a,type(a))-------------10 <class 'int'>
>>> a=0B1111
>>> print(a,type(a))------------15 <class 'int'>
>>> a=ob1010--------NameError: name 'ob1010' is not defined
>>> a=0b10102-----SyntaxError: invalid digit '2' in binary literal
```

=>Storing Octal Number System data in Python Programming

=>To store Octal number system data in python Environment, the octal data must be preceded with either '0o' or '0O'.

=>Syntax:- varname=0o Octal Data
(OR)
=>Syntax:- varname=0O Octal Data

=>Even though we represent the Octal data , internally, it is converted into default number system (Decimal)

Examples:

```
>>> a=0o15
>>> print(a,type(a))--------------13 <class 'int'>
>>> a=0o17
>>> print(a,type(a))------------15 <class 'int'>
>>> a=0o682--------SyntaxError: invalid digit '8' in octal literal
```

===
Base Conversions Functions
===

=>The purpose of base converstion functions is that " To convert One Base Value into another Base value".
(OR)
=>The purpose of base converstion functions is that " To convert One Number System into Another Number System value".

=>In Python Programming, we have 3 types of base conversion Functions. They are

1) bin()
2) oct()
3) hex()

----

1) bin():

=>This function is used for converting Any Base (8, 16, 10) into Binary Number System data (base 2)

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

Examples:

>>> a=15
>>> print(a,type(a))--------15 <class 'int'>
>>> b=bin(a)
>>> print(b,type(b))-------0b1111 <class 'str'>

>>> a=0o14
>>> b=bin(a)
>>> print(b,type(b))-----------0b1100 <class 'str'>

>>> a=0xAC
>>> b=bin(a)
>>> print(b,type(b))----------0b10101100 <class 'str'>
>>> a=0xF
>>> b=bin(a)
>>> print(b,type(b))-------------0b1111 <class 'str'>
===
2) oct():

=>This function is used for converting Any Base (2, 16, 10) into Octal Number System data (base 8)
=>Syntax:-   varname=oct(decimal / binary / Hexa decimal value)

>>> a=15
>>> b=oct(a)
>>> print(b,type(b))----0o17 <class 'str'>
>>> a=0b1111
>>> b=oct(a)
>>> print(b,type(b))--------0o17 <class 'str'>
>>> a=0xF
>>> b=oct(a)
>>> print(b,type(b))----0o17 <class 'str'>
>>> a=0XBEE
>>> b=oct(a)
>>> print(b,type(b))----0o5756 <class 'str'>
----
3) hex():

=>This function is used for converting Any Base (2, 8, 10) into Hexa Decimal Number System data (base 16)
=>Syntax:-   varname=hex(decimal / binary / Octal value)

Examples:

>>> a=10
>>> b=hex(a)
>>> print(b,type(b))------------0xa <class 'str'>
>>> a=8
>>> b=hex(a)
>>> print(b,type(b))-----------0x8 <class 'str'>
>>> a=9
>>> b=hex(a)
>>> print(b,type(b))--------0x9 <class 'str'>
>>> a=0b1111
>>> b=hex(a)

>>> print(b,type(b))----------0xf <class 'str'>
>>> a=0o17
>>> b=hex(a)
>>> print(b,type(b))---------0xf <class 'str'>
===========================X==============

===
2) float
===

=>'float' is one of the pre-defined class and it is treated as fundamental data type.
=>The purpose of 'float' data type is that " To Store Floting Point / Real Constant
values ".
=>float data type allows us to store only decimal number System values but not support to store Binary , Octal and Hexa Decimal Number System Values.
=>float data type also supports to store "scientific Notation" whose general format is "Mantisa e Exponent".
=>"Mantisa e Exponent" can be converted into general float point values as
"Mantisa x 10 to the power of Exponent"

Examples:

>>> a=34.99
>>> print(a,type(a))--------34.99 <class 'float'>
>>> a=0.009
>>> print(a,type(a))--------0.009 <class 'float'>
>>> a=22/7
>>> print(a,type(a))---------3.142857142857143 <class 'float'>
> a=0b1111.0b1010---------SyntaxError: invalid decimal literal
>>> a=0xF.0b1010--------SyntaxError: invalid decimal literal
>>> a=0o17.0o12---------SyntaxError: invalid decimal literal

>>> a=3e2
>>> print(a,type(a))---------300.0 <class 'float'>
>>> a=43e-2
>>> print(a,type(a))--------0.43 <class 'float'>
>>> a=0.00000000000000000000000000000000000000000004
>>> print(a,type(a))------4e-44 <class 'float'>
=======================X==================================

===
3) bool
===

=>'bool' is one of the pre-defined class and treated as Fundamental data Type.
=>The purpose of bool data type is that "To store True False Values (Logical Values). "
=>Internally the value True is considered as 1
=>Internally the value False is considered as 0

=>Examples:

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

```
>>> print(b,type(b))---------False <class 'bool'>
---------------------------
Special Examples:
------------------------------
>>> a=True
>>> b=True
>>> print(a+b)-------------2
>>> print(True+False)---------1
>>> print(True*False)----------0
>>> print(False+0b1111)---------15
>>> print(2*False+True)----------1
--------------------------------------------------------------------------


               =======================================
                              4. complex
              =======================================
=>'complex' is one of the pre-defined class and treated as Fundamental
data type.
=>The purpose of complex data type is that "to store complex values ".
=>The General Format of complex value is   shown bellow\
                 a+bj (or)    a-bj
           here  'a' is called  Real Part
           here 'b' is called Imaginary Part
           and 'j' is called Sqrt(-1)
=>The extract the real part we use a pre-defined attribute called "real"
present complex object
=>The extract the imgainary part we use a pre-defined attribute called
"imag" present complex object
=>Syntax:-    complexobj.real ---->Gives real part
                complexobj.imag--->Gives imaginary part
=>Internally real and imgaginary parts are treated as float.
--------------------------------------------------------------------------
-----------
Examples:
-----------------
>>> a=2+3j
>>> print(a,type(a))------------(2+3j) <class 'complex'>
>>> b=2.3+4.6j
>>> print(b,type(b))------------(2.3+4.6j) <class 'complex'>
>>> c=2-3j
>>> print(c,type(c))------------(2-3j) <class 'complex'>
>>> d=-2.5-3.5j
>>> print(d,type(d))----------(-2.5-3.5j) <class 'complex'>
>>> e=23+4.5j
>>> print(e,type(e))-----------(23+4.5j) <class 'complex'>
>>> x=2+3i---------SyntaxError: invalid decimal literal
>>> a=2+j3-------NameError: name 'j3' is not defined
----------------------------------------------
>> a=2+3j
>>> print(a,type(a))---------(2+3j) <class 'complex'>
>>> a.real---------2.0
>>> a.imag-----3.0
>>> b=2.3+4.5j
>>> print(b.real)--------2.3
>>> print(b.imag)-----4.5
>>> print((2+3.4j).real)---------2.0
>>> print((2+3.4j).imag)--------3.4
```

===
II. Sequence Category Data Types
===

=>The main purpose of Sequence Category Data Types is that "To store Sequence of Value".
=>We have 4 data types in Sequence Category. They are
1) str
2) bytes
3) bytearray
4) range

====================================
1) str
====================================

=>'str' is one of the pre-defined class name and treated as sequence data type
=>'str' data type is used for storing Sequence of Character(s) "
=>We can store two types of String data. They are
1) Single Line Sring data
2) Multi Line String Data

1) Single Line Sring data:
=======================

=> Single Line Sring data must be enclosed within either single Quotes or double Quotes.
=>Syntax:- varname=" str data"
(or)
varname=' str data '

Examples:

```
>>> s="PYTHON"
>>> print(s,type(s))----------PYTHON <class 'str'>
>>> s='PYTHON'
>>> print(s,type(s))----------PYTHON <class 'str'>
>>> s="P"
>>> print(s,type(s))---------P <class 'str'>
>>> s='P'
>>> print(s,type(s))--------P <class 'str'>
>>> s="Python is an oop lang"
>>> print(s,type(s))----Python is an oop lang <class 'str'>
>>> s='Python is an oop lang'
>>> print(s,type(s))---------Python is an oop lang <class 'str'>
```

=>But with single and double quotes we can't organize / store Multi Line String data.
----

2) Multi Line String Data:

=> Multi Line String Data must be enclosed within either Tripple single Quotes or tripple double Quotes.

```
=>Syntax:-      varname=" " " str data1
                                str data 2
                                -------------
                                ------------- " " "

                       (or)
```

```
                    varname=' ' ' str data1
                                   str data 2
                                   -------------
                                   ------------- ' ' '
```

Examples:

```
>>> addr1="""Guido van Rossum
... HNo:3-4-14 read sea side
... Python software Foundation
... Nether Lands 500001123 """
>>> print(addr1,type(addr1))
                       Guido van Rossum
                       HNo:3-4-14 read sea side
                       Python software Foundation
                       Nether Lands 500001123  <class 'str'>
>>> addr2='''James Gosling
... FNo3-6, Hill side
... Sun Micro System Inc
... USA 45678892 '''
>>> print(addr2,type(addr2))
                       James Gosling
                       FNo3-6, Hill side
                       Sun Micro System Inc
                       USA 45678892  <class 'str'>
```

```
>>> s="""python"""
>>> print(s,type(s))---------python <class 'str'>
>>>s='''A'''
>>> print(s,type(s))---------A <class 'str'>
```

=>Hence with Tripple double quotes / single quotes, we can store both single and multi line string data.

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

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

Operations on str data

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

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

1. Indexing
2. Slicing

----

1. Indexing:

=>The process of obtaining a single value / character from a given string object is called Indexing.

=>Syntax:-   strobj [Index]

=>here index can be either +ve and -ve.

=>if the index is valid then we get Character / value from that Index

=>if the index is invalid then we get IndexError.

Examples:

```
>>> s="PYTHON"
>>> print(s[3])----------H
>>> print(s[-2])--------O
>>> print(s[-4])--------T
```

```
>>> print(s[-2])--------O
>>> print(s[2])--------T
>>> print(s[12])--------IndexError: string index out of range
>>> print(s[-12])----IndexError: string index out of range
```
----
2. Slicing:

=>The processing obtaining range of characters / sub string from Given String is called Slicing.

Syntax1:   strobj[Begin Index : End Index]

=>This Syntax obtaining Characters from Begin Index to End Index-1 provided
    Begin Index < End Index otherwise we never any output.

Examples:

```
>>> s="PYTHON"
>>> print( s[1:4] )---------------YTH
>>> print( s[2:5] )--------------THO
>>> print( s[0:6] )---------------PYTHON
>>> print( s[4:6] )------ON
>>> print( s[4:2] )-------  empty / no output
>>> s[4:2]---------- ' '
>>> s="PYTHON"
>>> print(s[-6:-4] )----------PY
>>> print(s[-5:-1] )----YTHO
>>> print(s[-1:-4] )----  empty / no output
```

Syntax2:- strobj[Begin Index :   ]
=>This syntax gives range of characters from Begin Index to upto last character.
=>In this index we get Characters from begin index to end index where end index=len(strdata)-1

Examples:

```
>>> s="PYTHON"
>>> print(s[2:])------------THON
>>> print(s[4:])---------ON
>>> print(s[3:])-------HON
>>> print(s[0:])--------PYTHON
>>> print(s[-6:])-------PYTHON
>>> print(s[-4:])--------THON
>>> print(s[-5:])-------YTHON
>>> print(s[-2:])------ON
```

Syntax3:   strobj[: endIndex]
=>In this Syntax, we don't have Begin Index.
=>Here The value of Begin Index is by default Initial Index (0)
=>Syntax Syntax gives from Begin Index to end Index-1

Examples:

>>> s="PYTHON"
>>> print(s[:4])-----------------PYTH
>>> print(s[:3])-----------PYT
>>> print(s[:-4])---------PY
>>> print(s[:-5])--------P
>>> print(s[:6])---------PYTHON
>>> print(s[:-1])-----PYTHO

Syntax4: strobj[:]
=>In this Syntax we don't have Begin Index and End Index.
=>If we don't specify Begin Index then PVM takes Intial Index as Begin Index
=>If we don't specify End Index then PVM takes len(strdata)-1 as End Index
----
Examples:

>>> s="PYTHON"
>>> print(s[:])
PYTHON
>>>

Syntax-5:
----
strobj [Begin Index : End Index : Step]
=>This syntax gives range of characters vfrom begin Index to End Index-1 by maintaining Interval of Values with Step.

Rule1:- Here Begin Index , end Index and step Values can be either +ve or -ve

Rule2: If the VALUE OF STEP IS +VE then we consider / get the elements from
Begin Index to End Index-1 in forward direction provided Begin Index<End Index.

Rule3: If the VALUE OF STEP IS -VE then we consider / get the elements from
Begin Index to End Index+1 in backward direction provided Begin Index>End Index.
Rule 4:when we get the elements in forward direction and if the end index is 0
then we never get any output.

Rule 5:when we get the elements in backward direction and if the end index is -1 then we never get any output.

Examples:

>>> s="PYTHON"
>>> print(s[0:6:2])----------PTO
>>> print(s[2:5:1])---------THO
>>> print(s[2:5:2])----------TO
>>> print(s[:6:2])---------PTO

```
>>> print(s[-6:-2:1] )--------PYTH
>>> print(s[-6:-2:2] )--------PT
>>> print(s[::3] )-------PH
>>> s="PYTHON"
>>> print(s[4:1:-1])-------OHT
>>> s="PYTHON"
>>> print(s[5:1:-1])-------NOHT
>>> print(s[4:1:-2])--------OT
>>> print(s[-2:-5:-1])---------OHT
>>> print(s[-1:-5:-1])-------NOHT
>>> print(s[::-2])----------NHY
>>> print(s[::-1])----------NOHTYP
>>> print(s[:0:1])--------empty

>>> print(s[:-1:-1])------empty
>>> s="PYTHON"
>>> print(s[::-3])-----------NT
>>> print(s[::3])--------PH
>>> print(s[5::-1])--------NOHTYP
>>> s="KVR"
>>> s[::-1]----------'RVK'
============================================
>>> s="LIRIL"
>>> s[::-1]--------------'LIRIL'
>>> s="MADAM"
>>> s[::-1]-------------'MADAM'
>>> s="MALAYALAM"
>>> s[::-1]---------'MALAYALAM'
============================X==============================
```

==

Type Casting Techniques in Python

==

=>The process of Converting one type of Possible value into another type of value is known as Type Casting.
=>In Python Programming , we have 5 type casting techniques. They are

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

==

1) int():

==

=>This function is used for converting one Possible value into int type Value.
=>Syntax:- varname= int (float / bool / complex / str)

```
-------------------
Examples:    float---> int--->Possible
-------------------
>>> a=10.23
>>> print(a,type(a))----------10.23 <class 'float'>
>>> b=int(a)   # float---> int--->Possible
>>> print(b, type(b))---------10 <class 'int'>
```

```
--------------------------
Examples:     bool---->int-->Possible
------------------
>>> a=True
>>> print(a,type(a))-------------True <class 'bool'>
>>> b=int(a)
>>> print(b, type(b))---------1 <class 'int'>
>>> a=False
>>> print(a,type(a))---------False <class 'bool'>
>>> b=int(a)
>>> print(b, type(b))---------0 <class 'int'>
--------------------------------------------------------
Examples: complex---->int-->NOT POSSIBLE
--------------------------------------------------------
>>> a=2+3j
>>> print(a,type(a))-------(2+3j) <class 'complex'>
>>> b=int(a)----TypeError: int() it should not  'complex'
-------------------------------------------------------------------------
------------
Examples:
---------------------
>>> a="123"
>>> print(a,type(a))------------123 <class 'str'>
>>> b=int(a)
>>> print(b, type(b))---------123 <class 'int'>
>>> a="12.34"
>>> print(a,type(a))-----------12.34 <class 'str'>
>>> b=int(a)------------ValueError: invalid literal for int() with base
10: '12.34'
>>> a="2+3j"
>>> print(a,type(a))----------2+3j <class 'str'>
>>> b=int(a)-----ValueError: invalid literal for int() with base 10:
'2+3j'
>>> a="True"
>>> print(a,type(a))----------True <class 'str'>
>>> b=int(a)--------ValueError: invalid literal for int() with base 10:
'True'
>>> a="KVR"
>>> print(a,type(a))---------KVR <class 'str'>
>>> b=int(a)----------ValueError: invalid literal for int() with base 10:
'KVR'
```

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

2) float()

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

=>This function is used for converting one Possible value into float type Value.

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

```
-----------------------------------------------
Examples:   int---------->float--->Possible
-----------------------------------------------
>>> a=10
>>> print(a,type(a))----------10 <class 'int'>
>>> b=float(a)
>>> print(b, type(b))----------10.0 <class 'float'>
--------------------------------------------------
Examples:-  bool---->float--->Possible
```

```
----------------------------------------------------
>>> a=True
>>> print(a,type(a))-----------True <class 'bool'>
>>> b=float(a)
>>> print(b, type(b))----------1.0 <class 'float'>
-----------------------------------------------------------------
Examples:   complex---->float---> Not Possible
------------------------------------------------------------------
>>> a=2+3.5j
>>> print(a,type(a))----------(2+3.5j) <class 'complex'>
>>> b=float(a)------TypeError: float() argument must be a string or a
real number, not 'complex'
---------------------------------------------------------------------
Examples:
------------------------
>>> a="100"
>>> print(a,type(a))----------100 <class 'str'>
>>> b=float(a)
>>> print(b, type(b))----------100.0 <class 'float'>
>>> a="12.34"
>>> print(a,type(a))--------12.34 <class 'str'>
>>> b=float(a)
>>> print(b, type(b))-----12.34 <class 'float'>
------------------------------
>>> a="True"
>>> print(a,type(a))---------True <class 'str'>
>>> b=float(a)-----ValueError: could not convert string to float: 'True'
>>> a="-2-3.4j"
>>> print(a,type(a))-------2-3.4j <class 'str'>
>>> b=float(a)----ValueError: could not convert string to float: '-2-
3.4j'
>>> a="python"
>>> print(a,type(a))---------python <class 'str'>
>>> b=float(a)---ValueError: could not convert string to float: 'python'
```

```
=======================================
bool()
========================================
```

=>This function is used for converting one Possible value into bool type Value.

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

=>ALL NON-ZERO values are TRUE

=>ALL ZEROs values are FALSE

```
-----------------------------------------------------------
Example:   int---bool---Possible
----------------
>>> a=1003
>>> print(a,type(a))----------1003 <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'>
-----------------------------------------
Examples:  float--->bool-->Posssible
----------------------------------------
>>> a=12.34
>>> print(a,type(a))------------12.34 <class 'float'>
>>> b=bool(a)
>>> print(b, type(b))----------True <class 'bool'>
>>> a=0.0
>>> print(a,type(a))-------0.0 <class 'float'>
>>> b=bool(a)
>>> print(b, type(b))-------False <class 'bool'>
>>> a=0.0000000000000000000000000000000000000001
>>> print(a,type(a))----------1e-40 <class 'float'>
>>> b=bool(a)
>>> print(b, type(b))--------True <class 'bool'>
---------------------------------------------
Examples:  complex--->bool-->Posssible
----------------------------------------
>>> 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'>
------------------------------------------------------------------------
Examples:
--------------------
>>> a="1234"
>>> print(a,type(a))
1234 <class 'str'>
>>> b=bool(a)
>>> print(b, type(b))
True <class 'bool'>
>>> a="12.34"
>>> print(a,type(a))
12.34 <class 'str'>
>>> b=bool(a)
>>> print(b, type(b))
True <class 'bool'>
>>> a="0.0"
>>> print(a,type(a))
0.0 <class 'str'>
>>> b=bool(a)
>>> print(b, type(b))
True <class 'bool'>
-----------------------------------------------------
```

==

complex()

==

=>This function is used for converting one Possible value into complex type Value.

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

```
-------------------------------------------
>>> a=10
>>> print(a,type(a))-----------10 <class 'int'>
>>> b=complex(a)
>>> print(b, type(b))----------(10+0j) <class 'complex'>
---------------------------------------------
Examples---> float---->complex--Possible
---------------------------------------------
>>> a=-2.3
>>> print(a,type(a))-----------2.3 <class 'float'>
>>> b=complex(a)
>>> print(b, type(b))----------(-2.3+0j) <class 'complex'>
--------------------------------------------------------------------------
--
Examples:----- bool---->complex---Possible
-------------------------------------------------------
>>> a=True
>>> print(a,type(a))--------------True <class 'bool'>
>>> b=complex(a)
>>> print(b, type(b))-----------(1+0j) <class 'complex'>
>>> a=False
>>> print(a,type(a))----------False <class 'bool'>
>>> b=complex(a)
>>> print(b, type(b))-----------0j <class 'complex'>
--------------------------------------------------------------------------
-
Examples:--
--------------------------------------------------------------------------
>>> a="12"
>>> print(a,type(a))----------12 <class 'str'>
>>> b=complex(a)
>>> print(b, type(b))----------(12+0j) <class 'complex'>
>>> a="12.34"
>>> print(a,type(a))--------12.34 <class 'str'>
>>> b=complex(a)
>>> print(b, type(b))----------(12.34+0j) <class 'complex'>
>>> a="True"
>>> print(a,type(a))---------True <class 'str'>
>>> b=complex(a)---------ValueError: complex() arg is a malformed string
>>> a="python"
>>> print(a,type(a))---------python <class 'str'>
>>> b=complex(a)-------ValueError: complex() arg is a malformed string
--------------------------------------------------------------------------
----------------------------
```

==

str ()

==

=>This function is used for converting All Types value into str type Value.

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

Examples:

```
------------------
>>> a=100
>>> print(a,type(a))------------------100 <class 'int'>
>>> b=str(a)
>>> b------------- '100'
```

```
>>> a=12.34
>>> print(a,type(a))-----------12.34 <class 'float'>
>>> b=str(a)
>>> b--------------'12.34'
>>> a=True
>>> print(a,type(a))-----------True <class 'bool'>
>>> b=str(a)
>>> b------------  'True'
>>> a=2+3.6j
>>> print(a,type(a))------------(2+3.6j) <class 'complex'>
>>> b=str(a)
>>> b----------------'(2+3.6j)'
--------------------------------------------------
```

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

Mutability and Immutability

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

Mutability:

=>A mutable object is one, which allows us to do the changes / updations in the same address.

Example:-   list , set, dict....etc

Immutable:

=>An immutable object is one, which never allows us to do the changes / updations in the same address.

=>Changes can happen in same variable / object but placed in new memory address.

Examples: int , float   , bool,   complex ...etc

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

==

2) bytes

==

=>'bytes' is a pre-defined class and treated as Sequence data type.

=>The purpose of bytes data type is that " To store Squence of Positive Numerical Integer in the range(0,256). ie it stores 0 to 255 .

=>To represent the elements of bytes data type, we don't have any symbolic notaion but we can convert other data type elements into bytes data type values by using   bytes().

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

=>The object of bytes belongs to immutable bcoz 'bytes' object does not support item assignment.

=>An object of bytes allows to place / organize both unique and duplicate values.

Examples:

```
>>> lst=[10,23,45,56,256]
>>> print(lst, type(lst))-------------------[10, 23, 45, 56, 256] <class 'list'>
>>> b=bytes(lst)-------------ValueError: bytes must be in range(0, 256)
>>> lst=[10,0,-23,45,56,255]
>>> print(lst, type(lst))-----[10, 0, -23, 45, 56, 255] <class 'list'>
>>> b=bytes(lst)-------------ValueError: bytes must be in range(0, 256)
```

```
>>> lst=[10,0,23,45,56,255]
>>> print(lst, type(lst))-----------[10, 0, 23, 45, 56, 255] <class
'list'>
>>> b=bytes(lst)
>>> print(b, type(b))-----b'\n\x00\x17-8\xff' <class 'bytes'>
>>> for val in b:
...     print(val)
                             ...
                             10
                             0
                             23
                             45
                             56
                             255
>>> print( b[0])---------------------------  10
>>> print( b[3])--------------- 45
>>> print( b[-1])-----------255
>>> print( b[-4])------23
>>> print(b[2:5])------------   b'\x17-8'
>>> for val in b[2:5]:
...         print(val)
...
                             23
                             45
                             56
>>> for val in b[::-1]:
...     print(val)
                             ...
                             255
                             56
                             45
                             23
                             0
                             10
>>> tp=(10,23,45,67,"KVR")------------>>> print(tp,type(tp))
(10, 23, 45, 67, 'KVR') <class 'tuple'>
>>> b1=bytes(tp)-----------TypeError: 'str' object cannot be interpreted
as an integer
```

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

==

bytearray

==

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

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

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

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

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

=>An object of bytearray maintains Insertion Order.

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

Examples:

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

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

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

===

range

===

=>'range' is one of the pre-defined class name and terated as sequence data type.

=>The purpose of range data type is that "To Store sequence of Integer values with equal Interval."

=>The object of range data type is immutable.
=>On The the object range data type we can perform Indexing and slicing operations.
=>The range data type contains 3 syntaxes. They are

```
Syntax1:              varname=range(Value)
-------------
```

=>This syntax give 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 v in r:
...     print(v)
                       ...
                       0
                       1
                       2
                       3
                       4
                       5
>>> for v in range(11):
...     print(v)
                       ...
                       0
                       1
                       2
                       3
                       4
                       5
                       6
                       7
                       8
                       9
                       10
-------------------------------------------------------------------------------------------------
Syntax2:              varname=range(start,stop)
-------------
```

=>This syntax gives range of values from start to stop-1

```
>>> r=range(100,106)
>>> print(r,type(r))
range(100, 106) <class 'range'>
>>> for val in r:
...     print(val)
                       ...
                       100
                       101
                       102
                       103
                       104
                       105
>>> for val in range(90,101):
...     print(val)
                             ...
                             90
```

```
                              91
                              92
                              93
                              94
                              95
                              96
                              97
                              98
                              99
                              100
```

Syntax3:   varname=range(Start , Stop, step)

=>This syntax give range of values from start to stop-1 with equal interval of step value.

Examples:

```
>>> r=range(10,21,2)
>>> for val in r:
...     print(val)
                        ...
                        10
                        12
                        14
                        16
                        18
                        20
>>> for v in range(100,151,10):
...     print(v)
                  ...
                  100
                  110
                  120
                  130
                  140
                  150
>>> r=range(100,151,10)
>>> print(r[0])------------100
>>> print(r[3])------------130
>>> print(r[-1])----------150
>>> print(r[-3])-----------130
>>> print(r[2:5])------------range(120, 150, 10)
>>> for val in r[2:5]:
...     print(val)
                        ...
                        120
                        130
                        140
>>> r[0]=123---------TypeError: 'range' object does not support item assignment
```

NOTE:- In the above syntaxes, start, stop, step values must be Integers and they should not be float.

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

Examples:

Q1)Generate   sequence of   0   1   2   3 4 5

```
>>> for val in range(6):
...     print(val)
                     ...
                     0
                     1
                     2
                     3
                     4
                     5
```

Q2)Generate   sequence of 1 2   3 4 5   6   7   8 9   10

```
>>> for val in range(1,11):
...     print(val)
...
1
2
3
4
5
6
7
8
9
10
```

Q2) Generate   sequence of   10 15 20 25 30 35 40 45 50------

```
>>> for val in range(10,51,5):
...     print(val)
...
10
15
20
25
30
35
40
45
50
```

Q2) Generate   sequence of   1000   1010 1020 1030 1040 1050----

```
>>> for val in range(1000,1051,10):
...     print(val)
...
1000
1010
1020
1030
1040
1050
```


==
III. List Category Data Types(Collection data types)
==

=>The puurpose of List Category Data Types(Collection data types) is that "To store multiple values either of same type or different type or both the types with unique and duplicates."
=>we have two data types in List Category. They are
a) list
b) tuple
---x---------------------

List:

Purpose of list
organization of list
list indexing and slicing
Pre-defined functions in List:
append()
insert()
pop(index)
pop()
remove()
copy()-----deep copy and shallow copy
index()
count()
extend()
sort()
reverse()
update()
=>inner / nested list
=>Operations inner list
=>Pre-defined functions in inner list

==
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 unique and duplicates."
=>The elements of list must be written within square brackets [ ] and elements of list separated by comma.
=>An object of list maintains Insertion Oder.(In Whichever order we insert the data, in the same order elements will be displayed)
=>An object of list belongs to mutable.
=>On the object of list we can perform both Indexing and slicing operations.
=>We can convert other type value into list type value by using list()
=>We can create two types of list object. They are
a) empty list
b) non-empty list

a) empty list:

=>An empty list is one which does not contain any elements and whose length is 0
Syntax:-   varname=[]
                 (or)
            varname=list()
----
            b) non-empty list
b) non-empty list:

=>A non-empty list is one which contains elements and whose length is > 0
Syntax:-   varname=[val1,val2,.....val;-n]

Examples:

>>> l1=[10,20,30,40,10,20]
>>> print(l1,type(l1))--------[10, 20, 30, 40, 10, 20] <class 'list'>
>>> l2=[10,"Rossum",23.45,True,"Python"]
>>> print(l2,type(l2))----[10, 'Rossum', 23.45, True, 'Python'] <class 'list'>
>>> len(l1)--------6
>>> len(l2)-------5
>>> print(l1,type(l1),id(l1))-------[10, 20, 30, 40, 10, 20] <class 'list'> 1877217718528
>>> l1[2]=300
>>> print(l1,type(l1),id(l1))---[10, 20, 300, 40, 10, 20] <class 'list'> 1877217718528
>>> print(l1[2])-------300
>>> print(l1[2:5])-------[300, 40, 10]
>>> print(l1[::2])--------[10, 300, 10]
>>> print(l1[::-1])-------[20, 10, 40, 300, 20, 10]
>>> print(l1,type(l1),id(l1))---[10, 20, 300, 40, 10, 20] <class 'list'> 1877217718528
>>> l3=[]
>>> print(l3,type(l3))--------- [] <class 'list'>
>>> len(l3)------------0
>>> l4=list()
>>> print(l4,type(l4))--------------[] <class 'list'>
>>> len(l4)--------------0

==
Pre-defined functions in List:
==
=>We know that on the object of list we can both indexing and slicing operations. =>Along with Indexing and Slicing Operations, we can also perform some additional Operation by using pre-defined function in list. They are
----
1) append():

=>This function is used for adding the elements to list object at end of existing elements of list.
=>Syntax:-   listobj.append(Element)

Examples:

>>> l1=[]
>>> print(l1,type(l1),id(l1))---------[] <class 'list'> 1877217672192
>>> l1.append(10)
>>> print(l1,type(l1),id(l1))-------[10] <class 'list'> 1877217672192
>>> l1.append("Rossum")
>>> l1.append(23.45)
>>> print(l1,type(l1),id(l1))------[10, 'Rossum', 23.45] <class 'list'>
1877217672192
>>> l2=["Apple","Banana","Kiwi"]
>>> print(l2,type(l2),id(l2))--['Apple', 'Banana', 'Kiwi'] <class 'list'>
1877217968512
>>> l2.append("Sberry")
>>> print(l2,type(l2),id(l2))--['Apple', 'Banana', 'Kiwi', 'Sberry']
<class 'list'>
1877217968512
----

2) insert()

=>This Function is used for inserting an element at a specified valid exiting index / Position .
=>Syntax:- listobj.insert(Index,Element)
=>here 'index' can be either +ve or -ve.

Examples:

>>> l2=["Apple","Banana","Kiwi"]
>>> print(l2,type(l2),id(l2))---['Apple', 'Banana', 'Kiwi'] <class
'list'> 1877217969472
>>> l2.insert(1,"Guava")
>>> print(l2,type(l2),id(l2))---['Apple', 'Guava', 'Banana', 'Kiwi']
<class 'list'>
1877217969472
>>> l2.insert(1,"Wmellon")
>>> print(l2,type(l2),id(l2))---['Apple', 'Wmellon', 'Guava', 'Banana',
'Kiwi'] <class
'list'> 1877217969472
----

3)remove()

=>This function is used for removing First occurence of the specified element.
=>If the specified element is not present in list object then we get ValueError.
=>Syntax:- listobj.remove(element)

Examples:

>>> l2=["Apple","Banana","Kiwi","Guava"]
>>> print(l2,type(l2),id(l2))------['Apple', 'Banana', 'Kiwi', 'Guava']
<class 'list'> 1877217968512
>>> l2.remove("Banana")
>>> print(l2,type(l2),id(l2))----['Apple', 'Kiwi', 'Guava'] <class
'list'> 1877217968512
>>> l2.remove("kiwi")----ValueError: list.remove(x): x not in list
>>> l2.remove(100)--------ValueError: list.remove(x): x not in list

```
>>> l1=[10,20,30,10,20,40]
>>> print(l1,type(l1),id(l1))---[10, 20, 30, 10, 20, 40] <class 'list'>
1877217969472
>>> l1.remove(10)
>>> print(l1,type(l1),id(l1))---[20, 30, 10, 20, 40] <class 'list'>
1877217969472
>>> l1.remove(20)
>>> print(l1,type(l1),id(l1))--------[30, 10, 20, 40] <class 'list'>
1877217969472
--------------------------------------------------------------------------
------------
4) pop(index)
-----------------------
=>This Function is used for removing the element based on Valid Existing
Index
=>Syntax:-   listobj.pop(index)
=>here 'index' can be either +ve or -ve
=>If the value of index in invalid then we get IndexError.
Examples:
-----------------
>>> l2=["Apple","Banana","Kiwi","Guava","Banana"]
>>> print(l2)------['Apple', 'Banana', 'Kiwi', 'Guava', 'Banana']
>>> l2.pop(4)------'Banana'
>>> print(l2)------------['Apple', 'Banana', 'Kiwi', 'Guava']
>>> l2.pop(1)----------'Banana'
>>> print(l2)--------['Apple', 'Kiwi', 'Guava']
>>> l2.pop(11)-----------IndexError: pop index out of range
>>> [].pop(10)---------IndexError: pop from empty list
>>> print(l2)---------['Apple', 'Kiwi', 'Guava']
>>> l2.pop(-1)-----------'Guava'
>>> print(l2)---------------['Apple', 'Kiwi']
>>> l2.pop(-2)---------'Apple'
>>> print(l2)---------['Kiwi']
>>> list().pop(1)----------IndexError: pop from empty list
--------------------------------------------------------------------------
-----------------------
5) pop()
----------------
=>This Function  is used for removing last +ve indexed value.
=>Syntax:-    listobj.pop()
----------------
Examples:
-------------------
>>> l2=["Apple","Banana","Kiwi","Guava","Banana"]
>>> print(l2)-------['Apple', 'Banana', 'Kiwi', 'Guava', 'Banana']
>>> l2.pop()-------'Banana'
>>> print(l2)-------['Apple', 'Banana', 'Kiwi', 'Guava']
>>> l2.insert(1,"Sberry")
>>> print(l2)-------['Apple', 'Sberry', 'Banana', 'Kiwi', 'Guava']
>>> l2.pop()-------'Guava'
>>> print(l2)---------['Apple', 'Sberry', 'Banana', 'Kiwi']
>>> [].pop()----------IndexError: pop from empty list
>>> list().pop()----------IndexError: pop from empty list
--------------------------------------------------------------------------
-------------
6) clear():
-------------------
=>This Function is  used for removing all the elements of list object.
```

```
=>Syntax:     listobj.clear()
=>When we call clear() upon empty list object then we never get any error

Examples:
------------------
>>> l2=["Apple","Banana","Kiwi","Guava","Banana"]
>>> print(l2)-----------['Apple', 'Banana', 'Kiwi', 'Guava', 'Banana']
>>> len(l2)-----5
>>> l2.clear()
>>> print(l2)-------[]
>>> len(l2)---------0
>>> [].clear()-----------empty
>>> list().clear()---------empty
--------------------------------------------------------------------------
-------------
Note:-with   'del',  we can also delete the elements of list either based
on indexing
and slicing and entire object
--------------------------------------------------------------------------
-------------------------
Examples:
--------------------------------------------------------------------------
-------------------------
>>>l2=["Apple","Banana","Kiwi","Guava","Banana"]
>>> del l2[1]
>>> print(l2)---------['Apple', 'Kiwi', 'Guava', 'Banana']
>>> del l2[1:3]
>>> print(l2)----['Apple', 'Banana']
>>> l2=["Apple","Banana","Kiwi","Guava","Banana"]
>>> print(l2)-----['Apple', 'Banana', 'Kiwi', 'Guava', 'Banana']
>>> del l2[::2]
>>> print(l2)--------['Banana', 'Guava']
>>> del l2
>>> print(l2)--------NameError: name 'l2' is not defined. Did you mean:
'l1'?

>>> l2=["Apple","Banana","Kiwi","Guava","Banana"]
>>> print(l2)----['Apple', 'Banana', 'Kiwi', 'Guava', 'Banana']
>>> del l2[::]
>>> print(l2)------------[]
--------------------------------------------------------------------------
-------
7) copy():
------------------
=>This function is used for copying the content from one list object to
another list object( Implements shallow copy).

=>Syntax:-     listobj2=listobj1.copy()
-----------------
Examples:
----------------
>>> l1=[10,"Rossum"]
>>> print(l1,id(l1))--------------[10, 'Rossum'] 3180890173696
>>> l2=l1.copy()
>>> print(l2,id(l2))--------------[10, 'Rossum'] 3180890127360
>>> l1.append("Python")
>>> print(l1,id(l1))---------------[10, 'Rossum', 'Python'] 3180890173696
>>> print(l2,id(l2))-------[10, 'Rossum'] 3180890127360
```

```
>>> l2.insert(1,"Java")
>>> print(l1,id(l1))-------------[10, 'Rossum', 'Python'] 3180890173696
>>> print(l2,id(l2))-----------[10, 'Java', 'Rossum'] 3180890127360
```

Deep Copy Examples:

Examples:

```
>>> l1=[10,"Rossum"]
>>> print(l1,id(l1))-------[10, 'Rossum'] 3180890162560
>>> l2=l1  # Deep Copy
>>> print(l2,id(l2))------------[10, 'Rossum'] 3180890162560
>>> l1.append("DS")
>>> print(l1,id(l1))----------[10, 'Rossum', 'DS'] 3180890162560
>>> print(l2,id(l2))---------[10, 'Rossum', 'DS'] 3180890162560
>>> l2.insert(1,"Travis")
>>> print(l1,id(l1))---------[10, 'Travis', 'Rossum', 'DS'] 3180890162560
>>> print(l2,id(l2))----------[10, 'Travis', 'Rossum', 'DS']
3180890162560
```
==

Slicing Based Copy also comes under shallow copy

```
>>> l1=[10,20,40,50,60]
>>> l2=l1[::]  # Slicing Based Copy
>>> print(l1,id(l1))---------[10, 20, 40, 50, 60] 3180890423744
>>> print(l2,id(l2))----------[10, 20, 40, 50, 60] 3180890173696

>>> l1=[10,20,40,50,60]
>>> l2=l1[1:3]
>>> print(l1,id(l1))-----------[10, 20, 40, 50, 60] 3180890162560
>>> print(l2,id(l2))---------[20, 40] 3180890423744
```

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

8) index():

=>This Function is used for finding index of the specified element.
=>If the element does not exists then we get ValueError
=>Syntax:-   listoibj.index(element)

Examples:

```
>>> l1=[10,20,30,40,"Python","Java","DS",True]
>>> print(l1)--------[10, 20, 30, 40, 'Python', 'Java', 'DS', True]
>>> l1.index("Python")--------4
>>> l2=[10,20,10,"java",45.6]
>>> print(l2)---------[10, 20, 10, 'java', 45.6]
>>> l2.index(10)--------0
>>> l2.index(100)-------ValueError: 100 is not in list
```
----

9) count():

=>This function is used for finding number of occurences of a specified element in list object.
=>If the element does not exists then whose number of occurences is 0
=> Syntax:-   listobj.count(element)

Examples:

```
>>> l1=[10,20,20,10,10,20,30,10,30,40,50,60]
>>> print(l1)
[10, 20, 20, 10, 10, 20, 30, 10, 30, 40, 50, 60]
>>> l1.count(10)------------4
>>> l1.count(20)-----------3
>>> l1.count(30)----------2
>>> l1.count(40)----------1
>>> l1.count(400)--------0
>>> l1.count("KVR")-----------0
```
----

10) extend():

=>This function is used for extending the functionality of one list object with another list object.
=>Syntax:- listobj1.extend(listobj2)
=>Here extend() adding all the elements of listobj2 to listobj1.
Examples:

```
>>> l1=[10,"Rossum","Python"]
>>> l2=["DS","Django","Pandas"]
>>> print(l1)--------[10, 'Rossum', 'Python']
>>> print(l2)----------['DS', 'Django', 'Pandas']
>>> l1.extend(l2)
>>> print(l1)-------[10, 'Rossum', 'Python', 'DS', 'Django', 'Pandas']
>>> print(l2)--------['DS', 'Django', 'Pandas']
-----------------------------------------------------------------------------
>>> l1=[10,"Rossum","Python"]
>>> l2=["DS","Django","Pandas"]
>>> l3=["Ram","Lax","Man"]
>>> print(l1)----------------[10, 'Rossum', 'Python']
>>> print(l2)------------['DS', 'Django', 'Pandas']
>>> print(l3)--------------['Ram', 'Lax', 'Man']
>>> l1.extend(l2,l3)---TypeError: list.extend() takes exactly one argument (2 given)
>>> l1=l1+l2+l3  # using operator + we can extends many list objects contents                                                       into a single object
>>> print(l1)--[10, 'Rossum', 'Python', 'DS', 'Django', 'Pandas', 'Ram', 'Lax', 'Man']
```
--X---

11) reverse()

=>This function is used for obtaining reverse of list object content (back to front)
=>Syntax:- listobj.reverse()

Examples:

```
>>> l1=[10,20,20,10,10,20,30,10,30,40,50,60]
>>> print(l1)---------[10, 20, 20, 10, 10, 20, 30, 10, 30, 40, 50, 60]
>>> l1.reverse()
>>> print(l1)------------[60, 50, 40, 30, 10, 30, 20, 10, 10, 20, 20, 10]
```

```
>>> l1.reverse()
>>> print(l1)-----------[10, 20, 20, 10, 10, 20, 30, 10, 30, 40, 50, 60]
>>> l2=[10,"Rossum","Pytrhon",True,2+3j,23.45]
>>> print(l2)---------[10, 'Rossum', 'Pytrhon', True, (2+3j), 23.45]
>>> l2.reverse()
>>> print(l2)-----------[23.45, (2+3j), True, 'Pytrhon', 'Rossum', 10]
------------------------------------------------------------------------
------------------------------
12) sort()
---------------------
=>This function is used for sorting the similar type data of list object
either in Ascending order ( by default- reverse=False)
or in decending order ( reverse=True)

=>Syntax:-      listobj.sort(reverse=True | False )

Examples1:
--------------------
>>> l1=[10,-2,23,15,34,7,-5,0,23,56]
>>> print(l1)-----------[10, -2, 23, 15, 34, 7, -5, 0, 23, 56]
>>> l1.sort()
>>> print(l1)-------------[-5, -2, 0, 7, 10, 15, 23, 23, 34, 56]
>>> l1.reverse()
>>> print(l1)------------[56, 34, 23, 23, 15, 10, 7, 0, -2, -5]
>>> l2=["kiwi","apple","guava","sberry","banana"]
>>> print(l2)----------['kiwi', 'apple', 'guava', 'sberry', 'banana']
>>> l2.sort()
>>> print(l2)---------['apple', 'banana', 'guava', 'kiwi', 'sberry']
>>> l2.reverse()
>>> print(l2)----------['sberry', 'kiwi', 'guava', 'banana', 'apple']
-----------------------------------------------------------------
Examples2:
=--------------------
>>> l1=[10,-2,23,15,34,7,-5,0,23,56]
>>> print(l1)------------[10, -2, 23, 15, 34, 7, -5, 0, 23, 56]
>>> l1.sort(reverse=True)
>>> print(l1)------------[56, 34, 23, 23, 15, 10, 7, 0, -2, -5]
>>> l1=[10,-2,23,15,34,7,-5,0,23,56]
>>> print(l1)-----------[10, -2, 23, 15, 34, 7, -5, 0, 23, 56]
>>> l1.sort(reverse=False)
>>> print(l1)-----------[-5, -2, 0, 7, 10, 15, 23, 23, 34, 56]

>>> l2=["kiwi","apple","guava","sberry","banana"]
>>> print(l2)--------['kiwi', 'apple', 'guava', 'sberry', 'banana']
>>> l2.sort(reverse=True)
>>> print(l2)-----------['sberry', 'kiwi', 'guava', 'banana', 'apple']
>>> l2=["kiwi","apple","guava","sberry","banana"]
>>> print(l2)----['kiwi', 'apple', 'guava', 'sberry', 'banana']
>>> l2.sort()
>>> print(l2)-----------['apple', 'banana', 'guava', 'kiwi', 'sberry']
------------------------------------------------------------------------
--------------------------------

            ========================================
                 Types of Copy Processes
           ========================================
=>Copy is the process of Copying the content one object into  another
object .
```

=>In Python Programming, we have two copy Processes. They are
a) Shallow Copy
b) Deep Copy

----
a) Shallow Copy:

=>In Shallow Copy
a) Initially content of both the objects are same
b) The Memory Address of both the objects are Different
c) The Modification of both objects are Independent(modifications are not reflecting to each other)
=>In Python Programming, shallow copy is implemented by copy()
Examples:

```
>>> l1=[10,"Rossum"]
>>> print(l1,id(l1))
[10, 'Rossum'] 3180890173696
>>> l2=l1.copy()
>>> print(l2,id(l2))
[10, 'Rossum'] 3180890127360
>>> l1.append("Python")
>>> print(l1,id(l1))
[10, 'Rossum', 'Python'] 3180890173696
>>> print(l2,id(l2))
[10, 'Rossum'] 3180890127360
>>> l2.insert(1,"Java")
>>> print(l1,id(l1))
[10, 'Rossum', 'Python'] 3180890173696
>>> print(l2,id(l2))
[10, 'Java', 'Rossum'] 3180890127360
>>>
```
===
b) Deep Copy:

=>In Deep Copy
a) Initially content of both the objects are same
b) The Memory Address of both the objects are Same
c) The Modification of both objects are Dependent(modifications are reflecting to each other)
=>In Python Programming, Deep copy is implemented by Assigment Operator (=)

Syntax:- listobj2=listobj1 # deep copy

Examples:

```
>>> l1=[10,"Rossum"]
>>> print(l1,id(l1))-------[10, 'Rossum'] 3180890162560
>>> l2=l1  # Deep Copy
>>> print(l2,id(l2))------------[10, 'Rossum'] 3180890162560
>>> l1.append("DS")
>>> print(l1,id(l1))----------[10, 'Rossum', 'DS'] 3180890162560
>>> print(l2,id(l2))---------[10, 'Rossum', 'DS'] 3180890162560
>>> l2.insert(1,"Travis")
>>> print(l1,id(l1))---------[10, 'Travis', 'Rossum', 'DS'] 3180890162560
>>> print(l2,id(l2))----------[10, 'Travis', 'Rossum', 'DS']
3180890162560
```

---X---

==
Types of Copy Processes
==

=>Copy is the process of Copying the content one object into another object .
=>In Python Programming, we have two copy Processes. They are
a) Shallow Copy
b) Deep Copy

----
a) Shallow Copy:

=>In Shallow Copy
a) Initially content of both the objects are same
b) The Memory Address of both the objects are Different
c) The Modification of both objects are Independent(modifications are not reflecting to each other)
=>In Python Programming, shallow copy is implemented by copy()
Examples:

```
>>> l1=[10,"Rossum"]
>>> print(l1,id(l1))
[10, 'Rossum'] 3180890173696
>>> l2=l1.copy()
>>> print(l2,id(l2))
[10, 'Rossum'] 3180890127360
>>> l1.append("Python")
>>> print(l1,id(l1))
[10, 'Rossum', 'Python'] 3180890173696
>>> print(l2,id(l2))
[10, 'Rossum'] 3180890127360
>>> l2.insert(1,"Java")
>>> print(l1,id(l1))
[10, 'Rossum', 'Python'] 3180890173696
>>> print(l2,id(l2))
[10, 'Java', 'Rossum'] 3180890127360
>>>
```
===
b) Deep Copy:

=>In Deep Copy
a) Initially content of both the objects are same
b) The Memory Address of both the objects are Same
c) The Modification of both objects are Dependent(modifications are reflecting to each other)
=>In Python Programming, Deep copy is implemented by Assigment Operator (=)

Syntax:-   listobj2=listobj1 # deep copy

Examples:

```
>>> l1=[10,"Rossum"]
>>> print(l1,id(l1))-------[10, 'Rossum'] 3180890162560
>>> l2=l1  # Deep Copy
```

```
>>> print(l2,id(l2))------------[10, 'Rossum'] 3180890162560
>>> l1.append("DS")
>>> print(l1,id(l1))----------[10, 'Rossum', 'DS'] 3180890162560
>>> print(l2,id(l2))---------[10, 'Rossum', 'DS'] 3180890162560
>>> l2.insert(1,"Travis")
>>> print(l1,id(l1))---------[10, 'Travis', 'Rossum', 'DS'] 3180890162560
>>> print(l2,id(l2))----------[10, 'Travis', 'Rossum', 'DS']
3180890162560
```

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

==

inner (or) nested list

==

=>The Process of writing one list inside of another list is called inner / nested list

=>Syntax:-

listobj=[v1,v2......vn,[v11,v12,...v1n],[v21,v22,...v2n] ......]

=>Here [v11,v12,...v1n],[v21,v22,...v2n] are called inner / nested list
=>On the inner list we can perform Both Indexing and slicing Operations.
=>On the inner list we can perform various operations by using pre-defined function of list.

Examples:

```
>>> stuinfo=[10,"RS",[13,19,15] ,[76,56,67],"OUCET"]
>>> print(stuinfo, type(stuinfo))-----[10, 'RS', [13, 19, 15], [76, 56,
67], 'OUCET']                                    <class 'list'>
>>> print(stuinfo[0])----------10
>>> print(stuinfo[2])---------[13, 19, 15]
>>> print(stuinfo[3])-----------[76, 56, 67]
>>> print(stuinfo[4])---------OUCET
>>> print(stuinfo[-2])----------[76, 56, 67]
>>> print(stuinfo[-3])---------[13, 19, 15]
>>> print(stuinfo[-3][0])--------13
>>> print(stuinfo[-3][-3])---------13
>>> print(stuinfo[3][3])---------IndexError: list index out of range
>>> print(stuinfo[3][::])---------[76, 56, 67]
>>> print(stuinfo[3][::-1])----------[67, 56, 76]
>>> print(stuinfo)-------[10, 'RS', [13, 19, 15], [76, 56, 67], 'OUCET']
>>> stuinfo[2].append(14)
>>> print(stuinfo)--------[10, 'RS', [13, 19, 15, 14], [76, 56, 67],
'OUCET']
>>> stuinfo[-2].insert(1,68)
>>> print(stuinfo)--------[10, 'RS', [13, 19, 15, 14], [76, 68, 56, 67],
'OUCET']
>>> stuinfo[-3].sort()
>>> print(stuinfo)-------[10, 'RS', [13, 14, 15, 19], [76, 68, 56, 67],
'OUCET']
>>> stuinfo[3].sort(reverse=True)
>>> print(stuinfo)---------[10, 'RS', [13, 14, 15, 19], [76, 68, 67, 56],
'OUCET']
>>> stuinfo.pop(2)------[13, 14, 15, 19]
>>> print(stuinfo)--------[10, 'RS', [76, 68, 67, 56], 'OUCET']
>>> stuinfo.pop(-2)--------[76, 68, 67, 56]
```

```
>>> print(stuinfo)---------[10, 'RS', 'OUCET']
>>> stuinfo.insert(1,[12,19,14,11])
>>> print(stuinfo)----------[10, [12, 19, 14, 11], 'RS', 'OUCET']
>>> stuinfo[1].pop(2)--------14
>>> print(stuinfo)------[10, [12, 19, 11], 'RS', 'OUCET']
>>> stuinfo.insert(3,[56,78,66,71])
>>> print(stuinfo)--------[10, [12, 19, 11], 'RS', [56, 78, 66, 71],
'OUCET']
>>> stuinfo.append(["Apple","Kiwi","Sberry","Banana"])
>>> print(stuinfo)---[10, [12, 19, 11], 'RS', [56, 78, 66, 71], 'OUCET',
['Apple', 'Kiwi','Sberry', 'Banana']  ]

>>> stuinfo.append("Apple","Kiwi","Sberry","Banana")---
                 TypeError: list.append() takes exactly one argument (4
given)
```

===

b) 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 unique and duplicates."

=>The elements of tuple must be written within braces ( ) and elements of tuple separated by comma.

=>An object of tuple maintains Insertion Oder.(In Whichever order we insert the data, in the same order elements will be displayed)

=>An object of tuple belongs to immutable.

=>On the object of tuple we can perform both Indexing and slicing operations.

=>We can convert other type value into tuple type value by using tuple()

=>We can create two types of tuple objects. They are

a) empty tuple

b) non-empty tuple

a) empty tuple:

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

```
Syntax:-    varname=( )
                 (or)
             varname=tuple()
```

----

b) non-empty tuple:

=>A non-empty tuple is one which contains elements and whose length is > 0

```
Syntax:-    varname=(val1,val2,.....val;-n)
```

NOTE:- The Functionality of tuple is exactly similar to Functionality of list but an object of list belongs to mutable where as an object of tuple is immutable.

Examples:

```
>>> t1=(10,20,30,40,50,60)
```

```
>>> print(t1,type(t1))------(10, 20, 30, 40, 50, 60) <class 'tuple'>
>>> t2=(10,"Ram",34.56,True,"PYTHON")
>>> print(t2,type(t2))---------(10, 'Ram', 34.56, True, 'PYTHON') <class
'tuple'>
>>> len(t1)--------6
>>> len(t2)-------5
>>> t3=()
>>> print(t3,type(t3))-------() <class 'tuple'>
>>> t4=tuple()
>>> print(t4,type(t4))------() <class 'tuple'>
>>> len(t3)--------0
>>> len(t4)--------0
>>> t5=(10,10,20,10,10,20)
>>> print(t5,type(t5))---------(10, 10, 20, 10, 10, 20) <class 'tuple'>
>>> t6=10,"KVR","PYTHON",34.56
>>> print(t6,type(t6))-----(10, 'KVR', 'PYTHON', 34.56) <class 'tuple'>
>>> t2=(10,"Ram",34.56,True,"PYTHON")
>>> print(t2,id(t2))-------(10, 'Ram', 34.56, True, 'PYTHON')
1620156415920
>>> print(t2[0])-------10
>>> print(t2[0:5])-------(10, 'Ram', 34.56, True, 'PYTHON')
>>> print(t2[0:3])-------(10, 'Ram', 34.56)
>>> t2[0]=100----TypeError: 'tuple' object does not support item
assignment
>>> l1=[10,"Ram",34.56]
>>> print(l1,type(l1))----------[10, 'Ram', 34.56] <class 'list'>
>>> t11=tuple(l1)
>>> print(t11,type(t11))-------(10, 'Ram', 34.56) <class 'tuple'>
=============================X============================
>>> t1=(10,"Ram",(14,13,17),[56,67,34],"OUCET")
>>> print(t1,type(t1))---(10, 'Ram', (14, 13, 17), [56, 67, 34], 'OUCET')
<class 'tuple'>
>>> print(t1[3])----[56, 67, 34]
>>> print(type(t1[3]))----<class 'list'>
>>> print(type(t1[2]))----<class 'tuple'>
>>> print(type(t1))-------<class 'tuple'>
>>> t1[3][0]=100   # possible --- bcoz t1[3] is inner list
>>> print(t1)----------(10, 'Ram', (14, 13, 17), [100, 67, 34], 'OUCET')
>>> t1[2][0]=100-------TypeError: 'tuple' object does not support item
assignment
```

Pre-defined Functions in tuple

----

1) index()
2) count()

Functions not present in tuple

----

append(), insert(), remove() pop(index), pop(), copy(), clear(), extend()
sort(), reverse()

==

IV. Set Category Data Types(Collection data types)

==

=>The puurpose of Set Category Data Types(Collection data types) is that "To store multiple values either of same type or different type or both the types with unique ."

=>we have two data types in Set Category. They are
a) set (Mutable and immutable)
b) frozenset (immutable)
====================================
a) set
====================================
=>'set' is a 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 ."

==
Pre-defined Functions in set
==
=>set object contains different pre-defined functions to perform various operations.
=>The pre-defined of set are shown bellow.

1) add():

=>This function is used for adding the elements to set object
=>Syntax:- setobj.add(element)

=>Examples:

```
>>> s1={10,20}
>>> print(s1,type(s1))--------{10, 20} <class 'set'>
>>> print(s1,id(s1))--------{10, 20} 2605270068224
>>> s1.add("Python")
>>> s1.add("Java")
>>> print(s1,id(s1))-------{'Python', 10, 20, 'Java'} 2605270068224
```
----

2)clear()

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

Examples:

```
>>> b={10,"Rossum",23.45,"Python",True,"Python"}
>>> print(b,id(b))--------{True, 23.45, 'Rossum', 10, 'Python'}
2605270068672
>>> b.clear()
>>> print(b,id(b))---------set() 2605270068672
```
----

3)remove()

=>This Function is used for removing an element (key) from set object.
=>if Element does not exists in set object then we get KeyError
=>Syntax:- setobj.remove(element)

Examples:

```
>>> b={10,"Rossum",23.45,"Python",True,"Python"}
>>> print(b,id(b))----------{True, 23.45, 'Rossum', 10, 'Python'}
2605270068448
>>> b.remove("Rossum")
```

```
>>> print(b,id(b))----------{True, 23.45, 10, 'Python'} 2605270068448
>>> b.remove("Python")
>>> print(b,id(b))---------{True, 23.45, 10} 2605270068448
>>> b.remove("Rossum")------KeyError: 'Rossum'
>>> b.remove(100)-------KeyError: 100
```

----
4) discard():

=>This Function is used for removing/discarding an element (key) from set object.
=>if Element does not exists in set object then we never get any error
=>Syntax:-   setobj.discard(element)

Examples:

```
>>> b={10,"Rossum",23.45,"Python",True,"Python"}
>>> print(b,id(b))--------{True, 23.45, 'Rossum', 10, 'Python'}
2605270066880
>>> b.discard("Rossum")
>>> print(b,id(b))----------{True, 23.45, 10, 'Python'} 2605270066880
>>> b.discard("Rossum")
>>> print(b,id(b))--------{True, 23.45, 10, 'Python'} 2605270066880
```

----
5) pop():

=>It is used for used removing any arbitrary element from set object.
=>If we call pop() upon empty set object then we get KeyError
=>Syntax:-   setobj.pop()

=>Examples:

```
>>> b={10,"Rossum",23.45,"Python",True,"Python"}
>>> b.pop()-----------True
>>> b.pop()----------23.45
>>> b.pop()-----------'Rossum'
>>> b.pop()----------10
>>> b.pop()-----------'Python'
>>> b.pop()------KeyError: 'pop from an empty set'
>>> print(b)-----------set()
--------------------------------------------
>>> a={10,20,30,40,50,60,70,80,"python","java"}
>>> print(a)------{'java', 'python', 70, 40, 10, 80, 50, 20, 60, 30}
>>> a.pop()---------'java'
>>> a.pop()---------'python'
>>> a.pop()-------70
>>> a.pop()-------40
>>> a.pop()--------10
>>> a.pop()-------80
>>> print(a)--------{50, 20, 60, 30}
>>> a.pop()--------50
>>> print(a)------{20, 60, 30}
>>> a.pop()--------20
>>> print(a)--------{60, 30}
>>> a.pop()------60
>>> print(a)-------{30}
>>> a.pop()-------30
```

```
>>> print(a)------set()
>>> a.pop()-------KeyError: 'pop from an empty set'
```
----
6) isdisjoint() :

=>This Function returns True provided when there is no common elements in both set objects.
=>This Function returns False provided when there is at least one common elements in both set objects.

=>Syntax:-   setobj1.isdisjoint(setobj2)

Examples:

```
>>> s1={10,20,30,40}
>>> s2={10,50,60}
>>> s3={-10,-20,-30}
>>> print(s1,type(s1))-----------{40, 10, 20, 30} <class 'set'>
>>> print(s2,type(s2))---------{10, 50, 60} <class 'set'>
>>> print(s3,type(s3))----------{-30, -20, -10} <class 'set'>
>>> s1.isdisjoint(s2)----------False
>>> s1.isdisjoint(s3)-----------True
>>> {10,20,30}.isdisjoint({30,40,50})---------False
>>> {10,20,30}.isdisjoint({300,40,50})----------True
>>> {10,20,30}.isdisjoint( set() )---------True
>>> set().isdisjoint( set() )-----------True
```

=>The elements of set must be organized within the curly braces { } and elements must be separated by commal.
=>An object of set never maintains insertion order. bcoz set object elements can be displayed in any of its possibilities.
=>We create two types of set objects. They are
 a) empty set
 b) non-empty set
=>An empty is one, which does not contain any elements and whose size is 0
=>Syntax: setobj=set()
=>Examples: s=set()
=>A non-empty is one, which contains elements and whose size is >0
=>Syntax: setobj={v1,v2...vn}
=>Examples: s1={10,20,30,40,50,60,10}
 s2={10,"Rossum",23.45,True}

=>On the object of set, we can't perform Indexing and slicing operations bcoz it can't maintain insertion order.
=>An object of set belongs to both Mutable (in the case add()) and immutable (in the case of item assigment)
--X---
Examples:

```
>>> a={10,20,30,40,10,20,30,40}
>>> print(a,type(a))---------{40, 10, 20, 30} <class 'set'>
>>> b={10,"Rossum",23.45,"Python",True,"Python"}
```

```
>>> print(b,type(b))-----{True, 23.45, 'Rossum', 10, 'Python'} <class
'set'>
>>> a={}
>>> print(a,type(a))---------{} <class 'dict'>
>>> a=set()  # empty set
>>> print(a,type(a))------- set() <class 'set'>
>>> b={10,"Rossum",23.45,"Python",True,"Python"}
>>> print(b)------{True, 23.45, 'Rossum', 10, 'Python'}
>>> print(b[0])-------TypeError: 'set' object is not subscriptable
>>> print(b[0:4])-------TypeError: 'set' object is not subscriptable
>>> print(b[::-1])--------TypeError: 'set' object is not subscriptable
>>> b={10,"Rossum",23.45,"Python",True,"Python"}
>>> print(b)-------{True, 23.45, 'Rossum', 10, 'Python'}
>>> b[0]=100  # update the elements of set
                 TypeError: 'set' object does not support item assignment
>>> b={10,"Rossum",23.45,"Python",True,"Python"}
>>> print(b,id(b))---{True, 23.45, 'Rossum', 10, 'Python'} 2605270068000
>>> b.add("KVR") # adding the elements to set object
>>> print(b,id(b))------{True, 23.45, 'Rossum', 10, 'KVR', 'Python'}
2605270068000
-------------------------------------X------------------------------------
----------------------
```

==
Pre-defined Functions in set
==

=>set object contains different pre-defined functions to perform various operations.
=>The pre-defined of set are shown bellow.

1) add():

=>This function is used for adding the elements to set object
=>Syntax:-   setobj.add(element)

=>Examples:

```
>>> s1={10,20}
>>> print(s1,type(s1))--------{10, 20} <class 'set'>
>>> print(s1,id(s1))--------{10, 20} 2605270068224
>>> s1.add("Python")
>>> s1.add("Java")
>>> print(s1,id(s1))-------{'Python', 10, 20, 'Java'} 2605270068224
```

----

2)clear()

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

Examples:

```
>>> b={10,"Rossum",23.45,"Python",True,"Python"}
>>> print(b,id(b))--------{True, 23.45, 'Rossum', 10, 'Python'}
2605270068672
>>> b.clear()
>>> print(b,id(b))---------set() 2605270068672
```

```
--------------------------------------------------------------------------
-------------------
3)remove()
-------------------
=>This Function is used for removing an element (key) from set object.
=>if Element does not exists in set object then we get KeyError
=>Syntax:-      setobj.remove(element)
-----------------
Examples:
----------------
>>> b={10,"Rossum",23.45,"Python",True,"Python"}
>>> print(b,id(b))----------{True, 23.45, 'Rossum', 10, 'Python'}
2605270068448
>>> b.remove("Rossum")
>>> print(b,id(b))----------{True, 23.45, 10, 'Python'} 2605270068448
>>> b.remove("Python")
>>> print(b,id(b))---------{True, 23.45, 10} 2605270068448
>>> b.remove("Rossum")------KeyError: 'Rossum'
>>> b.remove(100)-------KeyError: 100
--------------------------------------------------------------------------
--------------
4) discard():
--------------------
=>This Function is used for removing/discarding an element (key) from set
object.
=>if Element does not exists in set object then we never get any error
=>Syntax:-      setobj.discard(element)
-------------------
Examples:
-------------------
>>> b={10,"Rossum",23.45,"Python",True,"Python"}
>>> print(b,id(b))--------{True, 23.45, 'Rossum', 10, 'Python'}
2605270066880
>>> b.discard("Rossum")
>>> print(b,id(b))----------{True, 23.45, 10, 'Python'} 2605270066880
>>> b.discard("Rossum")
>>> print(b,id(b))--------{True, 23.45, 10, 'Python'} 2605270066880
--------------------------------------------------------------------------
-----------------------------
5) pop():
-----------------------------
=>It is used for used removing any arbitrary element from set object.
=>If we call pop() upon empty set object then we get KeyError
=>Syntax:-    setobj.pop()
------------------------
=>Examples:
-------------------------
>>> b={10,"Rossum",23.45,"Python",True,"Python"}
>>> b.pop()-----------True
>>> b.pop()----------23.45
>>> b.pop()-----------'Rossum'
>>> b.pop()----------10
>>> b.pop()-----------'Python'
>>> b.pop()------KeyError: 'pop from an empty set'
>>> print(b)-----------set()
--------------------------------------------------------------------------
---------------------------------
6) isdisjoint() :
```

=>This Function returns True provided when there is no common elements in both set objects.
=>This Function returns False provided when there is at least one common elements in both set objects.

=>Syntax:-   setobj1.isdisjoint(setobj2)

Examples:

```
>>> s1={10,20,30,40}
>>> s2={10,50,60}
>>> s3={-10,-20,-30}
>>> print(s1,type(s1))-----------{40, 10, 20, 30} <class 'set'>
>>> print(s2,type(s2))---------{10, 50, 60} <class 'set'>
>>> print(s3,type(s3))----------{-30, -20, -10} <class 'set'>
>>> s1.isdisjoint(s2)----------False
>>> s1.isdisjoint(s3)-----------True
>>> {10,20,30}.isdisjoint({30,40,50})---------False
>>> {10,20,30}.isdisjoint({300,40,50})----------True
>>> {10,20,30}.isdisjoint( set() )---------True
>>> set().isdisjoint( set() )-----------True
```
----
7) issuperset():

Syntax:-   setobj1.issuperset(setobj2)
=>This Function returns True Provided all the of setobj2 are present in setobj1 otherwise it returns False
8) issubset():

Syntax:-   setobj1.issubset(setobj2)
=>This Function returns True Provided all the of setobj1 are present in setobj1 otherwise it returns False
Examples:

```
>>> s1={10,20,30,40,50}
>>> s2={10,20,30}
>>> s3={10,60,70}
>>> s1.issuperset(s2)-----------True
>>> s1.issuperset(s3)---------False
>>> s2.issubset(s1)----------True
>>> s3.issubset(s1)-----------False
>>> s3.issubset(s2)-----------False
>>> s1.issubset(s1)--------True
>>> set().issubset(set())--------True
>>> set().issubset(s1)--------True
>>> s3.issubset(s3)--------True
```
----
9) union()

Syntax:-   setobj3=setobj1.union(setobj2)
=>This function obtains all the unique elements both setobj1 and setobj2 and place the resultant values in setobj3.

Examples:

```
------------------
>>> s1={10,20,30,40}
>>> s2={40,30,50,60,70}
>>> s3=s1.union(s2)
>>> print(s3)-----------{70, 40, 10, 50, 20, 60, 30}
-----------------------------------------------------------------------------
------
10) intersection():
---------------------------
=>Syntax:   setobj3=setobj1.intersection(setobj2)
=>This obtains common elements from setobj1 and setobj2 and place the
resultant elements in setobj3.
-----------------
Examples:
-----------------
>>> s1={10,20,30,40}
>>> s2={40,30,50,60,70}
>>> s4=s1.intersection(s2)
>>> print(s4)---------{40, 30}
-----------------------------------------------------------------------------
------
11) difference() :
--------------------------
Syntax1:-       setobj3=setobj1.difference(setobj2)
-------------      This Function removes common elements from both
setobj1 and setobj2 and takes the remaining elements from setobj1 and
place them setobj3

Syntax2:-       setobj3=setobj2.difference(setobj1)
-------------      This Function removes common elements from both
setobj2 and setobj1 and takes the remaining elements from setobj2 and
place them setobj3

Examples:
---------------
>>> s1={10,20,30,40}
>>> s2={40,30,50,60,70}
>>> s5=s1.difference(s2)
>>> print(s5)---------{10, 20}
>>> s6=s2.difference(s1)
>>> print(s6)---------{50, 60, 70}
-----------------------------------------------------------
12) symmetric_difference():
------------------------------------------
Syntax:  setobj3=setobj1.symmetric_difference(setobj2)
=> This function removes the common elements from both setobj1 and
setobj2 and takes the remaining elements from setobj1 and setobj2 and
place the resultant values in setobj3.

Examples:
------------------
>>> s1={10,20,30,40}
>>> s2={40,30,50,60,70}
>>> s7=s1.symmetric_difference(s2)
>>> print(s7)-----------{50, 20, 70, 10, 60}
>>> s7=s2.symmetric_difference(s1)
>>> print(s7)----------{70, 10, 50, 20, 60}
```

```
--------------------------------------------------------------------------
---------------------------------
  Special Cases:
  ------------------------
  >>> s1={10,20,30,40}
>>> s2={40,30,50,60,70}
>>> s3=s1.union(s2)
>>> print(s3)-----------{70, 40, 10, 50, 20, 60, 30}
>>> s3=s1|s3
>>> print(s3)----------{70, 40, 10, 50, 20, 60, 30}
>>> s4=s1.intersection(s2)
>>> print(s4)--------{40, 30}
>>> s4=s1&s2
>>> print(s4)-----------{40, 30}
>>> s5=s1.difference(s2)
>>> print(s5)------------{10, 20}
>>> s5=s1-s2
>>> print(s5)----------{10, 20}
>>> s5=s2.difference(s1)
>>> print(s5)----------{50, 60, 70}
>>> s5=s2-s1
>>> print(s5)-------------{50, 60, 70}
>>> s1={10,20,30,40}
>>> s2={40,30,50,60,70}
>>> s3=s1.symmetric_difference(s2)
>>> print(s3)----------{50, 20, 70, 10, 60}
>>> s3=s1^s2
>>> print(s3)-----------{50, 20, 70, 10, 60}
--------------------------------------------------------------------------
--
Case study:
----------------------
>>> tp={"Ram","Lax","Raj"}
>>> cp={"Sachin","Kohli","Ram"}
>>> bothcptp=tp.union(cp)
>>> print(bothcptp)-----------{'Sachin', 'Kohli', 'Raj', 'Ram', 'Lax'}
>>> comcptp=cp.intersection(tp)
>>> print(comcptp)------------{'Ram'}
>>> comcptp=cp&tp
>>> print(comcptp)-------------{'Ram'}
>>> onlytp=tp.difference(cp)
>>> print(onlytp)-----------{'Raj', 'Lax'}
>>> onlycp=cp.difference(tp)
>>> print(onlycp)------------{'Sachin', 'Kohli'}
>>> onlytp=tp-cp
>>> print(onlytp)------------{'Raj', 'Lax'}
>>> onlycp=cp-tp
>>> print(onlycp)-----------{'Sachin', 'Kohli'}
>>> excptp=cp.symmetric_difference(tp)
>>> print(excptp)-----------{'Sachin', 'Lax', 'Kohli', 'Raj'}
>>> excptp=cp^tp
>>> print(excptp)--------------{'Sachin', 'Lax', 'Kohli', 'Raj'}
--------------------------------------------------------------------------
--------------------------------
13)update:
--------------------
=>Syntax:-    setobj1.update(setobj2)
=>This Function updates / add all the values of setobj2 to setobj1 .
```

Examples:

```
>>> s1={10,"Sai"}
>>> s2={"Python","Data Science","ML","DL"}
>>> s3=s1.update(s2)
>>> print(s1)----------{'Python', 'DL', 'ML', 'Data Science', 10, 'Sai'}
>>> print(s2)-------{'Data Science', 'DL', 'Python', 'ML'}
>>> print(s3)---------None
--------------------------------------------------
>>> s1={10,"Sai"}
>>> s2={"Python","Data Science","ML","DL"}
>>> print(s1,id(s1))-------------{10, 'Sai'} 1645435971712
>>> print(s2,id(s2))----------{'Data Science', 'DL', 'Python', 'ML'}
1645435974624
>>> s1.update(s2)
>>> print(s1,id(s1))----{'Python', 'DL', 'ML', 'Data Science', 10, 'Sai'}
1645435971712
>>> print(s2,id(s2))----{'Data Science', 'DL', 'Python', 'ML'}
1645435974624
>>> s1={10,20,30,40}
>>> s1.update({10,30,40,"Python"})
>>> print(s1)--------{40, 10, 20, 'Python', 30}
```

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

frozenset

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

=>'frozenset' is a pre-defined class and treated as Set Data Types.
=>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 ."
=>To represent the elemnts of frozenset , we don't have any symbolic notation but we can convert the elements list , tuple, set type elements into frozenset type by using frozenset()
=>An object of frozenset never maintains insertion order. bcoz frozenset object elements can be displayed in any of its possibilities.
=>We create two types of frozenset objects. They are

a) empty frozenset
b) non-empty frozenset

=>An empty frozen set is one, which does not contain any elements and whose size is 0
=>Syntax: fssetobj=frozenset()
=>Examples: fs=frozenset()
=>A non-empty frozenset is one, which contains elements and whose size is >0
=>Syntax: setobj=frozenset({v1,v2...vn})
=>Examples: s1=frozenset({10,20,30,40,50,60,10})
s2=frozenset({10,"Rossum",23.45,True})

=>On the object of frozenset, we can't perform Indexing and slicing operations bcoz it can't maintain insertion order.
=>An object of frozenset belongs to immutable (in the case add() , in the case of item assigment)

==

Dict Category Data Types(Collection data types)
==

=>Dict Category Data Type contains a data type called 'dict'
=>'dict' is one of the pre-defined class and treated as Dict Category Data Type.
=>The purpose of dict data type is that "To Store the data in the form of (Key,Value) "
=>In (Key,Value) , the value of Key are unique and Values of Value may or may not be unique.
=>The elements of dict must be organized in the form of (Key,value) anf they must be written within {} .
=>On the object of dict we can't perform Indexing and Slicing operations bcoz we have keys to access values.
=>An object is mutable bczo we can chamge / update the values of value by passing value of Key. Hence values of Value of dict are mutable and values of Key are immutable.
=>We can create 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 elements and whose size is 0
=>Syntax:- dictobj={}
Examples: d1={}
=>Syntax for adding (Key,Value) to the empty dict object

```
dictobj[KeyName1]=Value1
dictobj[KeyName2]=Value2
--------------------------------------
dictobj[KeyName-n]=Value-n
```

Examples:

```
>>> d3={}
>>> print(d3,type(d3))-----------{} <class 'dict'>
>>> len(d3)---------0
>>> d3[100]=2.3
>>> d3[200]=4.5
>>> d3[300]=2.3
>>> d3[400]=6.3
>>> print(d3,type(d3))--------{100: 2.3, 200: 4.5, 300: 2.3, 400: 6.3}
<class 'dict'>
>>> len(d3)--------4
>>> d3[500]=5.5
>>> print(d3,type(d3))----{100: 2.3, 200: 4.5, 300: 2.3, 400: 6.3, 500:
5.5} <class 'dict'>
>>> d3[200]=9.8
>>> print(d3,type(d3))----{100: 2.3, 200: 9.8, 300: 2.3, 400: 6.3, 500:
5.5} <class 'dict'>
```

----

b) non-empty dict:

=>non-Empty dict is one, which contains elements and whose size is > 0
Syntax:- dictobj={Keyname1:Value1,KeyName2:Value2...KeyName-n:Value-n}

Examnples:-

```
>>> d1={10:"Apple", 20:"Mango",30:"Sberry",40:"Kiwi"}
>>> print(d1,type(d1))---{10: 'Apple', 20: 'Mango', 30: 'Sberry', 40:
'Kiwi'} <class
                                   'dict'>
>>> d2={"Python":"RS","Java":"JG","C":"DR",".NET":"MS"}
>>> print(d2,type(d2))---{'Python': 'RS', 'Java': 'JG', 'C': 'DR',
'.NET': 'MS'} <class
                  'dict'>
>>> len(d1)-------4
>>> len(d2)---------4
>>> d1[10]="PApple"
>>> print(d1,type(d1))--{10: 'PApple', 20: 'Mango', 30: 'Sberry', 40:
'Kiwi'} <class
      'dict'>
```

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

pre-defined functions in dict

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

=>To perform Various operations on dict, we need to know the pre-defined functions in dict.

----

1) clear()

=>This function clears / removes all the entries of dict object.

=>Syntax:- dictobj.clear()

Examples:

```
>>> d1={10:"Apple", 20:"Mango",30:"Sberry",40:"Kiwi"}
>>> print(d1,id(d1))---{10: 'Apple', 20: 'Mango', 30: 'Sberry', 40:
'Kiwi'}                                          2968969442944
>>> d1.clear()
>>> print(d1,id(d1))----------- {}   2968969442944
```
-- -

2)copy():

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

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

=>Examples:

```
>>> d1={10:"Apple", 20:"Mango",30:"Sberry",40:"Kiwi"}
>>> d2=d1.copy()  # shallow copy
>>> print(d1,id(d1))---{10: 'Apple', 20: 'Mango', 30: 'Sberry', 40:
'Kiwi'}                                                2968969445184
>>> print(d2,id(d2))----{10: 'Apple', 20: 'Mango', 30: 'Sberry', 40:
'Kiwi'}                                          2968969442944
>>> d1[50]="Guava"
>>> d2[60]="Wmillon"
>>> print(d1,id(d1))---{10: 'Apple', 20: 'Mango', 30: 'Sberry', 40:
'Kiwi', 50: 'Guava'}                             2968969445184
>>> print(d2,id(d2))---{10: 'Apple', 20: 'Mango', 30: 'Sberry', 40:
'Kiwi', 60: 'Wmillon'}                           2968969442944
```
----

3) pop():

=>This function is used for removing (Key,Value) from dict object by passing Value of Key
=>If the Value of Key does not exists then we get KeyError
=>Syntax:-   dictobj.pop(key)

Examples:

```
>>> d1={10:"Apple", 20:"Mango",30:"Sberry",40:"Kiwi"}
>>> d1.pop(20)-----------'Mango'
>>> print(d1)----------{10: 'Apple', 30: 'Sberry', 40: 'Kiwi'}
>>> d1.pop(30)----------'Sberry'
>>> print(d1)------------{10: 'Apple', 40: 'Kiwi'}
>>> d1.pop(10)----------'Apple'
>>> print(d1)----------{40: 'Kiwi'}
>>> d1.pop(40)-----------'Kiwi'
>>> print(d1)-------------{}
>>> d1.pop(10)------------KeyError: 10
```
----

4) popitem():

=>This function is used fort removing last entry (Key,value) from dict object.
=>If we call popitem() on empty dict object then we get KeyError.
=>Syntax:-   dictobj.popitem()

Examples:

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

5) keys():

=>This function is used for obtaining set of keys.(Values of Key)
=>Syntax:-   dictobj.keys()

Examples:

```
>>> d1={10:"Apple", 20:"Mango",30:"Sberry",40:"Kiwi"}
>>> d1.keys()-----------dict_keys([10, 20, 30, 40])
>>> for k in d1.keys():
...     print(k)
                 ...
                 10
                 20
                 30
                 40
```

```
>>> {}.keys()-----------dict_keys([])
--------------------------------------------------------------------------
------------------
6)values():
----------------
=>This function is used for obtaining set of Values.(Values of Value)
=>Syntax:-      dictobj.values()
Examples:
-------------------
>>> d1={10:"Apple", 20:"Mango",30:"Sberry",40:"Kiwi"}
>>> print(d1)-----{10: 'Apple', 20: 'Mango', 30: 'Sberry', 40: 'Kiwi'}
>>> d1.values()----dict_values(['Apple', 'Mango', 'Sberry', 'Kiwi'])
>>> for v in d1.values():
...     print(v)
...
                 Apple
                 Mango
                 Sberry
                 Kiwi
>>> {}.values()------------dict_values([])
--------------------------------------------------------------------------
--------------------------
7) get():
------------
=>This function is used for obtaining Value of Value by Passing Value of
Key.

Examples:-
-------------------
>> d1={10:"Apple", 20:"Mango",30:"Sberry",40:"Kiwi"}
>>> print(d1)------{10: 'Apple', 20: 'Mango', 30: 'Sberry', 40: 'Kiwi'}
>>> d1[30]---------'Sberry'
>>> d1.get(30)--------'Sberry'
>>> d1.get(10)----------'Apple'
>>> d1.get(40)-----------'Kiwi'
---------------------------------------------------------
8)items():
------------------
=>This Function is used for obtaining both (Key,Value)
=>Syntax:-   dictobj.items()
-----------------------
Examples:
----------------------
>>> d1={10:"Apple", 20:"Mango",30:"Sberry",40:"Kiwi"}
>>> print(d1)---------{10: 'Apple', 20: 'Mango', 30: 'Sberry', 40:
'Kiwi'}
>>> d1.items()---dict_items([(10, 'Apple'), (20, 'Mango'), (30,
'Sberry'), (40, 'Kiwi')])
>>> for kv in d1.items():
...     print(kv)
            ...
            (10, 'Apple')
            (20, 'Mango')
            (30, 'Sberry')
            (40, 'Kiwi')
>>> for k,v in d1.items():
...     print(k,"-->",v)
                 ...
```

```
                 10 --> Apple
                 20 --> Mango
                 30 --> Sberry
                 40 --> Kiwi
Note:->>> d1={10:"Apple", 20:"Mango",30:"Sberry",40:"Kiwi"}
>>> print(d1)
{10: 'Apple', 20: 'Mango', 30: 'Sberry', 40: 'Kiwi'}
>>> for x in d1:
...     print(x)
...
10
20
30
40
```
----

9)update()

```
=>Syntax:-   dictobj1.update(dictobj2)
=>This function is used for updating dictobject1 with dictobject2.
```

Examples:

```
>>> d1={10:"Ramu","place":"Hyd"}
>>> print(d1)---------------{10: 'Ramu', 'place': 'Hyd'}
>>> d2={"crs1":"python","crs2":"Java","crs3":"Data Sci"}
>>> print(d2)-----------{'crs1': 'python', 'crs2': 'Java', 'crs3': 'Data
Sci'}
>>> d1.update(d2)
>>> print(d1)----{10: 'Ramu', 'place': 'Hyd', 'crs1': 'python', 'crs2':
'Java', 'crs3': 'Data Sci'}
>>> d1={10:2.3,20:3.4}
>>> d2={30:3.4,10:5.6}
>>> print(d1)-------------{10: 2.3, 20: 3.4}
>>> print(d2)--------------{30: 3.4, 10: 5.6}
>>> d1.update(d2)
>>> print(d1)---------------{10: 5.6, 20: 3.4, 30: 3.4}
>>> d1={"Ramu":"C","Raj":"Pascal"}
>>> d2={"Ramu":"Python","Raj":"Django"}
>>> print(d1)-----------{'Ramu': 'C', 'Raj': 'Pascal'}
>>> print(d2)----------{'Ramu': 'Python', 'Raj': 'Django'}
>>> d1.update(d2)
>>> print(d1)-----------------{'Ramu': 'Python', 'Raj': 'Django'}
===============================X==============================
```

==

NoneType Category Data Type

==

=>"NoneType" is one of the pre-defined class and treated None type data type
=>'None' is keyword and it is treated as the value of of <class, 'NoneType'>
=>The 'None' is not Null , space and False.
=>An object of NoneType can be created.

Examples:

```
>>> n=NoneType()--------NameError: name 'NoneType' is not defined
-------------------------------------------------
>>> a=None
```

```
>>> print(a,type(a))-----------None <class 'NoneType'>
```

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

===
No. of Approaches to develop the Python Program
===

=>In Python Environment, we have two approaches to develop the program. They are

a) Interactive Approach
b) Batch Mode Approach

a) Interactive Approach:

=> In This approach , as Programmer , we are giving one statement at a time and getting result at a time for that statement.
=>This approach is most useful to test one statement at a time (means whether the statement is working or not)

Example: Python Command Prompt (coming on the installation Python Software)

```
>>>a=10
>>>b=20
>>>c=a+b
```

=>These statements are unable to save to view in the future and it is not at all suitable for Problem solving. To solve the poroblems in real time we must always use Batch Mode approaches.
----

b) Batch Mode Approach:

=>This approach says that The python Programmer write batch / group of executable statements and saved on some file name with an extension .py [Ex: sum.py]

Examples:- a) Python IDLE Shell (coming on the installation Python Software)
b) Edit Plus
c) PyCharm
d) Spider
e) sublime text
f) jupiter note book etc
----

a) Python IDLE Shell :

=> Lauch the Python IDLE shell
=> Choose File--> New File (ensure that a new window will be opened)
=>Write the python Program
=>Save the Python Program (File-->Save (ctrl+s)) on some file name with an extension .py in a separate Folder
=>Run the Python Program (Choose Run-->Run Module (F5))
=>View the result

(OR)

=>To execute the python program from command prompt, we use a tool called "py" (or) "python"

```
Syntax:-     py  filename.py

                  (or)

             python filename.py
```

Example:

```
#Program for multiplying two numbers
#mul.py
a=10
b=2
c=a*b
print("Val of a=",a)
print("val of b=",b)
print("Mul=",c)
```


```
G:\KVR-PYTHON-4PM\FUNDAS-PROG>py mul.py
Val of a= 10
val of b= 2
Mul= 20

G:\KVR-PYTHON-4PM\FUNDAS-PROG>python mul.py
Val of a= 10
val of b= 2
Mul= 20
```
===

==

Displaying the result of python Program

==

=>To display the result of Python Program on the console (Monitor) , we use a pre-defined function called print()
=>In Otherwords, print() is used for displaying the result of the python program on the console.
=>print() contains Various syntaxes . They are

Syntax-1 :   print(Message)
=>This Syntax displays the messages (str) on the console.
Examples:

```
>>> print("Hello Python World")------Hello Python World
>>> print('Hello Python World')-----Hello Python World
>>> print('''Hello Python World''')----Hello Python World
>>> print("""Hello Python World""")----Hello Python World
```

Syntax-2:   print(val1,val2...val-n)
-------------   This Syntax displays the values on the console.

Examples:
```
>>> a=10
>>> print(a)-------------10
>>> stno=10
>>> sname="Rossum"
>>> print(stno,sname)------------10 Rossum
```

```
Syntax-3:    print(messages cum values)
=>This syntax displays messages and values together.
>>> a=10
>>> b=20
>>> c=a+b
>>> print("val of a=",a)----------val of a= 10
>>> print("""val of a=""",a)--------val of a= 10
>>> print(a,"is the val of a")--------10 is the val of a
>>> a=100
>>> print(a,"is the val of a")-----100 is the val of a
>>> a=10
>>> b=20
>>> c=a+b
>>> print("sum=",c)------sum= 30
>>> print(c," is the sum")----30  is the sum
>>> print("sum of ",a," and ",b,"=",c)----sum of  10  and  20 = 30
>>> stno=10
>>> sname="Rossum"
>>> print("My Number is ",stno," and my name is ",sname)
My Number is  10  and my name is  Rossum
---------------------------------------------------------------------------
------------
Syntax-4:  print(Messages cum Values with format() )
------------
>>> stno=10
>>> sname="Rossum"
>>> stno=10
>>> sname="Rossum"
>>> print("My Number is {} and My Name is {}".format(stno,sname))
                  My Number is 10 and My Name is Rossum
>>> a=10
>>> b=20
>>> c=a+b
>>> print("sum of {} and {}={}".format(a,b,c))----sum of 10 and 20=30
>>> print("sum({},{})={}".format(a,b,c))-----sum(10,20)=30
>>> print("sum({},{})={}\t sub({},{})={}".format(a,b,a+b,a,b,a-b))
                              sum(10,20)=30    sub(10,20)=-10
---------------------------------------------------------------------------
----------------------
Syntax-4:  print(Messages cum Values with format specifiers )
------------
>>> a=10
>>> b=1.2
>>> c=a+b
>>> print("sum=",c)---------sum= 11.2
>>> print("sum={}".format(c))---------sum=11.2
>>> print("sum=%f" %c)--------sum=11.200000
>>> print("sum=%0.2f" %c)--------sum=11.20
>>> print("sum of %d and %f=%0.1f" %(a,b,c))---sum of 10 and
1.200000=11.2
>>> print("sum of %d and %0.1f=%0.1f" %(a,b,c))--sum of 10 and 1.2=11.2
>>> print("sum of %f and %0.1f=%0.1f" %(a,b,c))--sum of 10.000000 and
1.2=11.2
>>> print("sum of %0.1f and %0.1f=%0.1f" %(a,b,c))--sum of 10.0 and
1.2=11.2
>>> stno=10
>>> sname="Rossum"
>>> print("My Number is %d and Name is %s" %(stno,sname))
```

```
                             My Number is 10 and Name is Rossum
>>> b=True
>>> print("Val of b=%s" %str(b) )----Val of b=True
>>> print("Val of b={}".format(b))----Val of b=True
>>> a=100
>>> print("Val of a=%d" %a)----Val of a=100
>>> print("Val of a=%s" %str(a))---Val of a=100
>>> print("Val of a=%f" %float(a))---Val of a=100.000000
```

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

==

Reading the data Dynamically from Key Board

==

=>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 any type of data from key board in the form of str.
=>Programatically, we can convert str data into any other data type by using Type
Casting Techniques (int(), float(), bool() , complex().......etc)

=>Syntax:- varname=input()
=>here Varname is of type <class,'str'>
=>We can convert str value into any other data type by using type casting techniques.
=>input() can read any type value from key board.

Examples:

```
#Program finding mul of two numbers
#dataread1.py
print("Enter First Value:")
a=input()
print("Enter Second Value:")
b=input()
print("Val of a={} and is type={}".format(a,type(a)))
print("Val of b={} and is type={}".format(b,type(b)))
print("--------------------------")
x1=float(a)
x2=float(b)
x3=x1*x2
print("mul({},{})={}".format(x1,x2,x3))
------------------------OR-------------------------------------
#dataread2.py
#Program finding mul of two numbers
print("Enter Two  Values:")
a=input()
b=input()
x1=float(a)
x2=float(b)
x3=x1*x2
print("mul(%0.2f,%0.2f)=%0.2f " %(x1,x2,x3))
-----------------------OR-------------------------
```

```
#dataread3.py
#Program for finding mul of two numbers
print("Enter Two  Values:")
x1=float(input())
x2=float(input())
print("mul of {} and {}={}".format(x1,x2,x1*x2))
```
----

2) input(Message)

=>This function is used for reading any type of data from key board in the form of str and additionally it can give user prompting messages.

Syntax:-   varname=input(Message)

=>here Varname is of type <class,'str'>

=>input() can read any type value from key board

=>"Message" is of str and it can represent any user-prompting message.

Examples:

```
#program for accepting two values from KBD and multiply them
#dataread4.py
a=input("Enter First Value:")
b=input("Enter Second Value:")
x1=float(a)
x2=float(b)
x3=x1*x2
print("Mul({},{})={}".format(x1,x2,x3))
```
--------------------OR--

```
#program for accepting two values from KBD and multiply them
#dataread5.py
x1=float(input("Enter First Value:"))
x2=float(input("Enter Second Value:"))
x3=x1*x2
print("Mul({},{})={}".format(x1,x2,x3))
```

```
1.#Program finding mul of two numbers
print("Enter First Value:")
a=input()
print("Enter Second Value:")
b=input()
print("Val of a={} and is type={}".format(a,type(a)))
print("Val of b={} and is type={}".format(b,type(b)))
print("--------------------------")
x1=float(a)
x2=float(b)
x3=x1*x2
print("mul({},{})={}".format(x1,x2,x3))

2.#dataread2.py
#Program finding mul of two numbers
print("Enter Two  Values:")
a=input()
b=input()
x1=float(a)
x2=float(b)
x3=x1*x2
```

```
print("mul(%0.2f,%0.2f)=%0.2f " %(x1,x2,x3))
```

3.#dataread3.py

```
#Program for finding mul of two numbers
print("Enter Two  Values:")
x1=float(input())
x2=float(input())
print("mul of {} and {}={}".format(x1,x2,x1*x2))
```

4.#program for accepting two values from KBD and multiply them

```
#dataread4.py
a=input("Enter First Value:")
b=input("Enter Second Value:")
x1=float(a)
x2=float(b)
x3=x1*x2
print("Mul({},{})={}".format(x1,x2,x3))
```

5.#program for accepting two values from KBD and multiply them

```
#dataread5.py
x1=float(input("Enter First Value:"))
x2=float(input("Enter Second Value:"))
x3=x1*x2
print("Mul({},{})={}".format(x1,x2,x3))
```

6.#program for accepting two values from KBD and multiply them

```
#dataread6.py
x3=float(input("Enter First Value:")) * float(input("Enter Second
Value:"))
print("Mul={}".format(x3))
```

7.#program for accepting two values from KBD and multiply them

```
#dataread7.py
print("Mul={}".format(float(input("Enter First Value:")) *
float(input("Enter Second Value:"))))
```

8.#Program for calculating simple interest and total amount to pay

```
#simpleint.py
p=float(input("Enter Principle Amount:"))
t=float(input("Enter Time:"))
r=float(input("Enter Rate of Interest:"))
#cal si and totamt
si=(p*t*r)/100
totamt=p+si
#display the values
print("==========================")
print("\tR e s u l t s")
print("==========================")
print("Principle Amount={}".format(p))
print("Time ={}".format(t))
print("Rate of Interest={}".format(r))
print("-----------------------------------------")
print("Simple Interest={}".format(si))
print("Total Amount to pay={}".format(totamt))
print("==========================")
```

===

Operators in Python

==

=>An Operator is a symbol , which is used to perform certain Operations.
=>If two or more variables / objects connected with an operator then it is called Expression.
=>In Pytrhon Programming, we have 7 types of Operators. They are

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

Note:- Python Programming does not contain ++ , - - and Ternary Operator (? :)
Note: Python can have Short Hand Operators
(can be prepared with existing operators)

1. Arithmetic Operators

=>The purpose of Arithmetic Operators is that "To Perform Arithmetic Operations such as addition, substract, multiplication..etc"
=>If two or more Variables / objects connected with Arithmetic Operators then It is called Arithmetic Expression.
=>The following Table gives list of Arithmetic Operators

Slno	Symbol	Meaning	Example: a=10 b=3
1	+	Addition	print(a+b)---->13
2	-	Substraction	print(a-b)-----> 7
3.	*	Multiplication	print(a*b)-----> 30
4.	/	Division	print(a/b)----->3.3333333333333335
5	//	Floor Division	print(a//b)-----> 3
6.	%	Modulo Division	print(a%b)---->1
7.	**	Exponentiation	print(a**b) ---->1000

Examples:

```
>>> a=10
>>> b=3
```

```
>>> print(a+b)------------13
>>> print(a-b)-----------7
>>> print(a*b)-----------30
>>> print(a/b)---------------3.3333333333333335
>>> print(a//b)-----------3
>>> print(a%b)----------1
>>> print(a**b)----------1000
>>>
>>>
>>> print(10.0/3.0)----------3.3333333333333335
>>> print(10.0//3.0)----------3.0
>>> print(10.0//3)----------3.0
>>> print(10//3.0)-------------3.0
==============================================

            ==================================
                 2. Assignment Operator
           ==================================
=>The purpose of Assignment Operator is that "To Transfer  Right Hand
Side (RHS)  value / Expression to the VariableLeft Hand Side(LHS)
Variable".

=>We can use assignment operator in two ways. They are
           a) Single Line Assigment
           b) Multi Line Assignment
------------------------------------
a) Single Line Assigment:
------------------------------------
Syntax:-      Varname=Value1
                 (OR)
           Varname=Expression

Examples:
---------------
>>> a=10
>>> b=20
>>> c=a+b
>>> print(a,b,c)--------10 20 30
-------------------------------------------------------------------------
b) Multi Line Assignment:
--------------------------------------
=>With this we can assign multiple RHS Values into LHS Variables.
=>Syntax:-     var1,var2,....var-n=val1,val2....val-n
=>Here val1,val2...val-n are assigned to var1,var2..var-n respectively.

Examples:
-------------
>>> a,b,c=10,20,30
>>> print(a,b,c)---------10 20 30
>>> d=a+b+c
>>> print(d)-----------60
-----------------------------------
>>> a,b=10,3
>>> r1,r2,r3,r4=a+b,a-b,a*b, a**b
>>> print(a,b)------------10 3
>>> print(r1,r2,r3,r4)-----------13 7 30 1000

1.#program for demonstrating Arithmetic Operations
```

```
#aop.py
a=float(input("Enter First Value:"))
b=float(input("Enter Second Value:"))
print("\t---------------------------------------------")
print("\tA r i t h m e t i c O p e r a t i o n s")
print("\t---------------------------------------------")
print("\t\tsum({},{})={}".format(a,b,a+b))
print("\t\tsub({},{})={}".format(a,b,a-b))
print("\t\tmul({},{})={}".format(a,b,a*b))
print("\t\tdiv({},{})={}".format(a,b,a/b))
print("\t\tFloorDiv({},{})={}".format(a,b,a//b))
print("\t\tmod({},{})={}".format(a,b,a%b))
print("\t\texpo({},{})={}".format(a,b,a**b))
print("\t---------------------------------------------")

#mysqrt.py
#This program calculates suare root of any number
n=float(input("Enter any Number:"))
res=n**0.5
print("mysqrt({})={}".format(n,res))

#power.py
a=float(input("Enter Base:"))
m=float(input("Enter Power:"))
res= a**m
print("pow({},{})={}".format(a,m,res))

#This program will interchange the any type of values
#swapex1.py
a=input("Enter Value of a:")
b=input("Enter Value of b:")
#display original Values
print("Original Value of a:{}".format(a))
print("Original Value of b:{}".format(b))
#swap the values
a,b=b,a
#display swapped Values
print("Swapped Value of a:{}".format(a))
print("Swapped Value of b:{}".format(b))
```


```
#This program will interchange the any type of values
#swapex1.py
a=input("Enter Value of a:")
b=input("Enter Value of b:")
#display original Values
print("Original Value of a:{}".format(a))
print("Original Value of b:{}".format(b))
#swap the values
a,b=b,a
#display swapped Values
print("Swapped Value of a:{}".format(a))
print("Swapped Value of b:{}".format(b))
```


===

Relational Operators

===

=>The purpose of relational operators is that " To compare two values ".

=>If two or more Variables / objects connected with Relational Operators then it is
called Relational Expression.
=>Relational Expressions are also called Conditions and they can be evaualted either
to be True or False.
=>The following gives list of relational operators.

SlNo	Symbol	Meaning	Examples a=10 b=20 c=10
1.	>	greater than	a>b-----------False b>c-----------True
2.	<	Less than	a<b-----------True b<c-----------False
3.	==	equality	a==b---------False a==c---------True
4	!=	not equal to	a!=b----------True a!=c----------False
5.	>=	greater than or equal to	a>=b---------False a>=c--------True
6.	<=	Less than or equal to	a<=b------->True b<=c------->False

Logical Operators

=>The purpose of Logical Operators is that " To Combine two or more Relational
Expressions".
=>If two or moreRelational Expressions connected with Logical Operators then it is called Logical Expression / Compound Condition.
=>The result of Logical Expression / Compound Condition is either to be True or False.
=>The following table gives list of Logical Operatos.

SlNo	Symbol	Meaning
1.	or	Physical ORing
2.	and	Physical ANDing
3.	not	--------------------

1) or (Physical ORing)

=>The Functionality of "or" operator is shown in the following truth table.
=>Truth table

RelExpr1	RelExpr2	RelExpr1 or RelExpr2 (Sum Rule)

```
--------------------------------------------------------------------------
----------
      True              False             True

      False       True                    True

      False       False             False

      True              True                    True
--------------------------------------------------------------------------
----------
Examples:
--------------
>>> 10>20 or 20>30-------------False
>>> 10>20 or 20<30----------True
>>> -10<10 or 100>200----------True
-----------------------------------------------------
Short Circuit Evaluation ( in or operator):
------------------------------------------------------
Short Circuit Evaluation in  the case of 'or' is that if First Rel
expression is True then rest of the Relational expressions will not be
evaualted and Total reslut of Logical Expression considered as True.
>>> a,b=10,20
>>> (a<b) or (a>100)-----------True----Short Circuited
>>> (a<b) or (a>100) or (b>100)----True---Short Circuited
>>> (a>b) or (a>100) or (b>100)---False--- not Short Circuited
>>> (a>b) or (a>100) or (b<100)---True---Short Circuited
========================================================
2) and (Physical ANDing):
--------------------------------------------
=>The Functionality of "and" operator is shown in the following truth
table.
=>Truth table
--------------------------------------------------------------------------
--------------------
      RelExpr1          RelExpr2          RelExpr1  and  RelExpr2 (product
Rule)
--------------------------------------------------------------------------
---------------------
      True              False             False

      False       True                    False

      False       False             False

      True              True                    True
--------------------------------------------------------------------------
---------------------
Examples---:
>>> 10>5 and 20>6----True
>>> 10<5 and 20>6----False
>>> 10>2 and 20>6 or 20!=10 or 20==20-----True
>>> 10<2 and 20>6 or 20!=10 or 20==20-----True
>>> 10<2 or 20>6 and 20==10 and 20==20----False

-----------------------------------------------------
Short Circuit Evaluation ( in and operator):
------------------------------------------------------
```

```
Short Circuit Evaluation in  the case of 'and' is that if First Rel
expression is False then rest of the Relational expressions will not be
evaulated and Total reslut of Logical Expression considered as False.
=============================================================
3) not operator:
----------------------
=>This operator obtains the opposite result of existing result of
Relational Expressions and logical Expressions.
=>The Functionality of 'not' operator is shown bellow.
---------------------------------------------------------------------------
--------------------
      RelExpr1         not RelExpr1
---------------------------------------------------------------------------
---------------------
      True             False

      False      True
---------------------------------------------------------------------------
---------------------
Examples:
----------------
>>> a=100
>>> b=200
>>> print(a>b)----------False
>>> print(not (a>b) )-----True
>>> print(not (a<b) )-----False
>>> 10>5 and 10==10------True
>>> not ( 10>5 and 10==10)------False
>>> not ( 10!=5 or 10!=10)------False
---------------------------------------------------------------------------
--
NOTE : Logical operators without realtional operators. (Most Important)

Operator    Example       Result
---------------------------------------------------------
   and        x and y       if x is False , It returns x otherwise it
return y
    or            x or y         if x  is False , It returns y otehrwise
it returns x
---------------------------------------------------------
>>> 100 and 200-----------200
>>> -101 and 300---------300
>>> 0 and 234------------0
>>> 100 and 0-----------0
>>> "KVR" and "PYTHON"-------'PYTHON'
>>> 0 and "PYTHON"---------0
------------------------------------------------------------------
>>> 100 or 200------------100
>>> 0 or 200--------200
>>> 0 or 0-----------------0
>>> 100 and "KVR"---------'KVR'
========================================
---------------------------------------------------------------------------
------------------
My requirement:
-----------------------
Company  want select u people by knowing either skill1 or skill2
```

```
            skill1:   Python
            skill2 :  Oracle

      (skill1=="Python") or (skill2=="Oracle")
Company  want select u people by knowing both the  skils

      (skill1=="Python") and (skill2=="Oracle")
```

```
1.#Program for demonstrating Relational Operators
#rop.py
a=int(input("Enter Value of a:"))
b=int(input("Enter Value of b:"))
c=int(input("Enter Value of c:"))
print("---------------------------------------------------")
print("\tResult of Relational Operators")
print("---------------------------------------------------")
print("\t\t{} > {}={}".format(a,b,a>b))
print("\t\t{} > {}={}".format(b,a,b>a))
print("\t\t{} < {}={}".format(a,c,a<c))
print("\t\t{} < {}={}".format(c,a,c<a))
print("\t\t{} == {}={}".format(a,b,a==b))
print("\t\t{} != {}={}".format(a,b,a!=b))
print("\t\t{} >= {}={}".format(b,c,b>=c))
print("\t\t{} <= {}={}".format(b,c,b<=c))
print("---------------------------------------------------")
```

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

Bitwise Operators (Most Imp)

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

=>In Python Programming, Bitwise Operators are used for performing Bitwise Calculations on Integer data but not on float data(bcoz number of decimal places internally varying).
=>The Bitwise operators are internally converting Integer data into Binary Format and performs the calculation bit by bit and hence they named as Bitwise Operators.
=>The result of Bitwise operators gives in the form of Decimal format.
=>In Python Programming, we have 6 bitwise operators. They are

1. Bitwise Left Shift operator (<<)
2. Bitwise Right Shift operator (>>)
3. Bitwise OR Operator (|)
4. Bitwise AND Operator (&)
5. Bitwise Complement Operator (~)
6. Bitwise XOR Operator (^)

----

1. Bitwise Left Shift operator (<<):

Syntax:-   varname = GivenData << No. of Bits

Explanation:

=>The Functionality of Bitwise Left Shift operator is that It shifts the Specified "no.of bits" towards left side and fills no. of zero (no. of bits) at right side .

Examples:

```
---------------
>>> a=10
>>> res=a<<3
>>> print(res)----------80
>>> print(50<<2)-------200
---------------------------------------------------------------------------
-----------------
2. Bitwise Right Shift operator ( >> ):
---------------------------------------------------------------------
Syntax:-         varname = GivenData >> No. of Bits
----------------
Explanation:
----------------
=>The Functionality of Bitwise Right Shift operator is that It shifts the
Specified "no.of bits" towards Right Side and fills no. of zero (no. of
bits)  at Left Side .
Examples:
--------------------
>>> print(10>>3)---1
>>> print(10>>2)---2
---------------------------------------------------------------------------
-----
3. Bitwise OR Operator ( | ):
---------------------------------------
=>The Functionality of Bitwise OR Operator ( | ) is shown in the
following truth table.
-------------------------------------------------
      P           Q           P | Q
-------------------------------------------------
        0              1            1
      1           0             1
      0           0             0
      1           1             1
-------------------------------------------------
Example-1:
----------------
>>>a=4---------------> 0 1 0 0
>>>b=5---------------> 0 1 0 1
--------------------------------------------
>>>c=a|b------------->0 1 0 1------> Result is 5
>>>print(c)----------5
------------------------------------
>>> print(10|15)-------15
>>> print(3|4)----------7
--------------------------------------------
Special Case:
-------------------
>>>s1={10,30,20}
>>>s2={20,10,40,50}
>>>s3=s1.union(s2)
>>>print(s3)
>>>s4=s1|s2
>>> print(s4)---------{50, 20, 40, 10, 30}
>>> s1={"Apple","Mango"}
>>> s2={"Mango","Kiwi"}
>>> s3=s1|s2
>>> print(s3)-----------{'Mango', 'Kiwi', 'Apple'}
---------------------------------------
```

```
--------------------------------------------------------------------------
----------
4. Bitwise AND Operator ( & ):
--------------------------------------------
=>The Functionality of Bitwise AND Operator ( & ) is shown in the
following truth table.
-------------------------------------------------
      P          Q          P & Q
-------------------------------------------------
        0              1          0
      1          0          0
      0          0          0
      1          1          1
-------------------------------------------------
Examples:
----------------
>>>a=4------------------ 0 1 0 0
>>>b=5------------------ 0 1 0 1
-------------               --------------
>>>c=a&b-------------   0 1 0 0-------> Result---> 4
>>>print(c)----------- 4
>>> print(5&15)-----------5
>>> print(4&3)----------0
-----------------------------------------------
Special Case(Intersection)
--------------------
>>> s1={10,20,30}
>>> s2={20,40,50}
>>> s3=s1.intersection(s2)
>>> print(s3)-------{20}
>>>
>>>
>>> s1={10,20,30}
>>> s2={20,40,50}
>>> s3=s1&s2
>>> print(s3)---------{20}
===============================================
5) Bitwise Complement Operator ( ~ ):
---------------------------------------------------
=>Syntax:    varname= ~Value / Variable

=>This operator is used for obtaining complement of Given value.

Formula:-     ~n =  -(n+1)
------------
---------------
Examples:
---------------
>>>a=10
>>>res= ~a ---------------->  ~(1010+1)
             ---------------->  - (1010+1)
                      ---------->    -  1010
                                       0001
                            ---------------------
                                 -   1011---->result is  -11
                            ------------------------
```

```
>>> a=10
>>> print( ~a)---------   -11
>>> a=100
>>> res=~a
>>> print(res)---------  -101
>>> a = -108
>>> result=~a
>>> print(result)-----------  107
==============================================================
6. Bitwise XOR Operator ( ^ ):
----------------------------------------
=>The Functionality of Bitwise XOR Operator ( ^ ) is shown in the
following truth table.
-------------------------------------------------
      P          Q          P ^ Q
-------------------------------------------------
        0              1             1
      1          0             1
      0          0             0
      1          1             0
-------------------------------------------------
Examples:
---------------
>>>a=4------------------ 0 1 0 0
>>>b=5------------------ 0 1 0 1
-------------                ---------------
>>>c=a^b ---------->   0  0 0 1----- Result is 1
>>>print(c)----> 1
>>> print(10^4)----14
>>> print(5^3)------6
--------------------------------------------------------------------------
---------------------
Special Case (Symmetric_difference)
-------------------------------------------------------------------
>>> s1={10,20,30}
>>> s2={20,40,50}
>>> s3=s1.symmetric_difference(s2)
>>> print(s3)-----------{40, 10, 50, 30}
>>> s1={10,20,30}
>>> s2={20,40,50}
>>> s3=s1^s2
>>> print(s3)----------{40, 10, 50, 30}
============================X=============================

--------------------------------------------------------------------------
----------
4. Bitwise AND Operator ( & ):
--------------------------------------------
=>The Functionality of Bitwise AND Operator ( & ) is shown in the
following truth table.
-------------------------------------------------
      P          Q          P & Q
-------------------------------------------------
        0              1          0
      1          0          0
      0          0          0
      1          1          1
-------------------------------------------------
```

```
Examples:
-----------------
>>>a=4------------------ 0 1 0 0
>>>b=5------------------ 0 1 0 1
-------------                 --------------
>>>c=a&b-------------   0 1 0 0-------> Result---> 4
>>>print(c)----------- 4
>>> print(5&15)-----------5
>>> print(4&3)----------0
------------------------------------------------
Special Case(Intersection)
--------------------
>>> s1={10,20,30}
>>> s2={20,40,50}
>>> s3=s1.intersection(s2)
>>> print(s3)-------{20}
>>>
>>>
>>> s1={10,20,30}
>>> s2={20,40,50}
>>> s3=s1&s2
>>> print(s3)---------{20}
================================================
5) Bitwise Complement Operator ( ~ ):
----------------------------------------------------
=>Syntax:    varname= ~Value / Variable

=>This operator is used for obtaining complement of Given value.

Formula:-     ~n =  -(n+1)
------------
---------------
Examples:
---------------
>>>a=10
>>>res= ~a ---------------->  ~(1010+1)
             ---------------->  - (1010+1)
                      ---------->    -  1010
                                          0001
                             ----------------------
                                  -   1011---->result is  -11
                             ------------------------

>>> a=10
>>> print( ~a)---------   -11
>>> a=100
>>> res=~a
>>> print(res)---------  -101
>>> a = -108
>>> result=~a
>>> print(result)-----------  107
==============================================================
6. Bitwise XOR Operator ( ^ ):
----------------------------------------
=>The Functionality of Bitwise XOR Operator ( ^ ) is shown in the
following truth table.
--------------------------------------------------
      P          Q           P ^ Q
```

```
-------------------------------------------------
        0              1             1
      1          0             1
      0          0             0
      1          1             0
-------------------------------------------------
Examples:
---------------
>>>a=4------------------ 0 1 0 0
>>>b=5------------------ 0 1 0 1
-------------              --------------
>>>c=a^b ---------->   0  0 0 1----- Result is 1
>>>print(c)----> 1
>>> print(10^4)----14
>>> print(5^3)------6
-------------------------------------------------------------------------
---------------------
Special Case (Symmetric_difference)
------------------------------------------------------------------
>>> s1={10,20,30}
>>> s2={20,40,50}
>>> s3=s1.symmetric_difference(s2)
>>> print(s3)-----------{40, 10, 50, 30}
>>> s1={10,20,30}
>>> s2={20,40,50}
>>> s3=s1^s2
>>> print(s3)----------{40, 10, 50, 30}
============================X=============================

            =====================================
                        Identity Operators
           =====================================
=>The main purpose of  Identity Operators is that " To compare the memory
address of     two objects / Variables".
=>We have two types of  Identity Operators. They are
           a) is
           b) is not
----------
a) is :
----------
Syntax:      object1  is  object2
                   (OR)
           Variable1  is  Variable2
=>"is" operator returns True provided Both the object1 and Object2 points
to same memory address (or) "is" operator returns True provided Both the
object1 and Object2 contains same memory  address .
=>"is" operator returns False provided Both the Object1 and Object2
points to Different memory addresses (or) "is" operator returns False
provided Both the object1 and Object2 contains Different  memory  address
.
-------------------------------------------------------------------
b) is not  :
---------------
Syntax:      object1  is not  object2
                   (OR)
           Variable1  is not Variable2
=>"is not" operator returns True provided Both the object1 and Object2
points to different  memory address (or) "is not " operator returns True
```

provided Both the Object1 and Object2 contains different memory address .
=>"is not " operator returns False provided Both the Object1 and Object2 points to Same memory address (or) "is not" operator returns False provided Both the object1 and Object2 contains Same memory address .
===
Examples:

```
>>> a=None
>>> b=None
>>> print(a,type(a), id(a))---None <class 'NoneType'> 140712191690744
>>> print(b,type(b), id(b))--None <class 'NoneType'> 140712191690744
>>> a is b---True
>>> a is not b---False
-----------------------------------------------------
>>> d1={10:"Apple",20:"Mango"}
>>> d2={10:"Apple",20:"Mango"}
>>> print(d1,type(d1),id(d1))--{10: 'Apple', 20: 'Mango'} <class 'dict'>
2326716104896
>>> print(d2,type(d2),id(d2))--{10: 'Apple', 20: 'Mango'} <class 'dict'>
2326716105024
>>> d1 is d2---False
>>> d1 is not d2---True
---------------------------------------------------------------------
>>> s1={10,20,30,40,50}
>>> s2={10,20,30,40,50}
>>> print(s1,type(s1),id(s1))---{50, 20, 40, 10, 30} <class 'set'>
2326716327616
>>> print(s2,type(s2),id(s2))---{50, 20, 40, 10, 30} <class 'set'>
2326716327840
>>> s1 is s2-----False
>>> s1 is not s2----True
>>> fs1=frozenset(s1)
>>> fs2=frozenset(s2)
>>> print(fs1,type(fs1),id(fs1))--frozenset({50, 20, 40, 10, 30}) <class
'frozenset'>                                                2326716327168
>>> print(fs2,type(fs2),id(fs2))---frozenset({50, 20, 40, 10, 30}) <class
'frozenset'>                                                2326716329408
>>> fs1 is fs2------False
>>> fs1 is not  fs2----True
--------------------------------------------------------------------------
-----
>>> t1=(10,"Python",34.56)
>>> t2=(10,"Python",34.56)
>>> print(t1,type(t1),id(t1))---(10, 'Python', 34.56) <class 'tuple'>
2326716160768
>>> print(t2,type(t2),id(t2))--(10, 'Python', 34.56) <class 'tuple'>
2326716162752
>>> t1 is t2---False
>>> t1 is not t2---True
>>> l1=[12,"Rossum"]
>>> l2=[12,"Rossum"]
>>> print(l1,type(l1),id(l1))---[12, 'Rossum'] <class 'list'>
2326716119616
>>> print(l2,type(l2),id(l2))---[12, 'Rossum'] <class 'list'>
2326716166080
>>> l1 is l2---False
>>> l1 is not l2---True
```

```
--------------------------------------------------------------
>>> r1=range(0,10)
>>> r2=range(0,10)
>>> print(r1,type(r1),id(r1))---range(0, 10) <class 'range'>
2326716275696
>>> print(r2,type(r2),id(r2))---range(0, 10) <class 'range'>
2326716276032
>>> r1 is r2----False
>>> r1 is not r2---True
----------------------------------------------------------------------
>>> t1=(10,20,30,255)
>>> ba1=bytearray(t1)
>>> ba2=bytearray(t1)
>>> print(ba1,type(ba1),id(ba1))
bytearray(b'\n\x14\x1e\xff') <class 'bytearray'> 2326716418544
>>> print(ba2,type(ba2),id(ba2))
bytearray(b'\n\x14\x1e\xff') <class 'bytearray'> 2326716418672
>>> ba1 is  ba2--------False
>>> ba1 is not  ba2-----True
>>> t1=(10,20,30,255)
>>> b1=bytes(t1)
>>> b2=bytes(t1)
>>> print(b1, type(b1),id(b1))---b'\n\x14\x1e\xff' <class 'bytes'>
2326716275984
>>> print(b2, type(b2),id(b2))---b'\n\x14\x1e\xff' <class 'bytes'>
2326716275264
>>> b1 is b2--------False
>>> b1 is not  b2------True
---------------------------------------------------------------
>>> s1="PYTHON"
>>> s2="PYTHON"
>>> print(s1,type(s1),id(s1))----PYTHON <class 'str'> 2326716418608
>>> print(s2,type(s2),id(s2))---PYTHON <class 'str'> 2326716418608
>>> s3="python"----
>>> print(s3,type(s3),id(s3))---python <class 'str'> 2326716418992
>>> s1 is s2----True
>>> s1 is not s2---False
>>> s1 is not s3---True
>>> s1 is  s3----False
>>> s1="""Python is an oop lang
... python is also Fun Prog lang"""
>>> s2="""Python is an oop lang
... python is also Fun Prog lang"""
>>> print(s1,id(s1))----Python is an oop lang
                         python is also Fun Prog lang 2326710819328
>>> print(s2,id(s2))---Python is an oop lang
                         python is also Fun Prog lang 2326715975584
>>> s1 is s2-------False
>>> s1 is not s2----True
--------------------------------------------------------------------------
---
>>> a=2+3j
>>> b=2+3j
>>> print(a, type(a), id(a))------(2+3j) <class 'complex'> 2326711621712
>>> print(b, type(b), id(b))-----(2+3j) <class 'complex'> 2326711621840
>>> a is b---------False
>>> a is not b----True
```

```
--------------------------------------------------------------------------
-------
>>> b1=True
>>> b2=True
>>> print(b1,type(b1),id(b1))----True <class 'bool'> 140712191638376
>>> print(b2,type(b2),id(b2))---True <class 'bool'> 140712191638376
>>> b1 is b2----True
>>> b1 is not b2---False
-------------------------------------------------------------------------
>>> a=3.4
>>> b=3.4
>>> print(a,type(a), id(a))-------3.4 <class 'float'> 2326711618160
>>> print(b,type(b), id(b))-------3.4 <class 'float'> 2326711621872
>>> a is b-------False
>>> a is not  b-------True
--------------------------------------------------------------
>>> a=100
>>> b=100
>>> print(a,type(a),id(a))
100 <class 'int'> 2326710586704
>>> print(b,type(b),id(b))
100 <class 'int'> 2326710586704
>>> a is b
True
>>> a is not b
False
>>>
>>> a=256
>>> b=256
>>> print(b,type(b),id(b))------256 <class 'int'> 2326710591696
>>> print(b,type(b),id(b))------256 <class 'int'> 2326710591696
>>> a is b--------True
>>> a is not b----False
>>> a=257
>>> b=257
>>> print(a,type(a),id(a))----257 <class 'int'> 2326711621840
>>> print(b,type(b),id(b))---257 <class 'int'> 2326711621552
>>> a is b----False
>>> a is not b----True
>>> a=-2
>>> b=-2
>>> print(a,type(a),id(a))----   -2 <class 'int'> 2326710583440
>>> print(b,type(b),id(b))--   -2 <class 'int'> 2326710583440
>>> a is b----True
>>> a is not b---False
>>> a=-5
>>> b=-5
>>> print(a,type(a),id(a))--- -5 <class 'int'> 2326710583344
>>> print(b,type(b),id(b))--- -5 <class 'int'> 2326710583344
>>> a is b-----True
>>> a is not b---False
>>> a=-6
>>> b=-6
>>> print(a,type(a),id(a))---   -6 <class 'int'> 2326710598800
>>> print(b,type(b),id(b))---- -6 <class 'int'> 2326711621552
>>> a is b---- False
>>> a is not b--- True
----------------------------------------------------------------------
```

=======================================
Flow Control Statements in Python
(OR)
Control Structures in Python
=======================================

=>The Purpose of Flow Control Statements in Python is that "To perform certain Operation Only Once when the Condition is True or False (OR) To perform certain Operation Repeatedly for finite number of times until condition becomes False".

=>In Python Programming, we have 3 types of Flow Control Statements. They are

1) Conditional / Selection / Branching Statements
2) Looping / Iterative / Repetative Statements
3) Misc. Flow Control Statements in Python (break, continue, pass)

==
1) Conditional / Selection / Branching Statements
==

=>The purpose of Conditional / Selection / Branching Statements is that "To Perform Certain Operation (X-Operation) when the condition is True or Perform another operation (Y-Operation) when the Condition is False Only Once".

(OR)

=>The purpose of Conditional / Selection / Branching Statements is that "To Perform X-Operation when the condition is True or Perform Y-Operation when the condtion is False" only once.

=>In Python Programming, we have 4 types of Conditional / Selection / Branching Statements. They are

i) Simple if statement
ii) if..else statement (nested / inner if..else...)
iii) if..elif..else statement
iv) match..case (Python 3.10 Version onwords only)

#Program for demonstrating simple if statement

```
#funny.py
tkt=input("Do u have ticket:")
if(tkt=="yes"):
      print("Enter into Threater")
      print("Watch Movieeeeeeeeeeeee")
      print("Enjoy the moviee")

print("\nGoto Home and Open Python Notes")
```

```
#Program for deciding weather the given number is +ve or -Ve (or ) zero
#posnegzero.py
n=int(input("Enter a Number:"))  # n=0
if (n>0):
      print("{} is +VE".format(n))
if(n<0):
      print("{} is -VE".format(n))
if(n==0):
      print("{} is ZERO".format(n))

print("Program execution completed")
```

```
#Program accepting two numerical value and decide the biggest among them
and check for equality.
#big.py
a=int(input("Enter Value of a:")) # a=10
b=int(input("Enter Value of b:")) # b=20
if(a==b):
	print("Both the values are Equal")
else:
	if(a>b):
		print("biggest={}".format(a))
	else:
		print("biggest={}".format(b))

#program for accpeting any digit and print its name
#digitex1.py
d=int(input("Enter any digit:"))
if(d==0):
	print("{} is ZERO".format(d))
else:
	if(d==1):
		print("{} is ONE".format(d))
	else:
		if(d==2):
			print("{} is TWO".format(d))
		else:
			if(d==3):
				print("{} is THREE".format(d))
			else:
				if(d==4):
					print("{} is FOUR".format(d))
				else:
					if(d==5):
						print("{} is FIVE".format(d))
					else:
						if(d==6):
							print("{} is SIX".format(d))
						else:
							if(d==7):
								print("{} is
SEVEN".format(d))
							else:
								if(d==8):
									print("{} is
EIGHT".format(d))
								else:
									if(d==9):
										print("{} is
NINE".format(d))
									else:
										print("It is a
Number:")

#program for for deciding whether the input is digit or number
#digitex2.py
d=int(input("Enter any digit:"))
if d in range(0,10):
	print("{} is a digit".format(d))
```

```
else:
	print("{} is a number".format(d))

#Program for generating pay slip of an employee
#payslip.py
empno=int(input("Enter Employee Number:"))
ename=input("Enter Employee Name:")
basicsal=float(input("Enter Basic Salary:"))  # 10000    -10000 / 0
9000
#check basicsal
if(basicsal<=0):
	print("Invalid Salary:")
else:
	if(basicsal>=10000):
		da=basicsal*(20/100)
		ta=basicsal*(15/100)
		hra=basicsal*(10/100)
		ma=basicsal*(5/100)
		gpf=(2/100)*basicsal
		lic=(3/100)*basicsal
	else:
		da=basicsal*(25/100)
		ta=basicsal*(20/100)
		hra=basicsal*(15/100)
		ma=basicsal*(6/100)
		gpf=(2/100)*basicsal
		lic=(3/100)*basicsal
	netsal = (basicsal+da+ta+hra+ma)-(gpf+lic)
	#Display Pay slip details
	print("========================================")
	print("\tEmployee Pay Slip Information:")
	print("========================================")
	print("\tEmployee Number:{}".format(empno))
	print("\tEmployee Name:{}".format(ename))
	print("\tEmployee Basic Salary:{}".format(basicsal))
	print("\tEmployee DA:{}".format(da))
	print("\tEmployee TA:{}".format(ta))
	print("\tEmployee HRA:{}".format(hra))
	print("\tEmployee MA:{}".format(ma))
	print("\tEmployee GPF:{}".format(gpf))
	print("\tEmployee LIC:{}".format(lic))
	print("---------------------------------------------")
	print("\tNet Salary of Employee:{}".format(netsal))
	print("=======================================")

#Program accepting a numerical value and decide weather it is +ve or -ve
or zero
#posneqzero1.py
n=float(input("Enter a number:"))
if(n>0):  # cond-1
	print("{} is POSSITIVE".format(n))
else:
	if(n<0): # cond-2
		print("{} is NEGATIVE".format(n))
	else:
		print("{} isZERO".format(n))

#Program for finding biggest of three numbers
```

```
#bigex1.py
a=int(input("Enter First Value:"))
b=int(input("Enter Second Value:"))
c=int(input("Enter Third Value:"))
if ( (a>b) and (a>c) ):
      print("biggest({},{},{})={}".format(a,b,c,a))
elif( (b>a) and(b>c)) :
      print("biggest({},{},{})={}".format(a,b,c,b))
elif((c>a) and (c>b)):
      print("biggest({},{},{})={}".format(a,b,c,c))
elif((a==b) and (b==c)):
      print("ALL VALUES ARE EQUAL")

#program for accpeting any digit and print its name
#digitex3.py
d=int(input("Enter any digit:"))
if(d==0):
      print("{} is ZERO".format(d))
elif(d==1):
      print("{} is ONE".format(d))
elif(d==2):
      print("{} is TWO".format(d))
elif(d==3):
      print("{} is THREE".format(d))
elif(d==5):
      print("{} is FIVE".format(d))
elif(d==4):
      print("{} is FOUR".format(d))
elif(d==6):
      print("{} is SIX".format(d))
elif(d==7):
      print("{} is SEVEN".format(d))
elif(d==8):
      print("{} is EIGHT".format(d))
elif(d==9):
      print("{} is NINE".format(d))
else:
      print("{} is NUMBER".format(d))
```

==

match case statement

==

=>It is one of Facility in Python 3.10 Version onwords .

=>The purpose of match...case statements is that " To handle menu driven applications/ pre-designed conditions ".

=>Syntax:

```
match (Choice Expression):
      case label1:
               Block of statements-I
      case label2:
               Block of statements-II
      --------------------
      --------------------
      case label-n:
               Block of statements-N
      case _:
```

            defaule Block of statements
----
other statemenets in program

==================
Explanation:
==================
=>here 'match' , 'case' are the keywords
=>Here "Choice Expression" can be either int or str or bool type.
=>If the value of Choice Expression is match with case label1 then execute
    curresponding block of statements and later execxute other statements in Program.
=>In General, If the value of Choice Expression is matching with any specified case labels then execute curresponding block of statements and later execxute other statements in Program.
=>If the value of Choice Expression is not matching with any specified case labels then PVM executes default case block of statements which are written under case_ (known as defdault case block) and later execxute other statements in Program. . Writing default case block is optional.
===========================X====================================

```
#program implementing menu driven application for computing all Arithmetic Operations by using match  ... case
#aop.py
print("=====================================================")
print("\tA r i t h m e t i c  O p e r a t i o n s")
print("=====================================================")
print("\t1.Addition")
print("\t2.Substraction")
print("\t3.Multiplication")
print("\t4.Division")
print("\t5.Floor Division")
print("\t6.Modulo Division")
print("\t7.Exponentiation")
print("\t8.Exit")
print("=====================================================")
ch=int(input("Enter Ur Choice:"))
match(ch):
      case 1:
            n1=float(input("Enter First Value for Addition:"))
            n2=float(input("Enter Second Value for Addition:"))
            print("sum({},{})={}".format(n1,n2,n1+n2))
      case 2:
            print("Enter Two values for substraction:")
            n1,n2=float(input()), float(input())
            print("sub({},{})={}".format(n1,n2,n1-n2))
      case 3:
            print("Enter Two values for Multiplication:")
            n1,n2=float(input()), float(input())
            print("mul({},{})={}".format(n1,n2,n1*n2))
      case 4:
            print("Enter Two values for Division:")
            n1,n2=float(input()), float(input())
            print("Div({},{})={}".format(n1,n2,n1/n2))
      case 5:
            print("Enter Two values for Floor Div:")
            n1,n2=float(input()), float(input())
```

```
            print("Floor Div({},{})={}".format(n1,n2,n1//n2))
        case 6:
            print("Enter Two values for Modulo Divisioin:")
            n1,n2=float(input()), float(input())
            print("Mod({},{})={}".format(n1,n2,n1%n2))
        case 7:
            print("Enter Two values for Exponentiation:")
            n1,n2=float(input()), float(input())
            print("pow({},{})={}".format(n1,n2,n1**n2))
        case 8:
            print("\nThanks for This program")
            exit() # pre-defined function used to terminate the program
physically.
        case _:
            print("Ur Selection of Operation is wrong!--execute again")
```

```
#Program for demonstrating match case
#week.py
wkname=input("Enter the week name:")
match(wkname):
        case "SUNDAY":
            print("{} is a Holiday:".format(wkname))
        case "MONDAY":
            print("{} is working Day:".format(wkname))
        case "TUESDAY":
            print("{} is working Day:".format(wkname))
        case "WEDNESDAY":
            print("{} is working Day:".format(wkname))
        case "THURSDAY":
            print("{} is working Day:".format(wkname))
        case "FRIDAY":
            print("{} is working Day:".format(wkname))
        case "SATURDAY":
            print("{} is Week End:".format(wkname))
        case _:
            print("{} is not week day:".format(wkname))
```

```
#Program for demonstrating match case
#week1.py
wkname=input("Enter the week name:")
match(wkname):
        case "SUNDAY" | "sunday":
            print("{} is a Holiday:".format(wkname))
        case "MONDAY" | "monday":
            print("{} is working Day:".format(wkname))
        case "TUESDAY"| "Tuesday":
            print("{} is working Day:".format(wkname))
        case "WEDNESDAY"| 'wednesday':
            print("{} is working Day:".format(wkname))
        case "THURSDAY" | 'thursday':
            print("{} is working Day:".format(wkname))
        case "FRIDAY" | "friday":
            print("{} is working Day:".format(wkname))
        case "SATURDAY" | "saturday":
            print("{} is Week End:".format(wkname))
        case _:
            print("{} is not week day:".format(wkname))
```

```
#Program for demonstrating match case
#week2.py
wkname=input("Enter the week name:")
match(wkname):
      case "SUNDAY" | "sunday":
            print("{} is a Holiday:".format(wkname))
      case "MONDAY" | "monday" | "TUESDAY"| "tuesday" |"WEDNESDAY"|
'wednesday'| "THURSDAY" | 'thursday'| "FRIDAY" | "friday":
            print("{} is working Day:".format(wkname))
      case "SATURDAY" | "saturday":
            print("{} is Week End:".format(wkname))
      case _:
            print("{} is not week day:".format(wkname))

#Program for demonstrating match case
#week3.py
wkname=input("Enter the week name:")
match(wkname.lower()):
      case "SUNDAY" | "sunday":
            print("{} is a Holiday:".format(wkname))
      case "MONDAY" | "monday" | "TUESDAY"| "tuesday" |"WEDNESDAY"|
'wednesday'| "THURSDAY" | 'thursday'| "FRIDAY" | "friday":
            print("{} is working Day:".format(wkname))
      case "SATURDAY" | "saturday":
            print("{} is Week End:".format(wkname))
      case _:
            print("{} is not week day:".format(wkname))
```

```
            ==========================================
                 2) Looping (or) Iterative (or) Repetative Statements
            ==========================================
=>The purpose of Looping statements is that "To perform certain Operation
Repetedly
     for finite number of times until condition becomes False.
=>In Python Programming , we have 2 types of Looping Statements. They
are.

                  i) while    (or)  while...else
                  ii) for      (or)    for...else

=>While we are dealing with Looping statements , programmer must ensure
there must exists 3 points. They are
                        i) Initlization Part
                        ii) Conditional Part
                        iii) Updation Part

             ====================================
                  i) while    (or)  while...else
            ====================================
Syntax1:
-------------
       -------------------
       -------------------
       while ( Test Cond ):
              -------------------------
              Block of Statements
            -------------------------
      ------------------------------------
```

```
      Other statements in Program
      ------------------------------------
================================================
Syntax-2:
-------------
       -------------------
       -------------------
       while ( Test Cond ):
              -------------------------
              Block of Statements
            -------------------------
      else:
            Else Block of Statements
      ------------------------------------
      Other statements in Program
      ------------------------------------
```

Explanation:

=>here 'while' and 'else' are the key words
=>Here Test cond will Evaluated First
=>If test condition is true then PVM will execute Block of statements and once again test condition will be evaluated. If Test condtion is once again True then PVM again executes Block of statements. Hence Block of statements will executed continuously as long as test cond is True.
=>Once Test cond is False then PVM execute else block of statements, which are written inside of else block and later other statements in the program will execute.
=>Writting else block of statements is Optional.

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

#Program for finding sum of 'n' natural numbers.
#NatNums.py
n=int(input("Enter How many natural Numbers sum u want to find:"))
if(n<=0):
      print("{} is invalid input".format(n))
else:
      i,s=1,0  #multiline assignment statement
      print("----------------------------------------------")
      print("Natural Numbers within:%d" %n)
      print("----------------------------------------------")
      while(i<=n):
            print("\t{}".format(i))
```

```
            s=s+i # Keeps Track of Sum of Individual Digits of Natural 
Numbers
            i=i+1
      else:
            print("------------------------------------")
            print("\tSum=%d" %s)
            print("------------------------------------")

#Program for generating 1 to N numbers where N is +ve
#NumGenEx1.py
n=int(input("Enter How Many Number u want to generate:")) # n = 10  or -
10  or 0
if (n<=0):
      print("{} is invalid input:".format(n))
else:
            print("---------------------------------")
            print("Numbers within:{}".format(n))
            print("---------------------------------")
            i=1  # Initlization part
            while(i<=n):    # conditional  part
                  print("\t{}".format(i))
                  i=i+1   # Updation Part
            else:
                  print("---------------------------------")
                  print("I am from else block")
                  print("---------------------------------")

#Program for generating 1 to N numbers where N is +ve
#NumGenEx2.py
n=int(input("Enter How Many Number u want to generate:")) # n = 10  or -
10  or 0
if (n<=0):
      print("{} is invalid input:".format(n))
else:
            print("---------------------------------")
            print("Numbers within:{}".format(n))
            print("---------------------------------")
            i=1  # Initlization part
            while(i<=n):    # conditional  part
                  print("{}".format(i), end=" ")
                  i=i+1   # Updation Part
            else:
                  print("\n---------------------------------")
                  print("I am from else block")
                  print("---------------------------------")

#Program for generating N to 1numbers where N is +ve
#NumGenEx3.py
n=int(input("Enter How Many Numbers u want to generate:"))
if(n<=0):
      print("{} is invalid input:".format(n))
else:
      print("-----------------------------------------------------------
---")
      print("Numbers within n in reverse order:{}".format(n))
      print("-----------------------------------------------------------
---")
      while(n>0): # cond part
```

```
            print("\t{}".format(n))
            n=n-1  #updation part
      else:
            print("-----------------------------------------------------------
---------")

#Program for generating N to 1numbers where N is +ve
#NumGenEx4.py
n=int(input("Enter How Many Numbers u want to generate:"))
if(n<=0):
      print("{} is invalid input:".format(n))
else:
      print("-----------------------------------------------------------------
---")
      print("Numbers within n in reverse order:{}".format(n))
      print("-----------------------------------------------------------------
---")
      i=n
      while(i>0): # cond part
            print("\t{}".format(i))
            i=i-1  #updation part
      else:
            print("-----------------------------------------------------------
---------")
```

```
            ================================================
                        ii) for      (or)     for  else
            ================================================
Syntax-1:
-----------
     for  varname  in  Iterable-object:
            ---------------------------
            Block of statements
            ---------------------------
      -------------------------------------
      Other statements in Program
      --------------------------------------
------------
Syntax-2:
------------
     for  varname  in  Iterable-object:
            ---------------------------
            Block of statements
            ---------------------------
     else:
          else block of statements
          --------------------------------
      -------------------------------------
      Other statements in Program
      --------------------------------------

------------------
Explanation:
-------------------
=>here 'for' , 'in'  are a keywords
=>Here Iterable_object is one, which contains Multiple elements .
       Examples:   Sequnece Type, List Type, set type and dict type
```

=>The Execution behaviour of for loop is that Each element of Iterbale_object is selected, placed in varname and executes Block of statemets. Hence the block of statements are executed repeatedly for finite number of times until all elements are completed in Iterable_object.

```
#listex1.py
lst=[10,20,"Python","Java",True,34.56]  # here lst is called Iterable
object
print("By using for Loop")
print("---------------------------------")
for kvr in lst:
      print("\t{}".format(kvr))
else:
      print("\n---------------------------------")
print("By using while Loop")
print("---------------------------------")
i=0
while(i<len(lst)):
      print("\t{}".format(lst[i]))
      i=i+1
print("---------------------------------")

#Program for reading list of values and display them
#listvalues.py
n=int(input("Enter How Many Value u have:"))  # n= 5
if(n<=0):
      print("{} is invalid input:".format(n))
else:
      lst=list()  # create an empty list
      for i in range(1,n+1):
            val=input("Enter {} Value:".format(i))
            lst.append(val)
      else:
            print("----------------------------------------")
            print("List of elements={}".format(lst))
            print("----------------------------------------")

#program for generating multable for a given number using for loop
#multable1.py
n=int(input("Enter a number:"))
if(n<=0):
      print("{} is invalid input".format(n))
else:
      print("-------------------------------------")
      print("Mul table for {}".format(n))
      print("-------------------------------------")
      for i in range(1,11):
            print("\t{} x {} = {}".format(n,i,n*i))
      else:
            print("--------------------------------------")

#sequecetypeex1.py
s="PYTHON PROGRAMMING"  # here s is called Iterable object
print("By using while Loop")
print("---------------------------------")
i=0
```

```
while(i<len(s)):
      print("{}".format(s[i]), end="  ")
      i=i+1
else:
      print("\n---------------------------------")

#sequecetypeex2.py
s="PYTHON PROGRAMMING"  # here s is called Iterable object
print("By using for Loop")
print("---------------------------------")
for x in s:
      print("\t{}".format(x))
else:
      print("\n---------------------------------")

#Program for reading list of values and find sum and average
#sumavg.py
n=int(input("Enter How Many Values u have:"))
if(n<=0):
      print("{} is invalid input".format(n))
else:
      #reading the values dynamically
      lst=[]  #empty list
      for i in range(1,n+1):
            val=float(input("Enter {} Value:".format(i)))
            lst.append(val)
      else:
            print("----------------------------------------")
            print("Content of lst:{}".format(lst))
            print("----------------------------------------")
            #find sum and average
            s=0
            for val in lst:
                  print("\t{}".format(val))
                  s=s+val
            else:
                  print("----------------------------------------")
                  print("Sum={}".format(s))
                  print("Avg=%0.3f" %(s/len(lst)))
                  print("----------------------------------------")
```

==

Misc. Flow Control Statements in Python

==

=>As part Python Programming, we have 3 Misc. Flow Control Statements. They are

a) break
b) continue
c) pass

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

a) break

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

=>"break" statement is used for terminating the execution part of loop based on certain condition.

=>Syntax1 :-   for varname in iterable_object:

```
                         if(Test Cond):
                              break
                    else:
                          else block of statements
                   ---------------------------------------
                   Other statements in Program
                   ----------------------------------------

=>Syntax2 :-        ----------------------------
                    while(Test Cond):
                      -----------------------
                      if(Test Cond):
                            break
                    else:
                          else block of statements
                   ---------------------------------------
                   Other statements in Program
                   ----------------------------------------
```

=>when break statements is executed, else part of 'for' loop and while loop will not execute.

```
            =====================================
                       b) continue
           =====================================
```

=>'continue' statements is used for continueing execution of the loop by taking the PVM to the beging of the loop when certain condition is taking place without execution those statements which are written after continue statement.
=>continue statements to be used inside loops.
=>when continue statement taking place in a loop then PVM executes else part of   'for ' and 'while' loops.
=>Syntax1:-

```
                 for varname in iterable_object:
                       --------------------------
                       if(test cond):
                            continue
                       ----Other statements---------------
                       -----Other statements--------------
                 else:
                      else block of statements
```

=>Syntax2:-

```
                 while (Test Cond1)
                       --------------------------
                       if(test cond2):
                            continue
                       ----Other statements---------------
                       -----Other statements--------------
                 else:
                      else block of statements
=========================X==============================
```

```
#breakex1.py
s="PYTHON"
for val in s:
      print("\t{}".format(val))
print("-----------------------------")
```

```
for val in s:
	if(val=="H"):
		break
	print("\t{}".format(val))
print("-----------------------------")

#breakex2.py
lst=[10,20,30,"Java","Python","Django","Data Science"]
for val in lst:
	if(val=="Python"):
		break
	print("\t{}".format(val))
else:
	print("i am from else part of for loop")
print("end of the program")


#continueex1.py---display     PYHON
s="PYTHON"
for val in s:
	if(val=="T"):
		continue
	print("\t{}".format(val))

#continueex2.py---display     PTON
s="PYTHON"
for val in s:
	if(val=="Y") or (val=="H"):
		continue
	print("\t{}".format(val))

#continueex3.py---
lst=[100,-200,-300,150,450,-23,67,-56,456]
for val in lst:
	if(val<=0):
		continue
	print(val)
else:
	print("i am from else block of for loop")

#continueex4.py---
lst=[100,-200,-300,150,450,-23,67,-56,456]
i=0
while(i<len(lst)):
	if(lst[i]<=0):
		i=i+1
		continue
	else:
		print("\t{}".format(lst[i]))
		i=i+1
else:
	print("i am from else block of while loop")

#continueex5.py
n=int(input("Enter How Many Numbers u have:"))
if(n<=0):
	print("{} is invalid input".format(n))
else:
```

```
        lst=list()  # create an empty list
        for i in range(1,n+1):
                val=int(input("Enter {} Value:".format(i)))
                lst.append(val)
        else:
                print("-------------------------------------")
                print("Original List:{}".format(lst))  # [10, 21, 43, 22, 56]
                print("-------------------------------------")
                #Get Even Numbers
                evenlst=list()
                for val in lst:
                        if(val%2!=0):
                                continue
                        else:
                                evenlst.append(val)
                else:
                        print("-------------------------------------")
                        print("Even List:{}".format(evenlst))  # [10, 22, 56]
                        print("-------------------------------------")
                        #Get Odd Numbers
                        oddlist=[]
                        for val in lst:
                                if(val%2==0):
                                        continue
                                else:
                                        oddlist.append(val)
                        else:
                                print("Odd List:{}".format(oddlist))  # [21,43]
                                print("-------------------------------------")


#Program extracting vowels from given line of text
#vowels.py
line=input("Enter a line of text:")
print("-------------------------------------")
print("Given Line={}".format(line))
print("-------------------------------------")
nov=0
for ch in line:
        if (ch  not in ['a','e','i','o','u','A','E','I','O','U']):
                continue
        else:
                print("\t{} ".format(ch))
                nov=nov+1
else:
        print("Numb er Vowels={}".format(nov))


#voterex1.py
age=int(input("Enter Age of Citizen:"))
if(age>=18):
        print("Citizen is eligible to Vote:")
else:
        print("Citizen is Not eligible to Vote:")

#Allow thr Voter as eligible where age lies with 18 and 100---   -44
#voterex2.py
while(True):
```

```
      age=int(input("Enter age of citizen:"))
      if(age>=18) and (age<=100):
            break
print("Ur Age  is :{} and Ur Citizen is Eligibile to Vote".format(age))

#marksmemo.py
#accept student details such as stno,name
stno=int(input("Enter Student Number:"))
sname=input("Enter Student Name:")
#validation of C Marks
while(True):
      cm=int(input("Enter Marks in C:"))
      if(cm>=0) and (cm<=100):
            break
#validation of CPP Marks
while(True):
      cppm=int(input("Enter Marks in CPP:"))
      if(cppm>=0) and (cppm<=100):
            break
#validation of PYTHON Marks
while(True):
      pytm=int(input("Enter Marks in PYTHON:"))
      if(pytm>=0) and (pytm<=100):
            break
#calculate total marks
totmarks=cm+cppm+pytm
markspercent=(totmarks/300)*100
#decide the grade----cm=40  cpp=40  pytm=40
if(cm<40) or (cppm<40) or (pytm<40):
      grade="FAIL"
else:
      if(totmarks<=300) and (totmarks>=250):
            grade="PASSED in DISTINCTION"
      elif(totmarks<=249) and (totmarks>=200):
            grade="PASSED in FIRST"
      elif(totmarks<=199) and (totmarks>=150):
            grade="PASSED in SECOND"
      elif(totmarks<=149) and (totmarks>=120):
            grade="PASSED in THIRD"

#Display the marks memo
print("*"*50)
print("\tS t u d e n t  M a r k s  R e p o r t")
print("*"*50)
print("\tStudent Number:{}".format(stno))
print("\tStudent Name:{}".format(sname))
print("\tStudent Marks in C:{}".format(cm))
print("\tStudent Marks in CPP:{}".format(cppm))
print("\tStudent Marks in PYTHON:{}".format(pytm))
print("-"*40)
print("\tTotal marks: {}".format(totmarks))
print("\tPercentage of Marks: {}".format(markspercent))
print("-"*40)
print("\tStudent Result: {}".format(grade))
print("*"*50)
```

Output:

```
-----------------
Quatity of Items: 80
Enter price per item: 100
Enter the discount: 10
Enter Tax:  14
------------------------------------
BILL
---------------------
Quantity Sold: 80
Price Per Item: 100
----------------------------------
Total Amount:  8000
Discount:         -800
----------------------------------
Discounted Total: 7200
Tax:                +1008
------------------------------------
Total amount to be paid: 8208
====================================
Electricity Bill
---------------------------------------------------------
Service Number:10101345
Consumer Name: Rakesh
Number of Units Consumed: 256 units
-------------------------------------------------
      Rates for Units
           0 to 100--------Per Unit Rs: 5
         101---200------per Unit cost Rs: 7.5
         201---300------Per Unit cost Rs:10.00
         above 300 per unit cots: Rs: 12
--------------------------------------------------------------------------
------
Q) Convert Celcious temp  into Foreign Heat and Kelvin temp
            ft=(9/5)*ct+32
            kt=ct+273
======================================================

Accept  student number,name and marks in Three Subjects
                                                    C Marks(100),CPP
Marks(100)
                                                    Python Marks (100)


=>Calculate Total Marks of three Subjects(totmarks=cm+cppm+pym )
=>Calculate Percentage of of Marks
=>Decide the Grade

      1) Give Grade=Fail provided the student marks less than 40 atleat
in one subject.
      2) Give Grade=Passed in Distinction provided total marks lies
within 300 and 250
      3) Give Grade=Passed in First provided total marks lies within 249
and 200
      4) Give Grade=Passed in Second provided total marks lies within199
and 150
      5) Give Grade=Passed in Third provided total marks lies within 120
and 149
```

=>Display th Marks Memo.

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

=>The process of defining one loop inside of another loop is called Inner / nested loop.
=>The execution behaviour of Inner / nested loop is that " For every value of Outer loop
    inner loop will execute repeatedly for finite number of times."

```
=>Syntax1:       for varname1 in Iterable_Object1:  # outer for loop
                                              -------------------------------
                                     -------------------------------
                                     for varname-2 in Iterable_Object2: # inner
for loop
                                           -----------------------------
                                         -----------------------------
                                     else:
                                           --------------------------
                                         --------------------------
                              else:
                                        -----------------------------
                                      -----------------------------

=>Syntax2:
                 -----------------------------
                 while( Test Cond1):  # outer while loop
                      ----------------------
                      ----------------------
                      while(Test Cond2): # inner while loop
                            --------------------
                          --------------------
                      else:
                           ---------------------
                         ---------------------
                 else:
                       -------------------------
                       -------------------------


=>Syntax3:-
                 -----------------------------
                 while( Test Cond1):  # outer while loop
                      ----------------------
                      ----------------------
                       for varname-2 in Iterable_Object2: # inner for
loop
                            -----------------------------
                             -----------------------------
                       else:
                            --------------------------
                             --------------------------
                 else:
                       -------------------------
                       -------------------------
```

```
=>Syntax4:
                         for varname1 in Iterable_Object1:  # outer for
loop
                                           -------------------------------
                                 -------------------------------
                                 while(Test Cond2): # inner while loop
                                   --------------------
                                   --------------------
                                 else:
                                  ---------------------
                                  ---------------------
                         else:
                                    -----------------------------
                                  -----------------------------

#innerforloopex1.py--------   for  in for
for i in range(1,6):
      print("-----------------------------")
      print("Val of i--outer for loop={}".format(i))
      print("-----------------------------")
      for j in range(1,4):
            print("\tVal of j--inner for loop={}".format(j))
      else:
            print("------------------------------")
            print("Out of inner for loop")
else:
      print("--------------------------------")
      print("Out of outer for loop:")

#innerforwhileloopex4.py--------  while  in for
for i in range(1,6):  # outer for loop
      print("-----------------------------")
      print("Val of i--outer for loop={}".format(i))
      print("-----------------------------")
      j=1  # Initlization Part
      while(j<=3):  # cond part---inner while loop
            print("\tVal of j--inner while loop={}".format(j))
            j=j+1 # updation part
      else:
            print("------------------------------")
            print("Out of inner while loop")
else:
      print("--------------------------------")
      print("Out of outer for loop:")

#innerwhileforloopex3.py-----for in while
i=1  # Initlization Part
while(i<=5):  # cond part  ---outer while loop
      print("-----------------------------")
      print("Val of i--outer while loop={}".format(i))
      print("-----------------------------")
      for j in range(1,4):   # inner for loop
            print("\tVal of j--inner for loop={}".format(j))
      else:
            print("------------------------------")
            print("Out of inner for loop")
      i=i+1 #outer while loop updation
else:
```

```
	print("--------------------------------")
	print("Out of outer while loop:")

""""
------------------------------
Val of i--outer while loop=1
------------------------------
	Val of j--inner for loop=1
	Val of j--inner for loop=2
	Val of j--inner for loop=3
-------------------------------
Out of inner for loop
------------------------------
Val of i--outer while loop=2
------------------------------
	Val of j--inner for loop=1
	Val of j--inner for loop=2
	Val of j--inner for loop=3
-------------------------------
Out of inner for loop
------------------------------
Val of i--outer while loop=3
------------------------------
	Val of j--inner for loop=1
	Val of j--inner for loop=2
	Val of j--inner for loop=3
-------------------------------
Out of inner for loop
------------------------------
Val of i--outer while loop=4
------------------------------
	Val of j--inner for loop=1
	Val of j--inner for loop=2
	Val of j--inner for loop=3
-------------------------------
Out of inner for loop
------------------------------
Val of i--outer while loop=5
------------------------------
	Val of j--inner for loop=1
	Val of j--inner for loop=2
	Val of j--inner for loop=3
-------------------------------
Out of inner for loop
---------------------------------
Out of outer while loop:

"""


#This program generates mul tables for list of values:
#multables.py
n=int(input("Enter How many values u want to enter:"))
if(n<=0):
	print("{} is invalid Input:".format(n))
else:
	lst=list()  # empty list
	for i in range(1,n+1):
```

```
            val=int(input("Enter {} Value :".format(i)))
            lst.append(val)
      else:
            print("List of Elements:{}".format(lst))
            print("----------------------------------------")
            #logic for mul tables
            for n in lst:  #outer for loop
                  if(n<=0):
                        print("\t{} is invalid input:".format(n))
                  else:
                        print("------------------------------------")
                        print("\tMul Table for :{}".format(n))
                        print("------------------------------------")
                        for i in range(1,11):
                              print("\t{} x {}={}".format(n,i,n*i))
                        else:
                              print("------------------------------------
")

#This program decides whether the given number is prime or not
#prime.py
n=int(input("Enter a number:"))
if(n<=1):
      print("{}  is invalid input:".format(n))
else:
      result=False
      for  i in range(2,n):
            if(n%i==0):
                  result=True
                  break

      if(result):
            print("{} is not prime".format(n))
      else:
            print("{} is prime:".format(n))

#This Program accept the number and find the sum of the digits.
#digits.py
n=int(input("Enter any number:"))
if(n<=0):
      print("{} is invalid input:".format(n))
else:
      s=0
      while(n>0):
            d=n%10
            s=s+d
            n=n//10
      else:
            print("sum of the digits={}".format(s))

#This program accepts any numerical integer value and converts into Roman
Number
#roman.py
n=int(input("Enter any number:"))  # 2009
if(n<=0):
      print("{} is invalid input".format(n))
else:
      while(n>=1000):
```

```
          print("M",end="")
          n=n-1000
     if(n>=900):
          print("CM",end="")
          n=n-900
     if(n>=500):
          print("D",end="")
          n=n-500
     if(n>=400):
          print("CD",end="")
          n=n-400
     while(n>=100):
          print("C",end="")
          n=n-100
     if(n>=90):
          print("XC",end="")
          n=n-90
     if(n>=50):
          print("L",end="")
          n=n-50
     if(n>=40):
          print("XL",end="")
          n=n-40
     while(n>=10):
          print("X",end="")
          n=n-10
     if(n>=9):
          print("IX",end="")
          n=n-9
     if(n>=5):
          print("V",end="")
          n=n-5
     if(n>=4):
          print("IV",end="")
          n=n-4
     while(n>=1):
          print("I",end="")
          n=n-1
```

===

Types of Programming Languages

==

=>In the context of Functional Approach of any Language, we have two types Programming languages. They are

1. Un-Structured Programming Lanugage
2. Structured Programming Language

----

1. Un-Structured Programming Lanugage:

=>Un-Structured Programming Lanugage does not contain the concept called Functions and unable to get code re-usability.

=>Since Un-Structured Programming Lanugage does not contain the concept called
Functions and we get the following Limitations.

1) Application Development time is More
2) Application Takes More Memory Space.
3) Application Execution Time is More
4) Application Performnace is Degraded

```
      5) Redundency (Duplication / repeatation) of the code is More
Examples: GW-BASIC
----------------------------------------------------------
2. Structured Programming Lanugage:
----------------------------------------------------------
=>Structured Programming Lanugage  contain the concept called
   Functions and able to get code re-usability.
=>Since Structured Programming Lanugage  contain the concept called
   Functions and we get the following Advatnages(Functions) .

           1) Application Development time is Less
           2) Application Takes Less Memory Space.
           3) Application Execution Time is Less / Fast.
           4) Application Performnace is Enhanced (Improved)
           5) Redundency (Duplication / repeatation) of the code is
Minimized

Examples:  C,COBOL,CPP,PYTHON,JAVA,.NET...etc
==============================X=========================

            ==============================================
                 Development of Functions in Python(Most Imp)
           ==============================================
=>The purpose of Functions concept is that "To perform Certain Operations
and
    Provides Code Re-usability".
---------------------------------
=>Definition of Function:
---------------------------------
=>Sub program of main Program is called Functions
                 (OR)
=>A part of main program is called Function.
-----------------------------------------------
=>Parts in Functions:
-----------------------------------------------
=>At the dealing with Functions, we have 2 parts. They are
           a) Function Definition
           b) Function Calls
=>Function Definition will exists Only Once and we can have multiple
Function Calls.
=>For Every Function Call there must exists a Function Definition
Otherwise we get NameError

            ============================================
                 Syntax for Defining the Function
           ============================================
=>To define the function in Python Programming, we follow the following
Syntax:

      def    functionname(List of formal params if any ):
                 """ doc String """
                 ------------------------------------------------
                 ------------------------------------------------
                 Block of statements--Performs Operation(Logic)
                 ------------------------------------------------
                 ------------------------------------------------

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

Number of approaches to develop functions

==

=>In Real time, we can define any function in 4 ways.

Approach-1:

=>In This approach, we Take

----> INPUT from Function Calls

----> PROCESSING in Function Body

---->RESULT / OUTPUT in Function Call

Examples:

```
def  sumop(a,b):   # here 'a' and 'b' are called Formal Parameters
      kvr=a+b  # here 'kvr' is called Local Variable--PROCESSING
      return kvr  # here return is a statement used for giving the result
back

#main program
a=float(input("Enter first value:"))
b=float(input("Enter second value:"))
kvr=sumop(a,b)  # Function Call---INPUT sending and OUTPUT taking
print("sum={}".format(kvr))
```

===

Approach-2:

=>In This approach, we Take

----> INPUT taking Inside of Function Body

----> PROCESSING in Function Body

---->RESULT / OUTPUT in Function Body

Example:

```
def  sumop():
      a=float(input("Enter First Value:"))
      b=float(input("Enter Second Value:"))  # Taking Input in Function
Body
      c=a+b  # here 'a' ,'b' and 'c' are called Local Variables--
Processing
      print("sum({},{})={}".format(a,b,c)) # Result

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

===

Approach-3:

=>In This approach, we Take

----> INPUT taking from Function Call

----> PROCESSING in Function Body

---->RESULT / OUTPUT in Function Body

Examples:

```
#This program defines a function and it computes sum of two numbers
#approach3.py
# INPUT Taking from   Function Calls
# PROCESSING  done in Function Body
# RESULT  gives in Function Body
def   sumop(x,y):
```

```
      z=x+y  # PROCESSING  done in Function Body
      print("sum of {} and {}={}".format(x,y,z)) # RESULT  gives in
Function Body

#main program
a=float(input("Enter first value:"))
b=float(input("Enter second value:"))
sumop(a,b) # INPUT Taking from   Function Calls
```

==

Approach-4:

=>In This approach, we Take
        ----> INPUT taking Inside of Function Body
      ----> PROCESSING in Function Body
      ---->RESULT / OUTPUT gives to Function Calls

Example:

```
#This program defines a function and it computes sum of two numbers
#approach1.py
# INPUT Taking from Function Calls
# PROCESSING  done in Function Body
# RESULT  gives to Function Call

def  sumop(a,b):   # here 'a' and 'b' are called Formal Parameters
      kvr=a+b  # here 'kvr' is called Local Variable
      return kvr  # here return is a statement used for giving the result
back

#main program
a=float(input("Enter first value:"))
b=float(input("Enter second value:"))
kvr=sumop(a,b)  # Function Call
print("sum={}".format(kvr))

#This program defines a function and it computes sum of two numbers
#approach2.py
# INPUT Taking in  Function Body
# PROCESSING  done in Function Body
# RESULT  gives in Function Body

def  sumop():
      a=float(input("Enter First Value:"))
      b=float(input("Enter Second Value:"))  # Taking Input in Function
Body
      c=a+b  # here 'a' ,'b' and 'c' are called Local Variables--
Processing
      print("sum({},{})={}".format(a,b,c)) # Result

#main program
sumop()  # Function call

#This program defines a function and it computes sum of two numbers
#approach3.py
# INPUT Taking from   Function Calls
# PROCESSING  done in Function Body
# RESULT  gives in Function Body
```

```
def   sumop(x,y):
      z=x+y  # PROCESSING  done in Function Body
      print("sum of {} and {}={}".format(x,y,z)) # RESULT  gives in
Function Body

#main program
a=float(input("Enter first value:"))
b=float(input("Enter second value:"))
sumop(a,b)

#This program defines a function and it computes sum of two numbers
#approach4.py
# INPUT Taking in  Function Body
# PROCESSING  done in Function Body
# RESULT  gives to Function Call

def  sumop():
      a=float(input("Enter First Value:"))
      b=float(input("Enter Second Value:"))  # Taking Input in Function
Body
      c=a+b  # here 'a' ,'b' and 'c' are called Local Variables--
Processing
      return c

#main program
result=sumop()  # Function call & RESULT  gives to Function Call
print("sum={}".format(result))

#This program defines a function and it computes sum of two numbers
#sumex.py
def  sumop(a,b):   # here 'a' and 'b' are called Formal Parameters
      c=a+b  # here 'c' is called Local Variable
      return c  # here return is a statement used for giving the result
back

#main program
res1=sumop(10,20)  # Function Call
print("sum={}".format(res1))

#This program defines a function and it computes sum of two numbers
#approach4_1.py
# INPUT Taking in  Function Body
# PROCESSING  done in Function Body
# RESULT  gives to Function Call

def  sumop():
      a=float(input("Enter First Value:"))
      b=float(input("Enter Second Value:"))  # Taking Input in Function
Body
      c=a+b  # here 'a' ,'b' and 'c' are called Local Variables--
Processing
      return a,b,c # here return statement can return one or more number
of values
#main program
n1,n2,n3=sumop()  # Function call & RESULT  gives to Function Call
print("sum({},{})={}".format(n1,n2,n3))
print("-----------OR-------------------")
kvr=sumop() # here kvr is a variable of type <class,'tuple'>
```

```
print("sum({},{})={}".format(kvr[0],kvr[1],kvr[2]))

#This Program calcuates Square Root of a given Number without using
sqrt()
#squarerootex1.py
""" Approach-1
----> INPUT from Function Calls
----> PROCESSING in Function Body
---->RESULT / OUTPUT  in Function Call """
def  squareroot(n):
      result=n**0.5
      return result


#main program
x=float(input("Enter a number:"))
res=squareroot(x)  # Function call
print("squareroot({})={}".format(x,res))


#This Program calcuates Square Root of a given Number without using
sqrt()
#squarerootex3.py
""" Approach-3
 ----> INPUT taking from Function Call
      ----> PROCESSING in Function Body
      ---->RESULT / OUTPUT  in Function Body """

def  sqrt(n):
      res=n**0.5
      print("squareroot({})={}".format(n,res))

#main program
n=float(input("Enter a number:"))
sqrt(n)



#This Program calcuates Square Root of a given Number without using
sqrt()
#squarerootex4.py
"""Approach-4
 ----> INPUT taking Inside of  Function Body
----> PROCESSING in Function Body
---->RESULT / OUTPUT  gives to Function Calls """

def  squareroot():
      n=float(input("Enter a number:"))
      res=n**0.5
      return n,res

#main program
m,n=squareroot()  # multi line assignment with Function call
print("Squareroot({})={}".format(m,n))
print("------------OR-----------")
result=squareroot()   # here result is of type tuple
print("squareroot({})={}".format(result[0],result[1]))
```

```
#program accepting list of names and sort the in both acending and
decending order
#sortnames.py
def readnames():
	n=int(input("Enter How Many Number of Names:"))
	if n<=0:
		return None
	else:
		lst=list()
		for i in range(1,n+1):
			name=input("Enter {} Name:".format(i))
			lst.append(name)
		return lst

def  dispnames(names):
	print("-"*50)
	for name in names:
		print("\t{}".format(name))
	print("-"*50)

def  sortnames(stnames):
	#sort the names in Ascending Order
	stnames.sort()
	print("Names In Ascending Order:")
	dispnames(stnames)
	stnames.sort(reverse=True)
	print("Names In Decending Order:")
	dispnames(stnames)

#main program
names=readnames()  # function call
if(names==None):
	print("Invalid Input, try again")
else:
	print("-"*50)
	print("Original Names:")
	dispnames(names) # function call
	sortnames(names) # function call

#This Program cal all Arithmetic Operations by using Functions
#aopex1.py
def   addop():
	a=float(input("Enter First Value for Addition:"))
	b=float(input("Enter Second Value for Addition:"))
	print("sum({},{})={}".format(a,b,a+b))

def  subop():
	x=float(input("Enter First Value for Substraction:"))
	y=float(input("Enter Second Value for Substraction:"))
	print("sub({},{})={}".format(x,y,x-y))

def  mulop():
	x=float(input("Enter First Value for Multiplication:"))
	y=float(input("Enter Second Value for Multiplication:"))
	print("mul({},{})={}".format(x,y,x*y))
def  divop():
	x=float(input("Enter First Value for Division:"))
	y=float(input("Enter Second Value for Division"))
```

```
      print("Div({},{})={}".format(x,y,x/y))
      print("Floor Div({},{})={}".format(x,y,x//y))
def  modop():
      x=float(input("Enter First Value for Modulo Division:"))
      y=float(input("Enter Second Value for Modulo Division"))
      print("Mod({},{})={}".format(x,y,x%y))
def expoop():
      x=float(input("Enter value for Base:"))
      y=float(input("Enter Value for power:"))
      print("pow({},{})={}".format(x,y,x**y))

#main program
addop()
subop()
mulop()
divop()
modop()
expoop()

#This Program cal all Arithmetic Operations by using Functions
#aopex2.py
def   readvalues(op):
      a=float(input("Enter First for {}".format(op)))
      b=float(input("Enter Second for {}".format(op)))
      return a,b
def   addop():
      x,y=readvalues("Addition:")
      print("sum({},{})={}".format(x,y,x+y))

def  subop():
      a,b=readvalues("substraction")
      print("sub({},{})={}".format(a,b,a-b))

def  mulop():
      x,y=readvalues("Multiplication:")
      print("mul({},{})={}".format(x,y,x*y))
def  divop():
      x,y=readvalues("Division:")
      print("Div({},{})={}".format(x,y,x/y))
      print("Floor Div({},{})={}".format(x,y,x//y))
def  modop():
      x,y=readvalues("Modulo Division:")
      print("Mod({},{})={}".format(x,y,x%y))
def expoop():
      x=float(input("Enter value for Base:"))
      y=float(input("Enter Value for power:"))
      print("pow({},{})={}".format(x,y,x**y))

#main program
addop()
subop()
mulop()
divop()
modop()
expoop()
```

==

Parameters and Arguments

==

=>In Functions, we come across two types of Parameters. They are

a) Formal Parameters / Variables
b) Local Parameters / Variables

=>Formal Parameters / Variables used in Function Heading and they are used for storing / holdling the values coming from Function Calls.

=>Local Parameters / Variables used in Function Body and they are used for storing temporary results in Function Body.

=>The values of Formal Parameters and Local parameters can be accessed in corresponding Function Definition only but not possible to access in the context of other Function definitions.

=>Arguments are the variables used in Function Calls and the arguments are also called "Actual Arguments"

=>Hence All Values of Actual Arguments are passing to Formal Parameters and this type of mechanism is called Arguments / Parameter Passing Mechnisms

==

Types of Arguments (or) Parameter Passing Mechnisms

==

=>The mechanism of Passing Values of Actual Arguments to Formal parameters from Function call to Function Definition is called Arguments / Parameter Passing Mechnisms.

=>in Python Programming, we have 5 types Arguments / Parameter Passing Mechnisms. They are

1) Possitional Parameters / Arguments
2) Default Parameters / Arguments
3) Keyword Parameters / Arguments
4) Variable Length Parameters / Arguments
5) Keyword Variable Length Parameters / Arguments

```
#formallocalactualvarex.py
def   disp(a,b):  # here 'a' and 'b' are called "formal Parameters"
      print("Value of a:{}".format(a))
      print("val of b:{}".format(b))
      c=a+b  # here 'c' is called Local Variable
      print("sum of {} and {}={}".format(a,b,c))

#main program
x=float(input("Enter First Value:"))
y=float(input("Enter Second Value:"))
disp(x,y) #Function call --here ''x' and "y" are called Actual Arguments
```

==

1) Possitional Arguments (or) Parameters

==

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

=>This Parameter mechanism also recommends Order of Parameters for Higher accuracy.
=>Python Programming Environment follows by default Possitional Parameters.

Syntax for Function Definition :

```
      def    functionname(parm1,param2.....param-n):
            ------------------------------------------------
            ------------------------------------------------
```

Syntax for Function Call:

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

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

```
#posparaargex1.py
def  dispstudinfo(sno,name,marks):
      print("{}\t{}\t{}".format(sno,name,marks))


#main program
dispstudinfo(10,"Rossum",33.33)
dispstudinfo(20,"Gosling",11.11)
dispstudinfo(40,"Travis",77.77)
dispstudinfo(50,"Kinney",55.55)
```

=====================================
2) Default Parameters (or) arguments
=====================================

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

Syntax: for Function Definition with Default Parameters

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

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

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

```
#posparaargex2.py
def  dispstudinfo(sno,name,marks,city):
      print("{}\t{}\t{}\t{}".format(sno,name,marks,city))


#main program
print("-"*40)
print("Stno\tName\tMarks\tHyd")
print("-"*40)
dispstudinfo(10,"Rossum",33.33,"HYD")
dispstudinfo(20,"Gosling",11.11,"HYD")
dispstudinfo(40,"Travis",77.77,"HYD")
dispstudinfo(50,"Kinney",55.55,"HYD")
print("-"*40)

#defaultparamex1.py
def  dispstudinfo(sno,name,marks,city="HYD"):
      print("{}\t{}\t{}\t{}".format(sno,name,marks,city))


#main program
print("-"*40)
print("Stno\tName\tMarks\tHyd")
print("-"*40)
dispstudinfo(10,"Rossum",33.33)
dispstudinfo(20,"Gosling",11.11)
dispstudinfo(40,"Travis",77.77)
dispstudinfo(50,"Kinney",55.55)
dispstudinfo(60,"Trump",15.55,"USA")
dispstudinfo(70,"Sunil",10.55)
dispstudinfo(80,"Josling",10.55,"RSA")
print("-"*40)

#defaultparamex2.py
def  dispemployeedet(eno=10,ename="Naveen",sal=1.2,cname="IBM"):
      print("{}\t{}\t{}\t{}".format(eno,ename,sal,cname))
```

```
#main program
print("-"*40)
print("Eno\tName\tSal\tCname")
print("-"*40)
dispemployeedet()
dispemployeedet(20,"Sameer")
dispemployeedet(30,"Kumar",1.5)
dispemployeedet(40,"Sampath",1.6,"TCS")
print("-"*40)

#defaultparamex3.py
def  areacircle(r,PI=3.14):
      ac=PI*r**2
      print("Radious={}".format(r))
      print("Area of Circle={}".format(ac))



#main program
areacircle(1.2)
areacircle(2.2)
areacircle(5.2)
areacircle(6.8)

#kwdparamex1.py
def   disp(a,b,c):
      print("{}\t{}\t{}".format(a,b,c))

#main program
print("-"*50)
print("a\tb\tc")
print("-"*50)
disp(10,20,30)# Function call---Possitional args
disp(c=30,a=10,b=20)  # function call---KWD args
disp(10,c=30,b=20) # function call---Possitional and KWD args
disp(10,20,c=30)# function call---Possitional and KWD args
#disp(c=30,10,20)--error->SyntaxError: positional argument follows
keyword argument
print("-"*50)

#kwdparamex2.py
def  dispstuddet(sno,sname,marks,crs="PYTHON"):
      print("{}\t{}\t{}\t{}".format(sno,sname,marks,crs))


#main prog
print("-"*50)
print("Stno\tName\tMarks\tCourse:")
print("-"*50)
dispstuddet(100,"RS",66.66)
dispstuddet(101,"JG",16.55)
dispstuddet(marks=33.33,sno=102,sname="RT")
dispstuddet(103,marks=10.11,sname="MC")
#dispstuddet(marks=13.33,sno=104,sname="ZC","JAVA")---error--SyntaxError:
positional argument follows keyword argument
dispstuddet(marks=13.33,crs="JAVA",sno=104,sname="ZC")
print("-"*50)
```

===
4) Variables Length Parameters (or) arguments
===

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

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

Syntax for function definition with Variables Length Parameters:

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

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

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

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

```
#This program demonstates the concept of Variable Length arguments
#varlenargsex1.py----This program will not execute
def  disp(x):
      print("{}".format(x))

def  disp(x,y):
      print("{}\t{}".format(x,y))

def  disp(x,y,z):
      print("{}\t{}\t{}".format(x,y,z))

def  disp(x,y,z,k):
      print("{}\t{}\t{}\t{}".format(x,y,z,k))

#main program
disp(10)  # Function Call
disp(10,20) # Function call
disp(10,20,30)# Function call
disp(10,20,30,40)# Function call

#This program demonstates the concept of Variable Length arguments
```

```
#varlenargsex2.py
def  disp(x):
      print("{}".format(x))

disp(10)  # Function Call

def  disp(x,y):
      print("{}\t{}".format(x,y))

disp(10,20) # Function call
def  disp(x,y,z):
      print("{}\t{}\t{}".format(x,y,z))

disp(10,20,30)# Function call

def  disp(x,y,z,k):
      print("{}\t{}\t{}\t{}".format(x,y,z,k))

#main program
disp(10,20,30,40)# Function call

#This program demonstates the concept of Variable Length arguments
#purevarlenargsex1.py
def  disp(*a):  # here *a is called Variable length Parameter and type is
tuple
      for val in a:
            print("{}".format(val),end=" ")
      print()


#main program
disp(10)  # Function Call
disp(10,20) # Function call
disp(10,20,30)# Function call
disp(10,20,30,40)# Function call
disp(10,20,30,40,50)# Function call
disp(10,20,30,40,50,"KVR")# Function call
disp("Java","Python")# Function call
disp(True)# Function call


#This program demonstates the concept of Variable Length arguments
#purevarlenargsex2.py
def  sumop(sname, *a):  # here *a is called Variable length Parameter and
type is tuple
      print("-"*50)
      s=0
      print("Name of Student={}".format(sname))
      for val in a:
            print("{}".format(val),end=" ")
            s=s+val
      print()
      print("Sum={}".format(s))
      print("-"*50)

#main program
sumop("RS",10)  # Function Call
sumop("JG",10,20) # Function call
```

```
sumop("TR",10,20,30)# Function call
sumop("MC",10,20,30,40)# Function call
sumop("ZM",10,20,30,40,50)# Function call
sumop("DR",10,20,30,40,50,60)# Function call

#This program demonstates the concept of Variable Length arguments
#purevarlenargsex3.py
def  sumop(sname, *a, crs="Python"):  # here *a is called Variable length
Parameter and type is tuple
      print("-"*50)
      s=0
      print("Name of Student={}".format(sname))
      print(" '{}' is doing '{}' course".format(sname,crs))
      for val in a:
            print("{}".format(val),end=" ")
            s=s+val
      print()
      print("Sum={}".format(s))
      print("-"*50)
#main program
#sumop()-----TypeError: sumop() missing 1 required positional argument:
'sname'
sumop("Hyd",crs="DL")
sumop(sname="Mohan",crs="Testing")
sumop("Rossum")
sumop("RS",10)  # Function Call
sumop("JG",10,20) # Function call
sumop("TR",10,20,30)# Function call
sumop("MC",10,20,30,40)# Function call
sumop("ZM",10,20,30,40,50)# Function call
sumop("DR",10,20,30,40,50,60)# Function call
#sumop(-10,-20,-30,sname="kvr")---TypeError: sumop() got multiple values
for argument 'sname'
#sumop("kvr",crs="DSC",4,5,-4,-5)--SyntaxError: positional argument
follows keyword argument
#sumop("kvr",4,5,-4,-5,"JAVA")---TypeError: unsupported operand type(s)
for +: 'int' and 'str'
sumop("kvr",4,5,-4,-5, crs="JAVA")
#sumop("RW",crs="ML",a=10,20,30,40,50)---SyntaxError: positional argument
follows keyword argument
```

==

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 astrik (** param) and the formal parameter with double astrik 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)

```
#kwdvarlenparamex1.py
def  disp( ** a):  # here **a  is called Key word Variable length Parameter--dict
      print("-"*50)
      for k,v in a.items():
            print("{}--->{}".format(k,v))



#main program
disp(sno=10)
disp(eno=100,ename="JG")
disp(city="TS",capital="HYD",lang="Telugu-Hindi-English" )
disp(eno=100,ename="Rossum",sal=3.4,org="PSF")
disp(a=10,b=20,c=30,d=40,e=50)
disp()
```

#Python Program , which will compute total marks secured by Different Student who are studying in different classes by securing Various subject marks

```
#kwdvarlenparamex2.py
def  findtotalmarks(sname,cls, **marks):
      print("-"*50)
      print("Student Name:{}".format(sname))
      print("Student Studying in:{} class".format(cls))
      print("-"*50)
      totmarks=0
      print("\tSubjectName\tSubJect Marks")
      print("-"*50)
      for subname,submarks in marks.items():
            print("\t{}\t\t{}".format(subname,submarks))
            totmarks=totmarks+submarks
      else:
            print("-"*50)
            print("\tTotal Marks={}".format(totmarks))

#main program
findtotalmarks("RS","X",maths=88,sci=67,soc=55,hindi=66,eng=55,tel=66)
findtotalmarks("JG","XII",Maths=75,Physics=58,Chemistry=55)
findtotalmarks("DR","B.Tech(CSE)", cm=80,CPP=70,Python=50,DSC=44)
findtotalmarks("TR","Research",Python=50)
findtotalmarks("MC","Author")
```

==

Global Variables and Local Variables

==

=>The purpose of Global variables is that "To store Common Values for Different Function Calls"
=>Global Variables are those, whish are defined before all the function calls.
=>Local Variables are those, which are used for storing Temporary results in the Function Body.
=>Local Variables are those, whish are defined within the Function Body
=>Syntax:-

```
                 FileName.py
                 ------------------
                 var1=val1
                 var2=val2   # here var1,var2...global variables
                 def   functionname1(list of formal params if any):
                        ------------------------------
                      var=val  # Local Variables
                      ------------------------------
                 def   functionname2(list of formal params if any):
                        ------------------------------
                      ------------------------------
                 def   functionname-n(list of formal params if any):
                        ------------------------------
                      ------------------------------
```

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

```
#globallocalvarex1.py
pfname="Rossum" # Global Variable
def  learnDataSci():
      crs1="Data Science"  # local Variable
      print("To develop '{}' based applocations, we use '{}' lang\n
invented by {}".format(crs1,lang,pfname))
def  learnML():
      crs2="Machine Learning"  # local Variable
      print("To develop '{}' based applocations, we use '{}' lang\n
invented by {}".format(crs2,lang,pfname))
def  learnDL():
      crs3="Deep Learning" # local Variable
      print("To develop '{}' based applocations, we use '{}' lang\n
invented by {}".format(crs3,lang,pfname))
#main program
lang="PYTHON"  #Global Variable
learnDataSci()
print("-"*70)
learnML()
print("-"*70)
learnDL()
```

==

Functions in Python

==

Index:

=>Purpose of Functions
=>Definition of Functions
=>Parts of Functions
=>Phases of Functions
=>Syntax for Defining Functions in Python
=>Approaches to Define Functions

```
=>Programming Examples
---------------------------------------------------------------------
=>Aguments and Parameters
=>Types of Aguments and Parameters
            i) Possisional Aguments and Parameters
            ii) default Aguments and Parameters
            iii) Keyword Aguments and Parameters
            iv) Variable length Aguments and Parameters
            V) Keyword variable length Aguments and Parameters
=>global and local Parameters
=>global Key word and globals()
=>Anonymous Functions / Lambda Functions
=>Special Functions in Python
            i) filter() with Normal Function and Anonymous Functions
            ii) map() with Normal Function and Anonymous Functions
            iii) reduce with Normal Function and Anonymous Functions
=>Programming Examples
```

```
            ========================================
                 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
          --------------------------
```

```
      ==============================================
            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 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 value 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
                   dictobj=globals()
                   globalval1=dictobj['var1']  # dictobj.get("var1") or
globals()['var1']
                   globalval2=dictobj['var2']  # dictobj.get("var2") or
globals()['var2']
                   ------------------------------------------------------
                   ------------------------------------------------------
```

```
#globalkwdex1.py
a=10  #global variable
def   increment():
      print("Val of a in increment()={}".format(a)) # accessing global
Varaibles (No Need to use global keyword)

#main program
print("Val of a before incrment()--main program={}".format(a)) # 10
increment()
```

```
#globalkwdex2.py
a=10  #global variable
def   increment():
      global a  # refering Global Variable
      print("Line-4-->Val of a in increment()={}".format(a))
      a=a+1
      print("Line-->7-->Val of a in increment()={}".format(a))

#main program
print("Line-->10-->Val of a before incrment()--main
program={}".format(a)) # 10
increment()
print("Line-->12-->Val of a before incrment()--main
program={}".format(a)) # 10
```

```
#globalkwdex3.py
a=10  #global variable
def   increment():
      global a
      print("Line--5-->Val of a in increment()={}".format(a)) # accessing
global Varaibles--10
      a=a+1
      print("Line--7-->Val of a in increment()={}".format(a)) # accessing
global Varaibles--11

def  updateval():
      global a
      print("Line--10---value of a in updateval()={}".format(a)) # 11
```

```
      a=a*2

#main program
print("Line-->14-->Val of a before incrment()--main
program={}".format(a)) # 10
increment()
print("Line-->16-->Val of a after incrment()--main program={}".format(a))
# 11
updateval()
print("Line-->18-->Val of a after incrment()--main program={}".format(a))
# 22


#globalkwdex4.py
a=10  #global variable
a=a+1
def   increment():
      global a
      print("Line--5-->Val of a in increment()={}".format(a)) # accessing
global Varaibles--10
      a=a+1
      print("Line--7-->Val of a in increment()={}".format(a)) # accessing
global Varaibles--11

def  updateval():
      global a
      print("Line--10---value of a in updateval()={}".format(a)) # 11
      a=a*2

#main program
print("Line-->14-->Val of a before incrment()--main
program={}".format(a)) # 10
increment()
print("Line-->16-->Val of a after incrment()--main program={}".format(a))
# 11
updateval()
print("Line-->18-->Val of a after incrment()--main program={}".format(a))
# 22

#globalkwdex5.py
a,b=10,20 # here 'a' and 'b' are called global variables
def  updateval1():
      global a,b
      a=a+1
      b=b+1
def updateval2():
      global a,b
      a=a*2
      b=b*2

#main program
print("Updated Val of 'a' before updateval1():{}".format(a))  # 10
print("Updated Val of 'b' before updateval1():{} ".format(b)) #20
updateval1()
print("\nUpdated Val of 'a' after updateval1():{}".format(a))# 11
print("Updated Val of 'b' after updateval1():{} ".format(b)) # 21
updateval2()
print("\nUpdated Val of 'a' after updateval2():{}".format(a))
```

```
print("Updated Val of 'b' after updateval2():{} ".format(b))

#globalsfunex1.py
a=10
b=20
c=30
d=40 # here 'a','b', 'c'  and 'd' are called Global Variables
def fun1():
      global c,d
      c=c+1     # c=31
      d=d+1    # d=41
      a=100
      b=200   # here 'a' and 'b are called Local Variables
      print("Local Vraible  a={}".format(a)) # a=100
      print("Local variable b={}".format(b)) # b=200
      print("Global Variable c={}".format(c)) # c=31
      print("Global Variable d={}".format(d)) # d=41
      print("Global Variable a={}".format(globals()['a'])) # a=10
      print("Global Variable b={}".format(globals()['b'])) # b=20
      print("============OR=================")
      print("Local Vraible  a={}".format(a)) # a=100
      print("Local variable b={}".format(b)) # b=200
      print("Global Variable c={}".format(c)) # c=31
      print("Global Variable d={}".format(d)) # d=41
      print("Global Variable a={}".format(globals().get('a'))) # a=10
      print("Global Variable b={}".format(globals().get('b'))) # b=20
      res=globals()['a']+globals().get('b')+c+d+a+b
      print("Result={}".format(res))

#main program
fun1()

#globalsfunex2.py
a=10
b="Python"
c=23.45
def   fun2():
      obj=globals()
      print("type of obj=",type(obj))
      for k,v in obj.items():
            print("{}======>{}".format(k,v))
      print("------------------------------------------")
      print("Val of global Variable a=",  obj.get('a'))
      print("Val of global Variable b=",  obj.get('b'))
      print("---------------OR----------------------")
      print("Val of global Variable a=", obj['a'])
      print("Val of global Variable b=",obj['b'])
      print("---------------OR----------------------")
      print("Val of global Variable a=", globals().get('a') )
      print("Val of global Variable b=", globals().get('b') )
      print("---------------OR----------------------")
      print("Val of global Variable a=", globals()['a'])
      print("Val of global Variable b=", globals()['b'])

#main program
fun2()

#globalsfunex3.py
```

```
a,b,c,d=1,2,3,4
def fun2():
      a,b,c,d=10,20,30,40
      d1=globals()
      print("Global variable a={}".format(d1.get('a')))
      print("Global variable b={}".format(d1.get('b')))
      print("Global variable c={}".format(d1['c']))
      print("Global variable d={}".format(d1['d']))
      print("\nLocal variable a={}".format(a))
      print("Local variable b={}".format(b))
      print("Local variable c={}".format(c))
      print("Local variable d={}".format(d))
      result=a+b+c+d+d1.get('a')+d1.get('b')+d1['c']+d1['d']
      print("Result={}".format(result))
      return a,b,c,d

#main program
print(a,b,c,d)
k1,k2,k3,k4=fun2()
print(k1,k2,k3,k4)
```

==

Anonymous Functions (or) Lambda Functions

==

=>Anonymous Functions are those which does not contains any name explicitly.
=>The purpose of Anonymous Functions is that "To Perform Instant Operations"
=>Instant Operations are those which we use at that point of time only and
no longer interested to use in further programs / projects".
=>Anonymous Functions definitions contains Single Executable statement.
=>Anonymous Functions automatically returns the result after executing single
statement.
=>To define Anonymous Functions, we use lambda key word and hence
Anonymous Functions are also called Lambda Functions.

=>Syntax:-
----

varname=lambda params-list : Single Statement

Explanation
=>"varname" is an object of <class,'function'>. here varname itself acts name of
anonymous function.
=>lambda is a keyword and used to defined anonymous Functions.
=>param-list represents list of formal oparameters and they are used for storing
the input values coming from function calls.
=>Single Statement reporsents valid python executable statement.
=============================X================================

```
#anonymousfunex1.py
def   sumop(a,b):   # Normal Function Def
      c=a+b
      return c
```

```
addop= lambda a,b:a+b  # Anonymous Function Definition.

#main program
result=sumop(10,20)
print("type of sumop=", type(sumop)) # <class,'function'>
print("sum by using normal function={}".format(result))
print("-"*60)
res=addop(10,20)
print("type of addop=", type(addop)) # <class,'function'>
print("sum by using anonymous function={}".format(res))

#anonymousfunex2.py
rectarea=lambda l,b : l*b

#main program
l=float(input("Enter Length:"))
b=float(input("Enter Breadth:"))
ar=rectarea(l,b)
print("Area of Rect={}".format(ar))
print("--------------------OR------------------")
print("Area of Rect={}".format( rectarea(l,b) ))

#anonymousfunex3.py
rectarea=lambda l,b : l*b

#main program
print("Area of Rect={}".format( rectarea(float(input("Enter
Length:")),float(input("Enter Breadth:"))) ))

#anonymousfunex4.py

findbig=lambda x,y : x  if x>y  else y
findsmall=lambda k,v: k if k<v else v
#main program
a=int(input("Enter First Value:"))
b=int(input("Enter Second Value:"))
print("max({},{})={}".format(a,b, findbig(a,b)) )
print("min({},{})={}".format(a,b, findsmall(a,b)) )

#anonymousfunex5.py

findbig=lambda x,y,z: x if (x>y) and (x>z) else y if(y>z) and (y>x) else
z
findsmall=lambda k,v,r: k if (k<v) and (k<r) else v if(v<r) and (v<k)
else r
#main program
a=int(input("Enter First Value:"))
b=int(input("Enter Second Value:"))
c=int(input("Enter Third Value:"))
print("max({},{},{})={}".format(a,b,c, findbig(a,b,c)) )
print("min({},{},{})={}".format(a,b,c, findsmall(a,b,c)) )


#anonymousfunex6.py

findbig=lambda x,y,z:   "Value are Equal" if (x==y) and (y==z) and (x==z)
else x if (x>y) and (x>z) else y if(y>z) and (y>x) else z
```

```
findsmall=lambda k,v,r: k if (k<v) and (k<r) else v if(v<r) and (v<k)
else r
#main program
a=int(input("Enter First Value:"))
b=int(input("Enter Second Value:"))
c=int(input("Enter Third Value:"))
print("max({},{},{})={}".format(a,b,c, findbig(a,b,c)) )
print("min({},{},{})={}".format(a,b,c, findsmall(a,b,c)) )


#anonymousfunex7.py

findbig=lambda l1:  max(l1)
findsmall=lambda k: min(k)
#main program
lst=[10,20,-30,40,50,100,-4,-5,0,34,67]
print("max({})={}".format(lst, findbig(lst)) )
print("min({})={}".format(lst, findsmall(lst)) )

#anonymousfunex8.py

findbig=lambda l1:  max(l1)
findsmall=lambda k: min(k)
#main program
print("Enter List of values separated by space:")
lst=[ int(x) for x in input().split()]
print("max({})={}".format(lst, findbig(lst)) )
print("min({})={}".format(lst, findsmall(lst)) )
```

==

Differenences between Anonymous Functions and Normal Functions

==

=>Normal Function are always used for performing Certain Operation which are longer to re-use in other part of python project. where as Anonymous Functions are used for performing Instant Operations.
=>Normal Functions contains Block of statements where Anonymous Functions contains single statement.
=>Normal Functions contains Name explicitly. where as Anonymous Functions does not contains its name explicitly.
=>Normal Functions can return the value(s) by using return statement explicitly. where as Anonymous Functions can return the value implicitly (No need to use return statement).
=>Every Normal Function definiton starts with "def" keyword where Anonymous Functions definition starts with "lambda".

==

Special Functions in Python

==

=>In Python Programming, we have 3 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 the that element will be neglected ". This process will be continued until all elements of Iterable object completed.

```
#filterex1.py
lst=[10,-20,30,-31,-42,41,-31,67,-45]
def    decide(n):
      if(n>0):
           return True
      else:
           return False
#main program
a=filter(decide,lst)  # By using Normal Function
print("Type of a=",type(a)) #  <class,'filter'>
print("content of a=",a)
#connvert filter object into list type.
pslist=list(a)
print("Possitive Elements=",pslist)
```

```
#filterex2.py
decide=lambda n : n>0
#main program
lst=[10,-20,30,-31,-42,41,-31,67,-45]
a=filter(decide,lst)  # By using anonymous function
print("Type of a=",type(a)) #  <class,'filter'>
print("content of a=",a)
#connvert filter object into list type.
pslist=list(a)
print("Possitive Elements=",pslist)
```

```
#filterex3.py
lst=[10,-20,30,-31,-42,41,-31,67,-45]
ps=tuple ( filter(lambda n : n>0 , lst) )
print("Possitive Elements=",ps)
```

```
#filterex4.py
n=int(input("Enter how many elements u have:"))
if(n<=0):
      print("{} is invalid input:".format(n))
else:
      lst=[]
```

```
        print("-"*50)
        print("Enter {} lements:".format(n))
        print("-"*50)
        for i in range(1,n+1):
            val=int(input())
            lst.append(val)
        else:
            print("-"*50)
            print("Original Elements of list:{}".format(lst))
            print("-"*50)
            pslist=list(filter(lambda n: n>0,lst))
            nslist=set(filter(lambda k: k<0,lst))
            zerolist=tuple(filter(lambda k: k==0,lst))
            print("Possitive Elements={}".format(pslist))
            print("Negative Elements={}".format(nslist))
            print("zero Elements={}".format(zerolist))
            print("-"*50)
```

```
#filterex5.py
print("Enter List of elements separated by space:")
lst=[int(x)   for x in input().split()]
print("-"*50)
print("Original Elements of list:{}".format(lst))
print("-"*50)
pslist=list(filter(lambda n: n>0,lst))
nslist=set(filter(lambda k: k<0,lst))
zerolist=tuple(filter(lambda k: k==0,lst))
print("Possitive Elements={}".format(pslist))
print("Negative Elements={}".format(nslist))
print("zero Elements={}".format(zerolist))
print("-"*50)
```

```
#readingvalues.py
#program for reading list of value dynamically from KBD
print("Enter the elements dynamically from KBD separated by space:")
lst=[ int(x) for x in input().split() ] #List comprehension
print("content of list=",lst)

print("\nEnter the elements dynamically from KBD separated by comma:")
lst1=[float(val) for val in input().split(",")]  #List comprehension
print("content of list=",lst1)
print("\nEnter the elements dynamically from KBD separated by #")
lst1=[str(val) for val in input().split("#")]  #List comprehension
print("content of list=",lst1)
```

====================================
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 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 continue until all elements of Iterable_object completed.

----

oldsallist=[10,20,5,30,40]
Company announced 2% hike to every employee

newsallist=list (map(lambda sal : sal*1.02, oldsallist))

```
#mapex1.py
def  hike(sal):
      sal=sal*1.02   #  OR    sal=sal+sal*0.02
      return sal

#main program
oldsallist=[10,20,5,30,40]
obj=map(hike,oldsallist)
print("Type of obj=",type(obj)) # Type of obj= <class 'map'>
newsallist=list(obj)
print("Old Salary List={}".format(oldsallist))
print("New Salary List={}".format(newsallist))
```

```
#mapex2.py
hike=lambda sal: sal*1.02  # anonymous function

#main program
oldsallist=[10,20,5,30,40]
newsallist=list(map(hike,oldsallist))
print("Old Salary List={}".format(oldsallist))
print("New Salary List={}".format(newsallist))
```

```
#mapex3.py
print("Enter Employee old salaries separated by space:")
oldsallist=[float(sal) for sal in input().split()]
newsallist=list(map(lambda sal:sal*1.02,oldsallist))
print("\nOld Salary List:")
for oldsal in oldsallist:
      print("\t{}".format(oldsal))
print("New Salary List:")
for newsal in newsallist:
      print("\t{}".format(round(newsal,2)))
print("-------------------------------------------------")
print("Old Salary\tNew Salary")
print("-------------------------------------------------")
for old,new in zip(oldsallist,newsallist):
      print("\t{}\t{}".format(old,round(new,2)))
print("-------------------------------------------------")
```

```
#mapex4.py
print("Enter list of elements separated by space:")
lst=[ float(val) for val in input().split() ]
sqlist=tuple(map(lambda val:val**2, lst))
clist=tuple(map(lambda val:val**3, lst))
print("---------------------------------------")
print("Given Number\tSquare\tCube")
print("---------------------------------------")
for n,sq,c in zip(lst,sqlist,clist):
      print("\t{}\t{}\t{}".format(n,sq,c))
print("---------------------------------------")
```

```
#mapex5.py
print("Enter list of elements separated by space:")
lst=[ int(val) for val in input().split() ]
possqlist=list(map(lambda n:n**2, list(filter(lambda val:val>0,lst)) ) )

negsqlist=tuple(map(lambda n: n**2, tuple(filter (lambda n:n<0,lst))))
print("\nOriginal List={}".format(lst))
print("\nPossitive Square List={}".format(possqlist))
print("\nNagetaive Square List={}".format(negsqlist))
```

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

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

----

Internal Flow of reduce()

----

step-1:- reduce() selects two First values of Iterable object and place them First var and Second var .

step-2:- The function-name utilizes the values 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

=>The reduce() belongs to a pre-defined module called" functools".

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

```
#reduceex1.py
import functools
print("Enter list of elements separated by space:")
lst=[ int(val) for val in input().split() ]
res=functools.reduce(lambda x,y:x+y,lst)
print("type of res=",type(res))
```

```
print("sum=",res)

#reduceex3.py
import functools
print("Enter list of words separated by space:")
lst=[ str(val) for val in input().split() ]
res=functools.reduce(lambda x,y:x+" "+y,lst)
print("List of words={}".format(lst)) #["Python", "is","an","OOP",
"Lang"]
print("Line of Text=",res) # Python is an OOP Lang

#bigsmall.py
import functools
print("Enter list of elements separated by space:")
lst=[int(x) for x in input().split()]
big=functools.reduce(lambda x,y:x if x>y else y, lst)
small=functools.reduce(lambda x,y : x if x<y else y, lst)
print("max({})={}".format(lst,big))
print("min({})={}".format(lst,small))
```

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

Modules in Python

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

Index:

=>Purpose of Modules
=>Definition of Module
=>Types of Modules
    a) Pre-defined Module
    b) Programmer / User Defined Module
=>Steps for Developing of Programmer / User Defined Module.
=>Number of approaches to re-use modules
    a) By using import statement (4 syntaxes)
    b) By using from ... import statement (3 syntaxes)
=>Programming Examples
=>Re-Loading Modules
=>Programming Examples

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

Introduction and Types of Modules

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

=>We know that Functions concept makes us to understannd how to perform the operations and How re-use the function code within the same program.But Functions concept unable to provided re-usability across the Programs.
=>To re-use the code across the program, In Python we have a concept called MODULES.
=>The purpose of Modules is that "To Re-use the code across the Programs ".
=>Definition of Module:

=>A Module is a collection of Variables, Functions and Classes.

Types of Modules:

=>In Python, we have two types of Modules. They are
a) Pre-defined (or) Built-in Modules
b) Programmer (or) user (or) Custom-defined Modules

a) Pre-defined (or) Built-in Modules

=>These modules are developed by Language Developers and They are available in Python API and whose role is to deal with Universal Requirements.

Examples: functools, sys, random, os, re, threading, cx_Oracle mysql-connector, time....etc
----
b) Programmer (or) user (or) Custom-defined Modules

=>These modules are developed by Language Programmers and They are available in Python Project and whose role is deal with Common Requirements.

Examples: pythoninfo, calculations............etc
=================================X=======================

```
#calculation.py---file name and acts as Module Name
def   addop(a,b):
      print("sum({},{})={}".format(a,b,a+b))
def   subop(a,b):
      print("sub({},{})={}".format(a,b,a-b))
```

```
#Programmer1.py
import calculation
calculation.subop(500,10) # Function Call
```

```
#PythonInfo.py--file name and acts as Module Name
pfname="GUIDO VAN ROSSUM"
pcname="NETHER LANDS"
PI=3.14
```

```
#Programmer2.py
import PythonInfo
print("Father of Python=",PythonInfo.pfname)
print("Python Country Name=",PythonInfo.pcname)
print("Val of Pi=",PythonInfo.PI)
```

==
Development of Programmer-Defined Modules
==

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

Step-1: Define Programmer-Defined Functions
Step-2: Define Variables Names (Global Variables)
Step-3: Define Classes (OOPs principles)
In an IDE and Save them on some file name with an extension .py(Ex: FileName.py)

=>Internally, Once we consider FileName.py as Module Name, Python Execution Environment creates a Folder automatically on the name of __pycache__ and it contains FileName.cpython-310.pyc.

```
=>                __pycache__
                  ------------------------
                        FileName1.cpython-310.pyc
                        FileName2.cpython-310.pyc
                        ----------------------------------------
                        FileName-n.cpython-310.pyc
==========================X========================
```

```
        ==================================================
           Number of approches to re-use the Modules
        ==================================================
```

=>To access the Function Names, Variables and Class Names of Modules, we have two approaches . They are

1) By Using Import statement
2) By using from...import statement.

---

1) By Using Import statement

=>Here 'import' is a keyword
=>The purpose of import statement is that "To access the the variable names,Function names and Class Names in the current python Program w.r.t Module Name/Alias name"
=>This Approach having 4 syntaxes. They are

```
-----------------
=>Syntax1:  importing  single module name
-----------------
      import   module name
Examples:-   import calculation
                  import  pythoninfo
                  import circle
---------------------------------------------------------------------------------------------
=>Syntax2:  importing  Multiple module names
-----------------
 importing   module name1, module name2.....module name-n

Examples:    import calculation,pythoninfo,circle
---------------------------------------------------------------------------------------------
=>Syntax3:  importing  Single module name as  alias name
-------------------------------------------------------------------------------------
 importing   module name as alias name

Examples:    import calculation as c
---------------------------------------------------------------------------------------------
=>Syntax4:  importing Multiple module names as  alias names
-------------------------------------------------------------------------------------
import module name1 as alias name , module name2 as ailas name
...........modulename-n  as alias name.

Examples:    import calculation as c,pythoninfo as p,circle as r
```

=>After importing the Module Name with import Import statement then we must access the Variable Names , Function Names and Class names of the module w.r.t Module Name otherwise we get Name Error.

```
            ModuleName.Variable Name
            ModuleName.Function Name
            ModuleName.Class Name
                  (OR)
            Alias Name of ModuleName.Variable Name
            Alias Name ofModuleName.Function Name
            aliasName of ModuleName.Class Name
```

============================x============================
2) By Using from ....import statement

=>Here 'from' and 'import' are the keywords
=>The purpose of from...import statement is that "To access the the variable names,Function names and Class Names in the current python Program without using Module Name/Alias name"

=>This Approach having 3 syntaxes. They are

=>Syntax1: importing Variable names, function names and class names of single module name
----
from module name import Var1,....Var-n, FunName1...FunName-n, Class Name-1,......Class Name-n

Examples: from calculation import subop,addop

=>Syntax2: importing Variable names, function names and class names of module name with alias name
----
from module name1 import Variables as alias name, FunNames as alias names, Class Names as alias name.

```
Examples:   from calculation import subop  as sp ,addop as ap
               from  math import sqrt as s, pi as k
```

=>Syntax3: importing ALL Variable names, function names and class names of module name
----
=> from module name import *
=>Here * represents a wild character and it instructs the PVM to import all variables , Functions and Class Names. (This Syntax is not recommeded bcoz It provides required Variables, Functions and classes and also provides un-interested Variables, function names and Class names and leads More Memory space and Takes more Execution Time)

```
Examples:   from calculation import *
                from  math import *
```

=>After imporing the Module Name with from....import statement ,we must access the Variable Names,Function Names and Class names of the module Directly without preceded with Module Name or alias name.

Variable Name
Function Name
Class Name

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

```
#calculation.py---file name and acts as Module Name
def   addop(a,b):
      print("sum({},{})={}".format(a,b,a+b))
def   subop(a,b):
      print("sub({},{})={}".format(a,b,a-b))


#circle.py---file name and treated as Module Name
PI=3.14  # Global Variable
def   area():  # Programmer-defined Function
      r=float(input("Enter Radious for Area of Circle:"))
      ac=PI*r**2
      print("Area of Circle={}".format(ac))
def   peri():   # Programmer-defined Function
      r=float(input("Enter Radious for Perimeter of Circle:"))
      pc=2*PI*r;
      print("Peri. of Circle={}".format(pc))


#PythonInfo.py--file name and acts as Module Name
pfname="GUIDO VAN ROSSUM"
pcname="NETHER LANDS"
PI=3.14

#Programmer1.py
from calculation import *
from  math import *
subop(500,10) # Function Call
addop(23,34)
print(sqrt(625))
print("Val of pi=",pi)

#Programmer2.py
import PythonInfo
print("Father of Python=",k.pfname)
print("Python Country Name=",k.pcname)
print("Val of Pi=",k.PI)

#Programmer3.py
import circle ,calculation as c,calendar  as a
#import math as m
from math import sqrt as k
print(circle.PI)  # Function call
c.addop(2,3)
print(a.month(2022,4))

print(k(49))
print(k(123))
print(k(25))
```

```
#aopmenu.py--file name and acts as module name
def  menu():
      print("="*50)
      print("\tA r i t h m e t i c  O p e r a t i o n s")
      print("="*50)
      print("\t1. Addition")
      print("\t2. Substraction")
      print("\t3. Multiplication")
      print("\t4. Division")
      print("\t5. Modulo Division")
      print("\t6. Exponentation")
      print("\t7. Exit")
      print("="*50)

      #aoperations.py--file name and acts as module name
def    addop():
      a=float(input("Enter First Value for Addition:"))
      b=float(input("Enter Second Value for Addition:"))
      print("Sum({},{})={}".format(a,b,a+b))
def  subop():
      a=float(input("Enter First Value for Substraction:"))
      b=float(input("Enter Second Value for Substraction:"))
      print("Sub({},{})={}".format(a,b,a-b))
def  mulop():
      a=float(input("Enter First Value for Multiplication:"))
      b=float(input("Enter Second Value for Multiplication:"))
      print("Mul({},{})={}".format(a,b,a*b))

def  divop():
      a=float(input("Enter First Value for Division:"))
      b=float(input("Enter Second Value for Division:"))
      print("Div({},{})={}".format(a,b,a/b))
      print("Floor Div({},{})={}".format(a,b,a//b))

def modop():
      a=float(input("Enter First Value for Modulo Div:"))
      b=float(input("Enter Second Value for Modulo Div:"))
      print("Mod({},{})={}".format(a,b,a%b))
def expoop():
      a=float(input("Enter Base:"))
      b=float(input("Enter Power:"))
      print("exp({},{})={}".format(a,b,a**b))

#aopdemo.py---main program
import sys
from aoperations import addop,subop,mulop,divop,modop,expoop
from aopmenu import menu
while(True):
      menu()
      ch=int(input("Enter ur Choice:"))
      match(ch):
            case 1:
                        addop()
            case 2:
                        subop()
            case 3:
                        mulop()
```

```
            case 4:
                        divop()
            case 5:
                        modop()
            case 6:
                        expoop()
            case 7:
                        print("Thanks for Using This program!")
                        sys.exit()
            case _ :  # default case block.
                        print("Ur Selection of Operation is Wrong-try
again")
```

```
#Program for accepting list of names and sort them in both ascending and
decending order
#sortnames.py--file name and acts as Module Name (sortnames.cpython-
310.pyc)
def  readnames():
      print("Enter List of Names separated by space:")
      lst=[str(names) for names in input().split()]
      return lst
def   dispnames(names):
      print("-"*50)
      for name in names:
            print("\t{}".format(name))
      else:
            print("-"*50)

def  sortnames():
      names=readnames()
      print("Original Names:")
      dispnames(names)
      #sort the names in Ascending order
      names.sort()
      print("Sorted Names in Ascending Order:")
      dispnames(names)
      #sort the names in Decending order
      print("Sorted Names in Decending Order:")
      names.sort(reverse=True)
      dispnames(names)

#Sortnamesdemo.py
from sortnames import sortnames as sn
sn() # function call
```

            ==
                  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 / 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()  #previously imported module
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)


#shares.py---file name and acts as module name
def   sharesinfo():
      d={"IT":111,"Pharma":222,"Auto:":5,"IRCTC":7}
      return d

#sharesdemo.py------Viewed by Varma
import shares
import time,imp   #  or  importlib
def   disp(k):
      print("="*50)
      print("\tShare Name\tShare Value")
      print("="*50)
      for sn,sv in k.items():
            print("\t{}\t\t{}".format(sn,sv))
      else:
            print("="*50)

#main program
d=shares.sharesinfo()
disp(d)
print("Going to sleep ...")
time.sleep(20)
print("i am coming from Sleep")
imp.reload(shares)
d=shares.sharesinfo()
```

```
disp(d)

      =============================================
                  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-11AM\\ACKAGES\\BANK")
                  (or)
sys.path.append("E:\KVR-PYTHON-11AM\ACKAGES\BANK")
            (or)
sys.path.append("E:\KVR-PYTHON-11AM/ACKAGES/BANK")
```

----

ii)by using PYTHONPATH Enviromental Variables:

=>PYTHONPATH is one of the Enviromental Variable
Steps for setting PYTHONPATH=E:\KVR-PYTHON-11AM\PACKAGES\BANK

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

Exceptional Handling

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

Index

=>Purpose of Exception Handling
=>Types of errors
- a) Compile Time Errors
- b) Logical Errors
- c) Runtime Errors

=>Definition of Exception
=>Definition of Exception Handling
=>Types of Exceptions
- a) Pre-defined exceptions
- b) Programmer / User / Custom Defined exceptions

=>Key words used in Exception Handling
1) try
2) except
3) else
4) finally
5) raise

=>Syntax for Handling the exceptions
=>Programming Examples
=>Development Programmer / User / Custom Defined exceptions
=>Programming Examples
=>ATM Case study.

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

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

Exceptional Handling

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

=>The purpose of exception handling is that "To build Robust (Strong) applications"
=>To develop any real time project, we must use a language. By the language, we develop, compile and execute various programs.During this process, we get Various errors. They are classfied into 3 types.
1) Compile Time error.
2) Logic Errors.
3) Runtime Errors.

----

1) Compile Time error.

=>Compile Time error are those which are occuring during Compile Time.
            (.py----->.pyc)
=>Compile Time error occurs due to syntaxes are not followed by programmer.
=>Compile Time errors solved by Programmers at Development Level.
----

2) Logic Errors.

=>Logical errors are those which are occuring during Execution / run Time.
=>Logical errors are occurs due to wrong representation of Logic
=>Logical errors always gives Wrong result and they solved by programmers at
    development time.
----
3) Runtime Errors.

=>Runtime errors are those which are occuring during Execution / run Time.
=>Runtime errors occurs due to Wrong / Invalid Inputs entered by End /
    Application Users.
=>Runtime errors are addresed by Programmers during development time.
==
Points to be remembered in Exceptions Handling
----

1. When the application user enters Invalid Input then we get Runtime Errors.
2. By default Runtime Errors generates Technical errors messdages and they are
    uderstandable by bProgrammers but not by end users .
  ----
3. Definition of Exception:-   Every Runtime Error is called Exception.

 (Invalid Input-->Runtime Error-->Exception. Hence Every Invalid input gives
                 exception)
4. Every exception by default generates Technical Error Message. they are
    uderstandable by bProgrammers but not by end users . Hence Industry recommends , Convert technical error messages into user-friendly error messages by using "exception handling" concept
----
5. Definition of Exception Handling:

=>The Process of Converting Technical Error Messages into User-Freindly Error Messages is called Exception Handling.

6. When an exception occurs internally 3 steps takes place. They are
      a) Program execution terminated abnormally.
      b) PVM comes out of Program flow without executing rest of the statements
      c) By default, PVM generates Technical Error Messages.
7. To do (a), (b) and (c) steps, PVM create an object of appropriate exception class.

8. When an exception occurs then PVM create an object of appropriate exception class, Program execution terminated abnormally, PVM comes out of Program flow without executing rest of the statements and By default, PVM generates Technical Error Messages.

9. Hence In Python all exception are are considered as objects and behind of objects there exception exception class names.
10. Therefore, Normal Classes provides Successful execution of program and exception classes provides Abnormal Termination.
===========================X==============================

=====================================
Types of exceptions in Python
=====================================

=>In Python Programming, we have two types of Exceptions. They are
1) Pre-defined (or) Built-in Exceptions
2) Programmer (or) User (or) Custom Defined Exception.

---

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

=>These exceptions are already defined (or) developed in Python Software / API and they are dealing with "Universal Problems"
=>Some of the Universal Problems are
a) Division By Zero Problems (ZeroDivisionError)
b) Invalid Number format (ValueError)
c) Invalid Arguments Passing (TypeError)
d) No valid Key in dict object (KeyError)
e) Invalid Index In Indexing Operations (IndexError)...etc

---

2) Programmer (or) User (or) Custom Defined Exception.

=>These exceptions are developed by Python Programmer and they are available in Python Projects and they are used by Other Python Programmers for dealing with "Common Problems".
=>Some of the "Common Problems" are
a) Attempting to enter Invalid PIN in ATM Based Applications(PinError).
b) Attempting to enter Invalid User Name / Passward (LoginError)
c) Attempting to withdraw More Amount than existing bal (InSuffFundError).
d) Attempting to insert the card in reverse order...etc (InsertError)

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

==
Handling the exceptions in python
==

=>Handling the exceptions in python is nothing but Converting Technical Error Messages into User-Friendly Error Messages. To do this Python Programming Provides 5 Key words. They are
1) try
2) except
3) else
4) finally

```
            5) raise
--------------------------------------------------------
Syntax for handling the exceptions:
--------------------------------------------------------
            try:
                Block of Statements generates
                exceptions in Python Program
            except  exception-class-name-1:
                Block of statements generates
                User-Freindly Error Message
            except  exception-class-name-2:
                Block of statements generates
                User-Freindly Error Message
            ------------------------------------------------
            ------------------------------------------------
            except  exception-class-name-n:
                Block of statements generates
                User-Freindly Error Message
            else:
                 Block of statements recommended
                 to generates Results
            finally:
                 Block of Statements executing
                 Compulsorily.
```

```
#Program for accepting two integer values and find their division
#Div1.py
s1=input("Enter First Value:")
s2=input("Enter Second Value:")
#s3=s1/s2---invalid process
a=int(s1)  #------X
b=int(s2) #-------X
c=a/b    #--------X
print("Val of a=",a)
print("Val of b=",b)
print("Div=",c)
```

```
#Program for accepting two integer values and find their division
#Div2.py
try:
      s1=input("Enter First Value:")
      s2=input("Enter Second Value:")
      a=int(s1)
      b=int(s2)
      c=a/b
except ZeroDivisionError:
      print("\nDon't enter Zero for Den...")
except ValueError:
      print("\nDon't enter strs / symbols / alpha-numerics")
else:
      print("------------------------------")
      print("Val of a=",a)
      print("Val of b=",b)
      print("Div=",c)
      print("------------------------------")
finally:
      print("\nI am from finally Block")
```

===
Explanation for the key words in handling the exceptions
===

----
1) try:

=>It is a block, In which we write block of statements generating exceptions. In
otherwords, what are all the statements are generating exceptions, those statements must be written within try block and it is known as exception monitering block.
=>When the exception occurs in try block, PVM comes out of try block and
executes appropriate except block and generates User-Friendly error message.
=>After executing except block, PVM never comes to try block to execute the rest
of the statements.
=>Every try block must contain atleast one except block and it is recommeded to
write multiple except blocks for generating multiple user-friendly error messages.
=>Every try block must be immediately followed by except block (Otherwise we get
syntaxerror).
---x---

2) except

=>It is the block, in which we write block of statements displays User-Friendly
error messages. In otherwords, except block will supresses the Technical error messages and displays User-Friendly error messages and except block is called
exception processing block.
=>Note:- Handling the exception = try block + except block
=>except block will execute when an exception occurs in try block.
=>Even we write multiple except blocks, PVM can execute only one except block
depends on type of exception occurs in try block.
=>we must the except block after try block and before else block.
----

3) else

=>It is block, in which we write block of statements recommeded to displays
Result of the python program[Result generating block].
=>else block will execute when there is no exception occurs in try block.
=>Writing else block is optional.
=>we write else block after except block and before finally block.
----

4) finally

=>It is a block, in which we write block of statements for Reqlinquishing (Closing
   or releasing or give-up or clean-up) the resources (files, databases) which are obtained in try block . [known as Resources Reqlinquishing Block]
=>finally block will execute Compulsorily (if we write)
=>Writing the finally block is optional.
=>We write finally block after else block .
---X---

===
Various forms of except blocks
===

=>In Python Programming, we can use except block in various forms They are

----
Syntax-1:

```
       try:
          -----------------
        -----------------
      except   exception-class-name-1:
            -----------------------------
      except   exception-class-name-2:
            -----------------------------
```

=>This syntax handles one exception at a time

Syntax-2:

```
try:
   -----------------
   -----------------
except (exception-class-name-1,exception-class-name-2..exception-class-name-n):
            -----------------------------
            ------------------------------
```
=>This syntax handles multiple specific exceptions by using single except block.

Syntax-3:

```
       try:
          -----------------
        -----------------
      except   exception-class-name-1  as  alias name:
            print(alieas name)
      except   exception-class-name-2 as alias name:
            print(alieas name)
```

=>This syntax handles one exception at a time and stores technical error messages in alias name generated due to exception occurance.

Syntax-4:

```
       try:
          -----------------
        -----------------
      except   Exception:
            print("OOPs some thing went wrong")
```

=>This syntax handles all types of exception and but unable generates user-friendly error messages.

```
#Program for accepting two integer values and find their division
#Div3.py
try:
      s1=input("Enter First Value:")
      s2=input("Enter Second Value:")
      a=int(s1)
      b=int(s2)
      c=a/b
except (ZeroDivisionError,ValueError): # Multi exception handling block
      print("\nDon't enter Zero for Den...")
      print("\nDon't enter strs / symbols / alpha-numerics")
else:
      print("------------------------------")
      print("Val of a=",a)
      print("Val of b=",b)
      print("Div=",c)
      print("------------------------------")
finally:
      print("\nI am from finally Block")
```

Syntax-5:

```
try:
   -----------------
   -----------------
except   :  # default except block
            -----------------------------
            ------------------------------
```

=>This syntax handles all types exceptions in except block and it is not recommeded bcoz enduser not getting user-friendly error messages.

```
#Program for accepting two integer values and find their division
#Div4.py
try:
      s1=input("Enter First Value:")
      s2=input("Enter Second Value:")
      a=int(s1)
      b=int(s2)
      c=a/b
except    :
      print("\nOOPs some went wrong...")
else:
```

```
	print("------------------------------")
	print("Val of a=",a)
	print("Val of b=",b)
	print("Div=",c)
	print("------------------------------")
finally:
	print("\nI am from finally Block")

#Program for accepting two integer values and find their division
#Div5.py
try:
	s1=input("Enter First Value:")
	s2=input("Enter Second Value:")
	a=int(s1)
	b=int(s2)
	c=a/b
except ZeroDivisionError as k:
	print(k)  # division by zero
except ValueError as v:
	print(v) # invalid literal for int() with base 10: 'kvr123'
else:
	print("------------------------------")
	print("Val of a=",a)
	print("Val of b=",b)
	print("Div=",c)
	print("------------------------------")
finally:
	print("\nI am from finally Block")

#Program for accepting two integer values and find their division
#Div6.py
try:
	s1=input("Enter First Value:")
	s2=input("Enter Second Value:")
	a=int(s1)
	b=int(s2)
	c=a/b
except Exception as e :
	print("\nOOPs some went wrong...",e)
else:
	print("------------------------------")
	print("Val of a=",a)
	print("Val of b=",b)
	print("Div=",c)
	print("------------------------------")
finally:
	print("\nI am from finally Block")
```

==

2) Development of Programmer (or) User (or) Custom Defined Exception.

==

=>These exceptions are developed by Python Programmer and they are available in Python Projects and they are used by Other Python Programmers for dealing with "Common Problems".

=>Some of the "Common Problems" are

a) Attempting to enter Invalid PIN in ATM Based Applications(PinError).
b) Attempting to enter Invalid User Name / Passward (LoginError)
c) Attempting to withdraw More Amount than existing bal (InSuffFundError).
d) Attempting to insert the card in reverse order...etc (InsertError)

=>Steps for Developing Programmer-Defined Exceptions:

----

1) Choose the Programmer-Defined Class Name
2) The Programmer-Defined Class Name must Inherit from pre-defined exception super class called " Exception (or) BaseException " . Hence programmer-defined class becomes programmer defined exception class.
3) Save the above code on file name with an extension .py .

Example: Define a programmer-defined exception class "PinError"

```
class PinError (Exception ):pass
```

Example: Define a programmer-defined exception class "LoginError"

```
class LoginError(BaseException ):pass
```

=>To develop any python based application with Programmer-defined exceptions, we must go for 3 phases: They are

----

Phase-1: Develop Programmer-Defined Exception Class Name.

Example:

```
            #kvr.py-----File name and acts as Module Name---(3)
                  # (1)                       (2)
      class KvrDivisionError(Exception):pass
```

Phase-2: Develop a common function, In which we Hit / raise the exceptions

----

```
#div.py---file name and acts as Module Name
from kvr import KvrDivisionError
def   division(a,b):
      if(b==0):
            raise KvrDivisionError
      else:
            return (a/b)
```

Phase-3: Develop a main program for handling the exceptions

```
#divdemo.py----main program
from div import division
from kvr import KvrDivisionError
x=int(input("Enter First Value:"))
y=int(input("Enter Second Value:"))
try:
      res=division(x,y)  # calling Function
except KvrDivisionError :
      print("\nDon't enter zero for Den...")
else:
      print("Result={}".format(res))
finally:
      print("\nI am from finally Block")
```

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

===

Various forms of except blocks

===

=>In Python Programming, we can use except block in various forms They are

----

Syntax-1:

```
       try:
          -----------------
        -----------------
      except   exception-class-name-1:
            -----------------------------
      except   exception-class-name-2:
            -----------------------------
```

=>This syntax handles one exception at a time

Syntax-2:

```
try:
   -----------------
   -----------------
except (exception-class-name-1,exception-class-name-2..exception-class-name-n):
            -----------------------------
            ------------------------------
```

=>This syntax handles multiple specific exceptions by using single except block.

Syntax-3:

```
       try:
          -----------------
        -----------------
      except   exception-class-name-1  as  alias name:
            print(alieas name)
      except   exception-class-name-2 as alias name:
```

```
            print(alieas name)
```

=>This syntax handles one exception at a time and stores technical error messages in alias name generated due to exception occurance.

Syntax-4:

```
       try:
          -----------------
        -----------------
      except   Exception:
            print("OOPs some thing went wrong")
```

=>This syntax handles all types of exception and but unable generates user-friendly error messages.

Syntax-5:

```
try:
   -----------------
   -----------------
except   :  # default except block
            -----------------------------
            ------------------------------
```

=>This syntax handles all types exceptions in except block and it is not recommeded bcoz enduser not getting user-friendly error messages.

==

raise key word

==

=>raise keyword is used for hitting / raising / generating the exception provided some condition must be satisfied.

```
=>Syntax:-     if (Test Cond):
                        raise   <exception-class-name>
```

Examples:

```
from kvr import KvrDivisionError
def division(a,b):
      if(b==0):
            raise KvrDivisionError
      else:
            return (a/b)


#div.py---file name and acts as Module Name
from kvr import KvrDivisionError
def   division(a,b):
      if(b==0):
            raise KvrDivisionError
      else:
            return (a/b)
```

```
# here   division(-,-) is a common function


#divdemo.py----main program
from div import division
from kvr import KvrDivisionError
try:
      x=int(input("Enter First Value:"))
      y=int(input("Enter Second Value:"))
      res=division(x,y)  # calling Function
except KvrDivisionError :
      print("\nDon't enter zero for Den...")
except ValueError:
      print("Don't enter strs/ symbols/ alpha-numeric")
except Exception as e:
      print(e)
else:
      print("Result={}".format(res))
finally:
      print("\nI am from finally Block")

#kvr.py-----File name and acts as Module Name---(3)
                  # (1)                          (2)
class KvrDivisionError(Exception):pass


# Here KvrDivisionError is comes under programmer-defined exception sub
class

#Invalid.py---file name and acts as module name

class InvalidInputError(Exception):pass

class ZeroError(BaseException):pass

#multable.py---file name and acts as module name
from Invalid import InvalidInputError,ZeroError

def   table():
      n=int(input("Enter a number:")) #implicitly PVM raises ValueError
      if(n<0):
            raise InvalidInputError  # explicitly we are raising
exception
      elif(n==0):
            raise ZeroError
      elif(n>0):
            print("-"*40)
            print("Mul Table for :{}".format(n))
            print("-"*40)
            for i in range(1,11):
                  print("\t{} x {}={}".format(n,i,n*i))
            print("-"*40)


#multabledemo.py--main program
from multable import table
```

```
from Invalid import InvalidInputError,ZeroError
try:
      table()
except InvalidInputError:
      print("\nDON'T ENTER -VE  NUMBER:")
except ZeroError:
      print("\nDON'T ENTER ZERO :")
except ValueError :
      print("\nDON'T ENTER strs/ symbols/alpha-numerics")


#atmmain.py-----file name
from atmmenu import atmmenu
import sys
from banking import deposit,withdraw,balenq
from bankexcep import DepositError,WithdrawError,InsuffFundError

while(True):
      atmmenu()
      try:
            ch=int(input("Enter Ur Choice:"))
            match (ch):
                  case 1:
                              try:
                                    deposit()
                              except ValueError:
                                    print("\nDON'T Deposit strs/
symbols/alpha-numerics in ur Account")
                              except DepositError:
                                    print("Don't Deposit  -ve and Zero
Value in ur Account:")
                  case 2:
                                    try:
                                          withdraw()
                                    except ValueError:
                                          print("\nDON'T withdraw strs/
symbols/alpha-numerics from ur Account")
                                    except WithdrawError:
                                          print("Don't withdraw  -ve and
Zero Value from ur Account:")
                                    except InsuffFundError:
                                          print("U don't have sufficient
Funds--read python notes")
                  case 3:
                              balenq()
                  case 4:
                                    print("\nThanks for using this ATM
App!")
                                    sys.exit()
                  case _:
                                    print("Ur Selection of Operation is
wrong-try again")
      except ValueError:
            print("\nDON'T ENTER strs/ symbols/alpha-numerics for ur
choice")


#atmmenu.py------File Name and acts as Module Name
```

```
def atmmenu():
      print("="*50)
      print("\tATM Operations")
      print("="*50)
      print("\t1.Deposit")
      print("\t2.Withdraw")
      print("\t3.Bal Enq")
      print("\t4.Exit")
      print("="*50)
```

      __init__

```
#atmmain.py-----file name and acts as module name
from atmmenu import atmmenu
import sys
from banking import deposit,withdraw,balenq
from bankexcep import DepositError,WithdrawError,InsuffFundError
def sbi():
      while(True):
            atmmenu()
            try:
                  ch=int(input("Enter Ur Choice:"))
                  match (ch):
                        case 1:
                                    try:
                                          deposit()
                                    except ValueError:
                                          print("\nDON'T Deposit strs/
symbols/alpha-numerics in ur Account")
                                    except DepositError:
                                          print("Don't Deposit  -ve and
Zero Value in ur Account:")
                        case 2:
                                          try:
                                                withdraw()
                                          except ValueError:
                                                print("\nDON'T withdraw
strs/ symbols/alpha-numerics from ur Account")
                                          except WithdrawError:
                                                print("Don't withdraw  -ve
and Zero Value from ur Account:")
                                          except InsuffFundError:
                                                print("U don't have
sufficient Funds--read python notes")
                        case 3:
                                    balenq()
                        case 4:
                                          print("\nThanks for using this
ATM App!")
                                          sys.exit()
                        case _:
                                          print("Ur Selection of Operation
is wrong-try again")
            except ValueError:
```

```
                 print("\nDON'T ENTER strs/ symbols/alpha-numerics for ur
choice")

#atmmenu.py------File Name and acts as Module Name
def atmmenu():
      print("="*50)
      print("\tATM Operations")
      print("="*50)
      print("\t1.Deposit")
      print("\t2.Withdraw")
      print("\t3.Bal Enq")
      print("\t4.Exit")
      print("="*50)


      #bankexcep.py--file name and acts as module name
class DepositError(Exception):pass
class WithdrawError(BaseException):pass
class InsuffFundError(Exception):pass

   #banking.py---file name and acts as module name
from bankexcep import DepositError,WithdrawError,InsuffFundError
bal=500.00 # global variable
def  deposit():
      damt=float(input("Enter how much amount u want to deposit:")) #
ValueError
      if(damt<=0):
            raise DepositError
      else:
            global bal
            bal=bal+damt
            print("Ur Account xxxxxxx123 credited with INR
:{}".format(damt))
            print("Now Ur Current Bal INR:{}".format(bal))

def  withdraw():
      global bal
      wamt=float(input("Enter how much amount u want to withdraw:")) #
ValueError
      if(wamt<=0):
            raise WithdrawError
      elif( (wamt+500)>bal ):
            raise InsuffFundError
      else:
            bal=bal-wamt
            print("Ur Account xxxxxxx123 debited with
INR:{}".format(wamt))
            print("Now Ur Current Bal INR :{}".format(bal))

def balenq():
      print("Now Ur Current Bal INR :{}".format(bal))

#runproject.py--file name and module name
import getpass,sys
from atmmain import sbi
def  runatm():
      ctr=0
      while(True):
```

```
            pin=getpass.getpass(prompt="Enter ur Pin:")
            if(pin=="2675"):
                  sbi()
            else:
                  print("Ur pin is invalid, try again")
                  ctr=ctr+1
                  if(ctr==3):
                        print("Ur card blocked")
                        sys.exit()
```

==
Files in Python
==

Index:

=>Purpose of Files
=>Types of Applications in the context of files
            a) Non-Persistant Applications
            b) Persistant Applications
=>Def. of File
=>Types of Files
            a) Text Files
            b) Binary Files
=>Operations on Files
            a) Read Operation
            b) Write Operation
=>File Opening Modes
            1) r   2) w   3) a   4) r+   5) w+   6) a+   7) x
=>Number of Approaches to open the file
            a) By using open()
            b) By Using " with open() as "
=>Functions required For Reading the Data from Files
            a) read()
            b) read(no.of chars)
            c) readline()
            d) readlines()
=>Random Access Files
=>Functions required For Writing the Data to the Files
            a) write()
            b) writelines()
=>Programming Examples

=>Pickling (Object Serialization) and Un-Pickling (Object De-Serialization)
=>Module Name for Pickling and Un-pickling
=>Programming Examples
================================X================================

==
Data Persistenecy by Files of Python
==

Def. of File:

=>A File is a collection Records.

=>Files Resides in Secondary Memory.
=>Technically, File Name is a named location in Secondary Memory.

=>All the objects data of main memory becomes records in File of Secondary memory and Vice-Versa.

Def. of Stream:

=>The Flow of Data between Main Memory and File of Seconday memory is called
    Stream.

==
Types application in Files of Python
==
=>The main Purpose of Files is that " To Achieve the Data Persistency ".
=>In the context of files, we have two types of Applications. They are
    a) Non-Persistant Applications.
    b) Persistant Applications.
=>In Non-Persistant Applications Development, We accept the data from Key Board, Stored in main memory(temporary data) , processed and results are displayed and shown on the moniter".
=>We know that Main Memory is a Temprary Memory and whose data is volatile.
=>Since Data is an important for organization for making effective decisions, so that data must stored permananetly .
=>In Persistant Applications Development, We accept the data from Key Board, Stored in main memory(temporary data) , processed and whose results must stored permananetly.
=>In Industry we have two approaches to store the data permanently. They are
    a) By using Files
    b) By Using Data Base Softwares (Oracle, MySQL,.....etc)
==============================x================================

=====================================
Types of Files
====================================
=>In Python Programming, we have two types of Files. They are
    a) Text Files
    b) Binary Files

----
a) Text Files:

=>Text Files contains Alphabets, Digits, Special Symbols only and it is human
    readable.
=>Text files are denoted by a 't'
=>By default a file is taken as Text File.
Example: .doc   .py   .cpp   .xlsx, txt ......etc

----
b) Binary Files:

=>A Binary Files contains the data in the form of Binary Format and it is machine readable.
=>Binary File is denoted by letter 'b'
=>Examples: images (.gif, jpeg, jpg, png..etc)
                          audio files
                   video files, MP3...etc
                   PDF files....
==============================X==================================

Operations on Files

=>On Files , we can perform 2 types of Operations. They are
1. Write Operation
2. Read Operation

----
1. Write Operation:

=>The purpose of write operation is that "To transfer Temporary data from main memory into file of secondary memory".
=>Steps:
1. Choose the File Name
2. Open the File Name in write mode.
3. Perform Cycle of Write Operations.

=>While we are performing Write Operations, we get some exceptions. They are
a) FileExistError
b) IOError

----
2. Read Operation:

=>The purpose of Read Operation is that " To read the data from file of secondary into object of main memory."
=>Steps:
1) Choose the file name.
2) Open the file name in read mode.
3) Perform cycle of read operations.

=>While we are performing Read Operations, we get some exceptions. They are
a) FileNotFoundError
b) EOFError

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

File Opening Modes

=>To perform read and write operations on files, we use file opening modes.
=>In Python, we have 7 file opening modes. They are
----
1) r :

=>This mode is used for opening the file in read mode and we can perform read operation.
=>It is one default file opening mode.
----

2) w

=>This mode is used for opening the file always in write mode newly irrespective
of new or existing file.
=>If the file already exist then existing data of the file overlapped with new data.
----
3) a

=>This mode is used for appending the data (Writing the data)
=>If we open the new file in 'a' mode then new data written to the file from the
beginning.
=>If we open the existing file in 'a' mode then new data added at the end of existing data (called Appending)
----
4) r+

=>This mode is used for Opening the File Read Mode.
=>When we open the file r+ mode then first we must perform read operation and
later we can perform write operation.
----
5) w+

=>This mode is used for opening the file always in write mode newly irrespective
of new or existing file.
=>If the file already exist then existing data of the file overlapped with new data.
=>With this mode additionally, we can perform read operation after performing
write Operations.
----
6) a+

=>This mode is used for appending the data (Writing the data)
=>If we open the new file in 'a+' mode then new data written to the file from the
begining.
=>If we open the existing file in 'a+' mode then new data added at the end of existing data (called Appending)
=>With this mode additionally, we can perform read operation after performing
write Operations.
----
7) x

=>This mode is used for Opening any New File in Write Mode Exclusively.
=>If the File already exists and if we open such file in 'x' mode then we get "FileExistError".
============================X================================

Number of approaches to Open the Files

=>To open the file for performing operations , we have 2 syntaxes. They are

1) By using open()
2) By using " with open() as " .

---

1) By using open():

Syntax:- varname=open("FileName", "File Mode")

Explanation:

=>'varname' is an object of type < class, '_TextIoWrapper'> and it acts as File Pointer.
=>open() is a Pre-defined Function used for opening the specified file name in specified file mode.
=>File Name represents Name of the file
=>File Modes can be either r, w, a, r+,w+,a+ and x.
=>When we open the file with this approach, we must close the file explicitly by using close() (Manual Closing of files) . This approach is unable to provide auto-closing the files (or) auto-closable files.

Examples:

```
#FileOpenEx1.py
try:
      fp=open("stud.info","r")
except FileNotFoundError:
      print("File does not Exists")
else:
      print("-"*50)
      print("File Opened in Read Mode Successfully")
      print("Type of fp=",type(fp))
      print("-"*50)
      print("File Name=",fp.name) # Gives Name of File
      print("File Mode=", fp.mode) # Gives  File Opening Mode- r
      print("is stud.info readable?=",fp.readable()) # True
      print("is stud.info writedable?=",fp.writable()) # False
      print("is stud.info closed?=",fp.closed) # False
      print("-"*50)
finally:
      print("\ni am from finally block")
      fp.close() # Manual Closing of file
      print("is stud.info closed?=",fp.closed) # True
```

==

2) By using " with open() as " .

Syntax:-

```
                     with     open("File Name","File Mode") as  VarName:
                         ------------------------------------------------
                      ---------Block of statements----------------
                      ---------Operations on Files----------------
```

Other Statements in Program

Exaplanation:

=>'with' and 'as' are the Key words
=> Open() is pre-defined Function used to open the file in specified file mode.
=>File Name represents Name of the file
=>File Modes can be either r, w, a, r+,w+,a+ and x.
=>VarName represents an object of type < class, '_TextIoWrapper'> and it acts as
    File Pointer.
=>As Long as PVM executes Block of statements witten within "with open() as " indentation block, file is active(Open) and once PVM comes out of Corresponding Indetation block then automatically File will be Closed(Known as Auto closeable)
=>The advantage of "with open() as " approach is that Auto closeable property (no need to close the file manuvally in finally block)

Examples:

```
with open("stud.info","w") as wp:
      print("-"*50)
      print("File  Opened  in Exclusively in Write Mode:")
      print("Type of wp=",type(wp))
      print("-"*50)
      print("File Name=",wp.name) # Gives Name of File
      print("File Mode=", wp.mode) # Gives  File Opening Mode
      print("is stud.info readable?=",wp.readable())
      print("is stud.info writedable?=",wp.writable())
      print("is stud.info closed?=",wp.closed) # False
      print("-"*50)
print("\nis stud.info closed after indentation of with open()?=",wp.closed) # True
```

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

==
Writing the data to the file
==
=>To write the data to the file, we have two pre-defined functions. They are
            a) write()
            b) writelines()
----
a) write()

=>This Function is used for writting any type of data to the file in the form of str.

=>Syntax:-   filepointer.write(str data)

Examples:

#This program writes address of different people in addr.info file--write()

```
#FileWriteEx1.py
with  open("addr.info", "a") as wp:
      #write the adress of Rossum
      wp.write("Dennis Ritche\n")
      wp.write("13-14, Green Port \n")
      wp.write("Bell Labs--USA\n")
      print("\nAddress written to the file successfully--verify")
```

===

b) writelines()   :

=>This function is used for writting any iterable object data to the file in the form of str only.

=>Syntax:-   filepointer.writelines(Iterableobject)

Examples:

```
#This program writes iterable objects to the file---writelines()
#FileWriteEx2.py
lst=[10,"Nags",33.33,"Python"]
with open("stud.addr","a") as fp:
      fp.writelines(str(lst)+"\n" )
      print("\nIterable object data written to the file:")
```

```
#This program writes iterable objects to the file---writelines()
#FileWriteEx2.py
tpl=(20,"Ganesh",63.33,"Java")
with open("stud.addr","a") as fp:
      fp.writelines(str(tpl)+"\n" )
      print("\nIterable object data written to the file:")
```

```
#This program writes iterable objects to the file---writelines()
#FileWriteEx2.py
s={30,"Ankit",23.33,"C"}
with open("stud.addr","a") as fp:
      fp.writelines(str(s)+"\n" )
      print("\nIterable object data written to the file:")
```

```
#This program writes iterable objects to the file---writelines()
#FileWriteEx2.py
d={10:"Python",20:"Django",30:"Java"}
with open("stud.addr","a") as fp:
      fp.writelines(str(d)+"\n" )
      print("\nIterable object data written to the file:")
```

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

```
"""
E:\KVR-PYTHON-4PM\FILES>type stud.addr
[10, 'Nags', 33.33, 'Python']
(20, 'Ganesh', 63.33, 'Java')
{'Ankit', 23.33, 'C', 30}
```

```
{10: 'Python', 20: 'Django', 30: 'Java'}
"""
#FileOpenEx1.py
try:
      fp=open("stud.info","r")
except FileNotFoundError:
      print("File does not Exists")
else:
      print("-"*50)
      print("File Opened in Read Mode Successfully")
      print("Type of fp=",type(fp))
      print("-"*50)
      print("File Name=",fp.name) # Gives Name of File
      print("File Mode=", fp.mode) # Gives  File Opening Mode- r
      print("is stud.info readable?=",fp.readable()) # True
      print("is stud.info writedable?=",fp.writable()) # False
      print("is stud.info closed?=",fp.closed) # False
      print("-"*50)
finally:
      print("\ni am from finally block")
      fp.close() # Manual Closing of file
      print("is stud.info closed?=",fp.closed) # True


#FileOpenEx2.py
wp=open("stud.info","w")
print("-"*50)
print("File Created and Opened in Write Mode:")
print("Type of wp=",type(wp))
print("-"*50)
print("File Name=",wp.name) # Gives Name of File
print("File Mode=", wp.mode) # Gives  File Opening Mode- w
print("is stud.info readable?=",wp.readable()) # False
print("is stud.info writedable?=",wp.writable()) # True
print("is stud.info closed?=",wp.closed) # False
print("-"*50)

#FileOpenEx3.py
wp=open("hyd.data","a+")
print("-"*50)
print("File  Opened  in Write Mode:")
print("Type of wp=",type(wp))
print("-"*50)
print("File Name=",wp.name) # Gives Name of File
print("File Mode=", wp.mode) # Gives  File Opening Mode
print("is stud.info readable?=",wp.readable())
print("is stud.info writedable?=",wp.writable())
print("is stud.info closed?=",wp.closed) # False
print("-"*50)

#FileOpenEx4.py
try:
      wp=open("stud.info","x")
      print("-"*50)
      print("File  Opened  in Exclusively in Write Mode:")
      print("Type of wp=",type(wp))
      print("-"*50)
      print("File Name=",wp.name) # Gives Name of File
```

```
      print("File Mode=", wp.mode) # Gives  File Opening Mode
      print("is stud.info readable?=",wp.readable())
      print("is stud.info writedable?=",wp.writable())
      print("is stud.info closed?=",wp.closed) # False
      print("-"*50)
except FileExistsError:
      print("File already exist")

#FileOpenEx5.py
with open("stud.info","w") as wp:
      print("-"*50)
      print("File  Opened  in Exclusively in Write Mode:")
      print("Type of wp=",type(wp))
      print("-"*50)
      print("File Name=",wp.name) # Gives Name of File
      print("File Mode=", wp.mode) # Gives  File Opening Mode
      print("is stud.info readable?=",wp.readable())
      print("is stud.info writedable?=",wp.writable())
      print("is stud.info closed?=",wp.closed) # False
      print("-"*50)
#out of Indentation of  "with  open()  as"
print("\nis stud.info closed after indentation of with
open()?=",wp.closed) # True

#This program writes address of different people in addr.info file--
write()
#FileWriteEx1.py
with  open("addr.info", "a") as wp:
      #write the adress of Rossum
      wp.write("Dennis Ritche\n")
      wp.write("13-14, Green Port \n")
      wp.write("Bell Labs--USA\n")
      print("\nAddress written to the file successfully--verify")

#This program writes iterable objects to the file---writelines()
#FileWriteEx2.py
lst=[10,"Nags",33.33,"Python"]
with open("stud.addr","a") as fp:
      fp.writelines(str(lst)+"\n" )
      print("\nIterable object data written to the file:")

      #Program reading the data from KBD and write that data to the file.
#DynamicFileWriteEx.py
import sys
with open("hyd.data","a") as fp:
      print("Enter the Lines of Text and press 'quit' to stop")
      while(True):
            kbddata=input()
            if(kbddata=="quit"):
                  sys.exit()
            else:
                  fp.write(kbddata+"\n")
      print("\nData written to the file--verify")


      #This program reads the data from file and display on the console--
-read()
#FileReadEx1.py
```

```
try:
      filename=input("Enter any file name:")
      with open(filename) as fp:
            filedata=fp.read()
            print("-"*50)
            print("Content of file")
            print("-"*50)
            print(filedata)
            print("-"*50)
except FileNotFoundError:
      print(" '{}'  File does not exist:".format(filename))

      #This program reads specified number of chars from file--read(no.of
chars)
#FileReadEx2.py
with open("Hyd.data","r") as fp:
      print("Inital Index /Pos of fp={}".format(fp.tell())) # 0
      fdata=fp.read(3)
      print("File Data=",fdata) # HYD
      print("Now  Index /Pos of fp={}".format(fp.tell()))
      fdata=fp.read(15)
      print("File Data=",fdata) #
      print("Now  Index /Pos of fp={}".format(fp.tell()))
      fdata=fp.read()
      print("File Data=",fdata) #
      print("Now  Index /Pos of fp={}".format(fp.tell()))
      fp.seek(0)
      fdata=fp.read(18)
      print("File Data=",fdata) #
      print("Now  Index /Pos of fp={}".format(fp.tell()))

#This program reads one line at a time from file--readline()
#FileReadEx3.py
with open("Hyd.data","r") as fp:
      line=fp.readline()
      print(line)
      line=fp.readline()
      print(line)
      line=fp.readline()
      print(line)

#This program reads all the lines from file--readlines()
#FileReadEx4.py
filename=input("Enter File name:")
try:
      with open(filename,"r") as fp:
            filelines=fp.readlines()  # filelines is type  <class,'list'>
            for line in filelines:
                  print("{}".format(line),end="")
except FileNotFoundError:
      print("File does not exists:")

#Program for copying the content of one file into another file
#FileCopy.py
sfile=input("Enter Source File:")
try:
      with  open(sfile) as rp:
            dfile=input("Enter Destination File:")
```

```
            with open(dfile,"a") as wp:
                  sfiledata=rp.read()
                  wp.write(sfiledata)
                  print("\n'{}' file data copied into '{}'
file".format(sfile,dfile))
except FileNotFoundError:
      print("Source file does not exists")
```

==
Reading the data from the file
==

=>To Read the data from the file, we have 4 pre-defined functions. They are

a) read()
b) read(no. of chars)
c) readline()
d) readlines()

----

a) read():

=>This Function is used for reading entire content of file data in the form of str.

=>Syntax:- varname=filepointer.read()

Examples:

```
#This program reads the data from file and display on the console---
read()
#FileReadEx1.py
try:
      filename=input("Enter any file name:")
      with open(filename) as fp:
            filedata=fp.read()
            print("-"*50)
            print("Content of file")
            print("-"*50)
            print(filedata)
            print("-"*50)
except FileNotFoundError:
      print(" '{}'  File does not exist:".format(filename))
```

----

b) read(no. of chars):

=>This Function is used for reading spcified number of characters from the given file.

=>Syntax:- varname=filepointer.read(no.of chars)

Examples:

```
#This program reads specified number of chars from file--read(no.of
chars)
#FileReadEx2.py
with open("Hyd.data","r") as fp:
      print("Inital Index /Pos of fp={}".format(fp.tell())) # 0
      fdata=fp.read(3)
```

```
	print("File Data=",fdata) # HYD
	print("Now  Index /Pos of fp={}".format(fp.tell()))
	fdata=fp.read(15)
	print("File Data=",fdata) #
	print("Now  Index /Pos of fp={}".format(fp.tell()))
	fdata=fp.read()
	print("File Data=",fdata) #
	print("Now  Index /Pos of fp={}".format(fp.tell()))
	fp.seek(0)
	fdata=fp.read(18)
	print("File Data=",fdata) #
	print("Now  Index /Pos of fp={}".format(fp.tell()))
```
----

c) readline():

=>This function is used for reading one line at a time from file.

=>Syntax:-   varname=filepointer.readline()

Examples:

```
#This program reads one line at a time from file--readline()
#FileReadEx3.py
with open("Hyd.data","r") as fp:
	line=fp.readline()
	print(line)
	line=fp.readline()
	print(line)
	line=fp.readline()
	print(line)
```
----

d) readlines():

=>This function is used for reading all the lines from file in the form list

=>Syntax:-   listobj=filepointer.readlines()

Examples:

```
#This program reads all the lines from file--readlines()
#FileReadEx4.py
filename=input("Enter File name:")
try:
	with open(filename,"r") as fp:
		filelines=fp.readlines()  # filelines is type  <class,'list'>
		for line in filelines:
			print("{}".format(line),end="")
except FileNotFoundError:
	print("File does not exists:")
```

```
# This program counts number of lines, words and chars in a file
#Filecount.py
filename=input("Enter any file name:")
try:
```

```
	with open(filename, "r") as fp:
		print("-"*50)
		print("Content of File:")
		print("-"*50)
		for kvr in fp:
			print(kvr,end="")
		else:
			print("-"*50)
			nl=0
			nw=0
			nc=0
			fp.seek(0)
			lines=fp.readlines()
			for line in lines:
				nl=nl+1
				nw=nw+len(line.split())
				nc=nc+len(line)
			else:
				print("-"*50)
				print("Number of Lines=",nl)
				print("Number of words=",nw)
				print("Number of Chars=",nc)
				print("-"*50)
except FileNotFoundError:
	print("File does not exists")

#This program copy an image  by using files
#imagecopy.py
with open("C:\\KVR-HYD\\robo.png","rb") as fp:
	filedata=fp.read()
	with open("pythstudent.png","wb") as wp:
		wp.write(filedata)
		print("Image Copied --very")
```

==

Pickling and Un-Pickling

(OR)

Object Serialization or Object De-Serialization

==

Pickling

=>Let us assume there there exist an object which contains multiple values. To save or write object data of main memory into the file of secondary memory by using write() and writelines() , they transfers the values in the form of value by value and it is one of the time consuming process(miltiple write operations).

To Overcome this time consuming process, we must use the concept of Pickling.

=>The advantage of pickling concept is that with single write operation , we can save or write entire object data of main memory into the file of secondary memory.

=>Definition of Pickling:

=>The Process saving or transfering entire object content of main memory into the file of secondary memory by performing single write operation is called Pickling.
=>Pickling concept participates in Write Operations.

Steps for implementing Pickling Concept:

=>import pickle module, here pickle is one of the pre-defined module
=>Choose the file name and open it into write mode.
=>Create an object with collection of values (Iterable object)
=>use the dump() of pickle module. dump() save the content of any object into the
file with single write operation.
Syntax: pickle.dump(object,filepointer)
=>NOTE That pickling concept always takes the file in Binary Format.

Un-Pickling

=>Let us assume there exists a record with multiple values in a file of secondary memory. To read or trasfer the entire record content from file of secondary memory if we use read(), read(no.of chars), readline() and readlines() then they read record values in the form value by value and it is one of the time consuming process(multiple read operations).
=>To overcome time consuming process, we must use the concept of Un-pickling.
=>The advantange of Un-pickling is that with single read operation, we can read entire record content from the file of secondary memory into the object of main memory.
=>Definition of Un-Pickling:

=>The process of reading or trasefering the enrite records content from file of secondary memory into the object of main memory by performing single read operation is called Un-pickling.

=>Un-Pickling concept participates in Read Operations.

Steps for implementing Un-Pickling Concept:

=>import pickle module
=>Choose the file name and open it into read mode.
=>Use the load() of pickle module. load() is used for trasfering or loading the
entire record content from file of secondary memory into object of main memory.
Syntax: objname=pickle.load(filepointer)
=>NOTE That Un-pickling concept always takes the file in Binary Format.

```
#Accept employee details and save them file by using pickling
#emppick.py
import pickle
noe=int(input("Enter How many employees data u have:"))
if(noe<=0):
      print("{} invalid number of employees:".format(noe))
else:
      #open the file write mode
```

```
      with open("emp.data","ab") as fp:
            for i in range(1,noe+1):
                  print("-"*50)
                  print("\nEnter {} Employee Details:".format(i))
                  print("-"*50)
                  #accept employee details
                  eno=int(input("\tEnter Employee Number:"))
                  ename=input("\tEnter Employee Name:")
                  sal=float(input("\tEnter Employee Salary:"))
                  #create an empty list
                  lst=list()
                  #append employee data to list
                  lst.append(eno)
                  lst.append(ename)
                  lst.append(sal)
                  #dump the lst data into file
                  pickle.dump(lst,fp)
                  print("-"*50)
                  print("\n{} Employee Record Saved Successfully in
file".format(i))
```

```
#This reads employee records from file by using un-pickling
#empunpick.py
import pickle
try:
      with open("emp.data","rb") as fp:
            print("-------------------------------------------")
            print("Emplyee Records")
            print("-------------------------------------------")
            while(True):
                  try:
                        obj=pickle.load(fp)
                        for val in obj:
                              print("{}".format(val),end="   ")
                        print()
                  except EOFError:
                        print("--------------------------------------")
                        break
except FileNotFoundError:
      print("Source File does not exists:")
```

```
#Program for accepting student no,name,marks and college name and save
them in file by using pickling
#studentpick.py----------------Program--(A)
import pickle
with open("stud.data","ab") as sp:
      while(True):
            print("-"*50)
            #accepting student details
            sno=int(input("Enter Student Number:"))
            sname=input("Enter Student Name:")
            marks=float(input("Enter Student Marks:"))
            uname=input("Enter Student University Name:")
            #create an empty list
            l=list()
```

```
            #append student values to l
            l.append(sno)
            l.append(sname)
            l.append(marks)
            l.append(uname)
            #save object l in file
            pickle.dump(l,sp)
            print("-"*50)
            print("Student Record Saved in a file:")
            print("-"*50)
            ch=input("Do u want to insert another Record(yes or no):") #
hyd
            if(ch.upper()=="NO"):
                  print("Thanks for using this  program")
                  break
            if(ch.lower()!="yes"):
                  print("Enter 'yes' for continuing the data to insert ")
                  break
```

```
#Program for reading the records from file by using un-pickling
#studentunpick.py----------------Program--(B)
import pickle
try:
      with open("stud.data","rb") as fp:
            print("-"*50)
            print("Student details")
            print("-"*50)
            while(True):
                  try:
                        obj=pickle.load(fp)
                        for val in obj:
                              print("{}".format(val),end="  ")
                        print()
                  except EOFError:
                        print("-"*50)
                        break
except FileNotFoundError:
      print("File does not exists")
```

==
Random Access files in Python
==

=>To access the data of the file randomly, we use to two function, where they can point to the data file. They are

1) tell()
2) seek()

1) tell():

=>This Function will give index of file pointer where it is pointing in data of file.

=>Syntax:-   Index=filepointer.tell()

----

2) seek():

=>This function makes the file pointer to point to the specfied index in the data of file.

```
=>Syntax:-       filepointer.seek(Index)
----------------------------------------------------------------
Examples:
-------------------------
Hyd.data---File Name
--------------
Hyd is the capital of TS
In HYD , there is ammerpet
which is hub of IT Courses
and Python is one trending lang
Python class now going on files
------------------------------------------------------
#This program reads specified number of chars from file--read(no.of
chars)
#FileReadEx2.py
with open("Hyd.data","r") as fp:
      print("Inital Index /Pos of fp={}".format(fp.tell())) # 0
      fdata=fp.read(3)
      print("File Data=",fdata) # HYD
      print("Now  Index /Pos of fp={}".format(fp.tell()))
      fdata=fp.read(15)
      print("File Data=",fdata) #
      print("Now  Index /Pos of fp={}".format(fp.tell()))
      fdata=fp.read()
      print("File Data=",fdata) #
      print("Now  Index /Pos of fp={}".format(fp.tell()))
      fp.seek(0)
      fdata=fp.read(18)
      print("File Data=",fdata) #
      print("Now  Index /Pos of fp={}".format(fp.tell()))
------------------------------------------------------------------------
""" OUTPUT:

E:\KVR-PYTHON-4PM\FILES>py FileReadEx2.py
Inital Index /Pos of fp=0
File Data= Hyd
Now  Index /Pos of fp=3
File Data=  is the capital
Now  Index /Pos of fp=18
File Data=  of TS
In HYD , there is ammerpet
which is hub of IT Courses
and Python is one trending lang
Python class now going on files

Now  Index /Pos of fp=148
File Data= Hyd is the capital
Now  Index /Pos of fp=18  """
==========================================
```

======================================
OS module
======================================

=>'os' is one of the pre-defined module
=>The purpose of "os" module is that "To perform certain Operating System Based Operations".
=>Some of the OS based Operations are
a) obtaining current working folder / directory (getcwd())

```
            b) create a folder / directory   ( mkdir()  )
            c) create  folders / directories ( makedirs() )
            d) remove folder ( rmdir() )
            e) remove folders ( removedirs() )
            f) renaming the folder ( rename() )
            g) obtain the files in a folder...etc ( listdir() )
```
----
a) obtaining current working folder / directory:

=>For obtaining current working folder, we use   getcwd()
Syntax: varname=os.getcwd()

```
#cwdname.py
import os
fname=os.getcwd()
print("current working folder=",fname)
```
==========================
b) create a folder / directory :

=>To create a folder , we mkdir()
=>Syntax: os.mkdir("FolderName")
=>This function can create only one folder at a time but not able to create multiple
    folders.
=>If the folder already exists and if we create then we get FileExistError

Examples:

```
import os
try:
      os.mkdir("D:\INDIA"")
      print("Folder Created Successfully:")
except FileExistsError:
      print("Folder already exists")
except FileNotFoundError:
      print("We can't crate Root , sub or sub-sub folders")
```
----
d) create folders / directories :

=>To crate Folders at a time , we use makedirs() .
=>Syntax:-   os.makedirs("Folders Hierarchy")
=>Here Folders Hierarchy represents Root Folder, Sub Folder, Sub Sub Folders
     etc.
=>If Folders Hierarchy already exists and if we create again then we get
    FileExistError.
----
d) remove folder :

=>To remove a folder, we rmdir() .
=>Syntax:-   os.rmdir("Folder Name")
=>This Function can remove one folder at a time but not able to remove Folders
    Hierarchy.

```
# program for  remove a folder / directory   ( rmdir()  )
#removefolder.py
import os
try:
      os.rmdir("C:\KVR")
      print("Folder Removed Successfully:")
except FileNotFoundError:
      print("No folder exists")
except OSError:
      print("This Folder is not empty")
```

===

e) remove folders:

=>To remove Folders Hierarchy , we use removedirs() .
=>Syntax:-   os.removedirs("Folders Hierarchy")
=>Folders Hierarchy represents Root Folder , Sub Folder, sub-sub Folder etc.
=>if Folders Hierarchy contains files then we get OSError.
=>If Folders Hierarchy does not exist then we get FileNotFoundError.

Examples:

```
# program for  remove a folders --removedirs()
#removefolders.py
import os
try:
      os.removedirs("D:\INDIA\HYD\python")
      print("Folders Removed Successfully:")
except FileNotFoundError:
      print("Folders does not  exists")
except OSError:
      print("This Folders are not empty")
```

==

f) renaming the folder :

=>To Rename a folder, we use rename()
=>Syntax:-   os.rename("Old Folder Name","New FolderName")
=>If old folder name does not exist then we get FileNotFoundError.
=>If Old Folder Name exists then Old Folder Name replaced with New Folder Name.

Examples:

```
# program for RENAMING a   folder / directory   ( rename() )
#renamefolder.py
import os
try:
      os.rename("C:\Rossum","C:\Ross")
      print("Folder Renamed Successfully:")
except FileNotFoundError:
      print("No such folder exists")
```

----

g) obtain the files in a folder...etc :

=>To obtain the files in a folder , we use listdir().
=>Syntax:-   os.listdir("folder name")
=>If folder name does not exists then we get FileNotFoundError.

Examples:

```
# program for listing files in folders.
#fileslist.py
import os
try:
      fileslist=os.listdir("E:\KVR-PYTHON-4PM\FILES")
      print("---------------------------------------------")
      print("File Names :")
      print("---------------------------------------------")
      for file in fileslist:
            print("\t{}".format(file))
      else:
            print("---------------------------------------------")
            print("Number of Files={}".format(len(fileslist)))
            print("---------------------------------------------")

except FileNotFoundError:
      print("No such folder exists")
```

```
# program for listing files in current folder
#currentfolderfileslist.py
import os
try:
      fileslist=os.listdir(".") # here dot (.) represents current working
folder
      print("---------------------------------------------")
      print("File Names :")
      print("---------------------------------------------")
      for file in fileslist:
            print("\t{}".format(file))
      else:
            print("---------------------------------------------")
            print("Number of Files={}".format(len(fileslist)))
            print("---------------------------------------------")
except FileNotFoundError:
      print("No such folder exists")


      # program for listing files in folders.
#currentfolderfileslist.py
import os
try:
      fileslist=os.listdir(".") # here dot (.) represents current working
folder
      print("---------------------------------------------")
      print("File Names :")
      print("---------------------------------------------")
      for file in fileslist:
            print("\t{}".format(file))
      else:
            print("---------------------------------------------")
            print("Number of Files={}".format(len(fileslist)))
            print("---------------------------------------------")

except FileNotFoundError:
```

```
      print("No such folder exists")

      #Program for  obtaining current working folder / directory
#cwdname.py
import os
fname=os.getcwd()
print("current working folder=",fname) # E:\KVR-PYTHON-4PM\OS MODULE>


"""
E:\KVR-PYTHON-4PM\OS MODULE>py cwdname.py
current working folder= E:\KVR-PYTHON-4PM\OS MODULE
"""

# program for listing files in folders.
#fileslist.py
import os
try:
      fileslist=os.listdir("E:\KVR-PYTHON-4PM\FILES")
      print("---------------------------------------------")
      print("File Names :")
      print("---------------------------------------------")
      for file in fileslist:
            print("\t{}".format(file))
      else:
            print("---------------------------------------------")
            print("Number of Files={}".format(len(fileslist)))
            print("---------------------------------------------")

except FileNotFoundError:
      print("No such folder exists")

      # program for  create a folder / directory   ( mkdir()  )
#foldercreate.py
import os
try:
      os.mkdir("D:\INDIA"")
      print("Folder Created Successfully:")
except FileExistsError:
      print("Folder already exists")
except FileNotFoundError:
      print("We can't crate Root , sub or sub-sub folders")

# program for  create a folders / directories   ( makedirs()  )
#folderscreate.py
import os
try:
      os.makedirs("D:\INDIA\HYD\python")
      print("Folders  Created Successfully:")
except FileExistsError:
      print("Folder already exists")
# program for  remove a folder / directory   ( rmdir()  )
#removefolder.py
import os
try:
      os.rmdir("C:\KVR")
      print("Folder Removed Successfully:")
except FileNotFoundError:
```

```
      print("No folder exists")
except OSError:
      print("This Folder is not empty")


# program for  remove a folders --removedirs()
#removefolders.py
import os
try:
      os.removedirs("D:\INDIA\HYD\python")
      print("Folders Removed Successfully:")
except FileNotFoundError:
      print("Folders does not  exists")
except OSError:
      print("This Folders are not empty")

# program for RENAMING a   folder / directory   ( rename() )
#renamefolder.py
import os
try:
      os.rename("C:\Rossum","C:\Ross")
      print("Folder Renamed Successfully:")
except FileNotFoundError:
      print("No such folder exists")
```

```
            ========================================
                  random module
            ========================================
```

=>random one of pre-defined module present in python
=>The purpose of random is that "To generate random values in various contexts".
=>random module contains the follwoing essential functions.

a) randrange()
b) randint()
----
c) random()
d) uniform()
----
e) choice()
f) shuffle()
g) sample()

==

a) randrange()

=>This function is used for generating random integer values between specified limits.

Syntax1:-   random.randrang(Value)
            This syntax generates any random value between 0 to Value-1

Syntax-2:   random.rangerange(start,stop)
            This syntax generates any random value between start to stop-1

Examples:

```
>>> import random
```

```
>>> print(random.randrange(100,150))----133
>>> print(random.randrange(100,150))----121
>>> print(random.randrange(100,150))----139
>>> print(random.randrange(100,150))----143
>>> print(random.randrange(100,150))---106
>>> print(random.randrange(100,150))---133
>>> print(random.randrange(10))----5
>>> print(random.randrange(10))----9
-----------------------------------------------------
#randrangeex.py
import random
for i in range(1,6):
      print(random.randrange(10))
print("---------------------------------------")
for i in range(1,6):
      print(random.randrange(1000,1100))
print("---------------------------------------")
================================X============================
```

b) randint():

=>Syntax:-   random.radint(start,stop)

=>This syntax generates any random value between start to stop. Here start and stop are inclusive.

Examples:

```
>>> print(random.randint(10,15))------10
>>> print(random.randint(10,15))-----13
>>> print(random.randint(10,15))----14
>>> print(random.randint(10,15))----11
>>> print(random.randint(10,15))----15
----------------------------------------------------------
#randintex.py
import random
for i in range(1,6):
      print(random.randint(10,20))
print("---------------------------------------")
==============================X================================
```

c) random()

=>Syntax:-   random.random()

=>This syntax generates floating point random values between 0.0 and 1.0 (Exlusive))

Examples:

```
>>> import random
>>> print(random.random())----------0.1623906138450063
>>> print(random.random())--------0.15382209709271966
>>> print(random.random())-------0.09542283007844476
>>> print(random.random())-----0.6134301633766425
-------------------------
#randomex.py
import random
lst=[]
for i in range(1,6):
      lst.append("%0.2f" %random.random())
print("---------------------------------------")
print("Content of lst={}".format(lst))
============================X==============================
```

d) uniform()

Syntax:-   random.uniform(start,stop)
=>This generates random floting point values from start to stop-1 values
=>The values of start and stop can both Integer or floating point values.
Examples:

```
>>> import random
>>> print(random.uniform(10,15))----------14.416746067678286
>>> print(random.uniform(10,15))----13.2420406264978
>>> print(random.uniform(10,15))-----11.716110933506432
>>> print(random.uniform(10,15))--------10.703499588966528
>>> print(random.uniform(10,15))-----11.306226559323017
>>> print(random.uniform(10.75,15.75))--------13.939787347170148
>>> print(random.uniform(10.75,15.75))----10.760428232717597
```

```
#uniformex.py
import random
lst=[]
for i in range(1,6):
      lst.append(float("%0.2f" %random.uniform(10,15.5)))
print("----------------------------------------")
print("Content of lst={}".format(lst))
```

===========================X================================
e) choice():

Syntax:-   random.choice(Iterable_object)
=>This function obtains random values from Iterable_object.

EXAMPLES:

```
>>>
print(random.choice([10,20,30,40,50]),random.choice("PYTHON"),random.choice(range(10,15)))---40 T 11
>>>
print(random.choice([10,20,30,40,50]),random.choice("PYTHON"),random.choice(range(10,15)))----------30 P 12
>>>
print(random.choice([10,20,30,40,50]),random.choice("PYTHON"),random.choice(range(10,15)))-----------40 N 12
```

```
#choiceex.py
import random
s="AaBRe#^%@8YuQLPau*&"
for i in range(1,6):

print(random.choice(s),random.choice(s),random.choice(s),random.choice(s))
```

===========================X================================
f) shuffle():

=>This Function is used for re-organizing the elements of any mutable object.

Syntax:-   random.shuffle(list)
=>We can shuffle the data of list but not other objects of Data Types
Examples:

```
-------------------
>>> d={10:"cadburry",20:"kitkat",30:"malkybar", 40:"dairymilk"}
>>> print(d)---{10: 'cadburry', 20: 'kitkat', 30: 'malkybar', 40:
'dairymilk'}
>>> for k,v in d.items():
...     print(k,"--",v)
...
      10 -- cadburry
      20 -- kitkat
      30 -- malkybar
      40 -- dairymilk
>>> import random
>>> print(random.shuffle(d))----Traceback (most recent call last):
                                          File "<stdin>", line 1, in
<module>
                                          File
"C:\Users\nareshit\AppData\Local\Programs\Python\Python310\lib\random.py"
, line 394, in shuffle
                                            x[i], x[j] = x[j], x[i]
                                        KeyError: 3
>>> s={10,20,30,40,50}
>>> print(random.shuffle(s))
                                  Traceback (most recent call last):
                                    File "<stdin>", line 1, in <module>
                                    File
"C:\Users\nareshit\AppData\Local\Programs\Python\Python310\lib\random.py"
, line 394, in shuffle
                                      x[i], x[j] = x[j], x[i]
                                  TypeError: 'set' object is not
subscriptable

>>> t=(10,20,30,40,50)
>>> print(random.shuffle(t))
                                  Traceback (most recent call last):
                                    File "<stdin>", line 1, in <module>
                                    File
"C:\Users\nareshit\AppData\Local\Programs\Python\Python310\lib\random.py"
, line 394, in shuffle
                                      x[i], x[j] = x[j], x[i]
                                  TypeError: 'tuple' object does not
support item assignment
>>> l=[10,20,30,40,50]
>>> print(random.shuffle(l))-----None
>>> print(l)-------------[30, 40, 50, 10, 20]
>>> random.shuffle(l)
>>> print(l)------------[40, 30, 10, 20, 50]
>>> random.shuffle(l)
>>> print(l)---------[40, 10, 50, 20, 30]
>>> random.shuffle(l)
>>> print(l)------------[30, 50, 20, 40, 10]

#shuffleex.py
import random as r
l=[10,"Python","Rossum",34.56,True]
for i in range(1,6):
      r.shuffle(l)
      print("content of l=",l)
===================================X=============================
```

g) sample()

=>This Function is used for selecting random samples from any Iterable object based on number of samples(+ve)
Syntax:-   random.sample(iterable_object, k)
=>Here 'k' can be number of samples.

Examples:

```
>>> import random
>>> s="ABCabcERTYUertyu$%^&*#@!%^&ghjkiyl"
>>> print(random.sample(s,5))----------['A', '*', '^', 'j', 't']
>>> print(random.sample(s,5))---------['%', 'l', 'b', 'C', 'y']
>>> print(random.sample(s,5))----------['%', 'e', 'Y', 'j', 'u']
>>> print(random.sample(s,5))------['y', 'E', '&', '$', '#']
>>> print(random.sample(s,5))----------['j', '*', 't', '$', 'u']
```

```
#sampleex.py
import random
lst=[10,"Rossum","Python",34.56,True]
for i in range(1,6):
      print(random.sample(lst,2))
```

Examples:
```
#sampleex.py---sample()
import random
s="abcABC455678#$%@wertyKLMNHO"
s2="ABCDEFGHIJKLMNOPQRSTUVWXYZ"
for i in range(1,6):
      print(random.sample(s,6))
print("-----------------------------------")
for i in range(1,6):
      s1=" "
      s1=s1.join(random.sample(s,6))
      print(s1)
print("-----------------------------------")
for i in range(1,6):
      s1=" "
      s1=s1.join(random.sample(s2,6))
      print(s1)
```
================================X===========================

```
#choiceex.py
import random
s="AaBRe#^%@8YuQLPau*&"
for i in range(1,6):

print(random.choice(s),random.choice(s),random.choice(s),random.choice(s)
)
```

```
#randintex.py
import random as r
for i in range(1,6):
      print(r.randint(100,105))
```

```
#randomex.py
import random
lst=[]
for i in range(1,6):
      lst.append("%0.2f" %random.random())
print("---------------------------------------")
print("Content of lst={}".format(lst))


#randrangeex.py
import random as r
for i in range(1,10):
      print(r.randrange(100,150))


#sampleex.py
import random
lst=[10,"Rossum","Python",34.56,True]
for i in range(1,6):
      print(random.sample(lst,2))

#shuffleex.py
import random as r
l=[10,"Python","Rossum",34.56,True]
for i in range(1,6):
      r.shuffle(l)
      print("content of l=",l)

#uniformex.py
import random
lst=[]
for i in range(1,6):
      lst.append(float("%0.2f" %random.uniform(10,15.5)))
print("---------------------------------------")
print("Content of lst={}".format(lst))
```

==

Python DataBase Communication (PDBC)

==

=>Even we achieved the Data Persistency with Files concept, we have the following limitations.

1. Files Concept of any language does not contain security bcoz files concept does not contain user names and passwords.
2. To extract or process the data from files is very complex bcoz Files data must always processed with Indices.
3. The data of the files does not contain Column Names and Very complex to Process / Query the data
4. Files are unable to store large Volume of Data.
5. The Architecture of Files Changes from One OS to another OS (OR) Files are Dependent on OS.

=>To Overcome the limitations of Files, we must use the concept of DataBase Softwares Which are purely RDBMS Products(Oracle, MySQL, MongoDB,SQLITE3, DB2,SQL SERVER...........)

=>When we Data Base Softwares for acheving the data persistency, we get the following advantages.

1. DataBase Softwares are Fully Secured bcoz they provides User Name and password
2. To Process or Extract or Querying the data from DataBase Softwares is very easy bcoz the data present in tables of Database softwares are qualified with Column Names.
3. Data Base Software are able to store large Volume of Data.

4. Data Base Softwares are InDepeendent from OS.

--X--

=>If Python Program want to communicate with Any Database Software Product then we must use pre-defined modules and such pre-defined modules are not present in Python Library. So that Python Programmer must get / Install the pre-defined modules of Database Software by using a tool called pip.
=>To Make any Python Program to communicate with any data base software then we must install a third party module which is related Data base software.
=>For Example, To communicate with Oracle Database, we must install cx_Oracle Module, To communicate with MySQL data base , we must install mysql-connector.....etc and they must be installed explicitly by using pip tool

=>Syntax: pip install module name

=>here pip tool present in the following folder
"C:\Users\nareshit\AppData\Local\Programs\Python\Python310\Scripts "

=>If we get an error as " pip is not recognized internal command " then set the path as follows

C:\Users\nareshit>set path="C:\Users\nareshit\AppData\Local\Programs\Python\Python310\Scripts "

Example: intsall cx_Oracle
----

pip install cx_Oracle

Example: install mysql-connector

pip install mysql-connector

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

Communication between Python and Oracle DataBase

=>steps for developing a python program for communicating with Oracle database:

1. import cx_Oracle
2. Python Program must get the connection from Oracle Data Base.
3. Create an object of Cursor .

4. Design the Query, Place the Query in an object of Cursor and execute.
5. Python Program Process the Result
6. Python Program Closes the connection.

Explanation:
==========================

1. import   cx_Oracle

=>If python Program wants to communicate with Oracle data base then must import corresponding cx_Oracle module and it must be installed by using pip tool
=>Example: import cx_Oracle
=>Once the module is imported then Python Programmer ready to use Variable Names, Function Names and Class Names.

----

2. Python Program must get the connection from Oracle Data Base.

=>If a Python Program wants to perform some operations on Oracle data base then First we must get Connection from oracle data base.
=>If a Python Program wants get connection from Oracle data base then we must use connect() of cx_Oracle module.

Syntax:-   varname=cx_Oracle.connect("Connection url")
=>Varname is an object of <class,'cx_Oracle.connection">
=>cx_Oracle is pre-defined third party module used to communicate with Oracle Database.
=>connect() is one of the pre-defined function present in cx_Oracle module and it is used to get the connection from Oracle Data base.
=>The General format of Connection Url is
"UserName/Password@DNS/serviceid"
(OR)
=>The General format of Connection Url is

"UserName/Password@IPAddress/serviceid"
=>Here "user name " represents User Name of Oracle Data base (Ex: scott)
=>Here "password " represents pasword of Oracle Data base (Ex: tiger)
=>here DNS (Domain Naming Service) represents Name of Machine, where Oracle Database software installed. The default DNS of every computer is "localhost"
=>Here IPAddress (Internet Protocol Address) represents Address of a machine where Oracle Data base installed. The default IPAddress of every computer is "127.0.0.1"(Loop Back Address)
=>here serviceid represents on what name Oracle data base is installed (or) alias name Oracle Database in the current working machine.
=>Once Connection URL is wrong then we get DatabaseError of cx_Oracle and we handle that exception.
=>To find serviceid of Oracle Data base of any machine , goto SQL Prompt.

ORacle Enviroment:

```
SQL> select * from global_name;

           OUTPUT
           ---------------
```

```
                       GLOBAL_NAME
                       ---------------------------
                       ORCL  <-----Service Id
                       ---------------------------
```

Python Programming Env:

```
      import  cx_Oracle
      kvr=cx_Oracle.connect("scott/tiger@localhost/orcl")
                 (OR)
      kvr=cx_Oracle.connect("scott/tiger@127.0.0.1/orcl")
      print("Python Program got connection from Oracle DB")
```
----
3. Create an object of Cursor .

=>Here cursor is a pre-defined class present in cx_Oracle module.
=>The purpose of creating an object of cursor is that "To carry the Query from Python Program to Data base, Query Executed in Data base, Query result placed by Data base in the object of cursor and cursor gives result of the query to the Python Program".
=>Hence an object of cursor acts as driver between Python program and Database Software.
=>Programtically, to create an object of cursor, we must use cursor() which is present in connection object.
Syntax:-   varname=conobj.cursor()
=>here varname is an object of <class, 'cx_Oracle.cursor ' >
----
4. Design the Query, Place the Query in an object of Cursor and execute.

=>A Query is request / Question / statement to the data base from Python Program for perfoming certain Database Operations.
=>To execute the Query from Python Program, we must use execute(), which is present in cursor object.
=>Syntax:-   varname=cursorobj.execute("Query")
=>Here Query can be Either DDL or DML or DRL

```
#This Program get the connection and creates cursor object
#cursorobjex.py
import cx_Oracle  # step-1
con=cx_Oracle.connect("scott/tiger@localhost/orcl")  # step-2
print("\nPython Program obtained connection Oracle DB:")
cur=con.cursor() # step-3
print("\nType of cur variable=",type(cur)) # Type of cur variable= <class 
'cx_Oracle.Cursor'>
print("Cursor object created:")

#This program obtains Connection from Oracle Data base
#testoraclecon.py
import cx_Oracle  # step-1
try:
      con=cx_Oracle.connect("scott/tiger@localhost/orcl") # step-2
```

```
      print("\nType of con var=",type(con))#Type of con var= <class
'cx_Oracle.Connection'>
      print("Python Program got Connection from Oracle DB")
except cx_Oracle.DatabaseError as db:
      print("Problem in Connection:", db)
finally:
      con.close()
      print("\nPython closes the connection from Oracle db")

#This program obtains Connection from Oracle Data base
#testoraclecon1.py
import cx_Oracle  # step-1
try:
      con=cx_Oracle.connect("scott/tiger@127.0.0.1/orcl") # step-2
      print("\nType of con var=",type(con))# Type of con var= <class
'cx_Oracle.Connection'>
      print("Python Program got Connection from Oracle DB")
except cx_Oracle.DatabaseError as db:
      print("Problem in Connection:", db)
finally:
      con.close()
      print("\nPython closes the connection from Oracle db")
```

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

Types of Queries in Oracle

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

=>In Oracle, we have 3 types of Queries. They are

1) DDL statements (Data Definition Language)
   - a) create
   - b) alter
   - c) drop
2) DML statements (Data Manipulation Language)
   - a) insert
   - b) delete
   - c) update
6) DRL statements (Data Retrieval Language)
   - a) select

==

1) DDL statements (Data Definition Language)

==

=>The purpose of DDL statements is that "To create , alter and droping a table".

=>In Oracle , we have 3 types of DDL statements. They are

1) create
2) alter
3) drop

----

1) create :

=>"create " is used for creating a table on oracle database.

=>Syntax:-

SQL> create table table-name (Col1 Data Type, Col2 data type.....Col-n data type)

Example: create student table with sno,name and marks

SQL> create table student (sno number(3) primary key, name varchar2(10) not null, marks number(5,2) not null);

----

2) alter:

=>"alter" command is used for altering the table either by adding new column name or by changing column sizes.

=>Syntax1:-

    SQL> alter table table-name modify(existing col name data type)

=>Syntax2:-

    SQL> alter table table-name add(new col name data type)

Example: SQL> alter table teacher modify(tno number(4));

    SQL> alter table teacher add(sub varchar2(10));

----

3) drop:

=>"drop" command is used for droping or deleting the entire table.

=>Syntax:- drop table table-name

=>Example: delete / drop student table

    SQL> drop table student;

```
#This Program create a table from Python Program in Oracle database
#tabcreate.py
import cx_Oracle   # step-1
try:
      con=cx_Oracle.connect("scott/tiger@localhost/orcl") # step-2
      cur=con.cursor() # step-3
      #design and execute the query----Step-4
      qry="create table teacher(tno number(3) , name varchar2(10) ) "
      cur.execute(qry)
      print("\nTable created successfully in Oracle DB")
except cx_Oracle.DatabaseError as db:
      print("Problem in Data base:",db)
```

```
#This Python program drop / delete a table from python program
#tabledrop.py
import cx_Oracle
try:
      con=cx_Oracle.connect("scott/tiger@localhost/orcl")
      cur=con.cursor()
      cur.execute("drop table teacher")
      print("Teacher Table dropped--verify in Oracle")
except cx_Oracle.DatabaseError as db:
      print("Problem in Data base:",db)
```

```
#This Program alters(Modifying col names) a table from Python Program in
Oracle database
#altertablemodify.py
import cx_Oracle   # step-1
try:
      con=cx_Oracle.connect("scott/tiger@127.0.0.1/orcl")
      cur=con.cursor()
      qry="alter table employee modify(eno number(4) )"
      cur.execute(qry)
      print("Employee Table altered--verify in Oracle")
except cx_Oracle.DatabaseError as db:
      print("Problem in Data base:",db)
```

==

2) DML statements (Data Manipulation Language)

==

=>The purpose of DML statements in Database softwares is that "To Manipulate records of table."

=>Manipulating records of a table is nothing inserting records , deleting records and updating records.

=>In Database softwares , we have 3 types of DML statements . They are

a) insert
b) delete
c) update

=>When we use any DML statement, we must commit the data base by using commit() and rollback the DML operation we use rollback(). commit() and rollback() are present in connection object.

----

a) insert:

=>This statement is used for inserting a record into a table.

=>Syntax:-

SQL>insert into employee values(val1 for col1, val2 for col2...val-n col-n)

=>Examples:

SQL> insert into employee values(222,'Renuka',7.7,'TCS');

----

b) delete:

=>This command is used for deleting the records.

=>Syntax1:-   SQL> delete from table-name
                 SQL> delete from table-name where condition list

=>Example1:-   SQL> delete from employee

=>Example2:-   SQL> delete from employee where eno=555

=>Example3:- SQL> delete from employee where sal>4.0;

```
#This accept the values of employee and insert into employee table
#empinsertex1.py
import cx_Oracle
try:
      con=cx_Oracle.connect("scott/tiger@localhost/orcl")
```

```
	cur=con.cursor()
	#design the query and execute
	qry="insert into employee values (444,'DR',4.7,'BELL Labs') "   #
DML Query
	cur.execute(qry)
	con.commit()
	print("Employee Record inserted in employee table:")
except cx_Oracle.DatabaseError as db:
	print("Problem in Database:",db)


#This accept the values of employee from KBD and insert into employee
table
#empinsertex2.py
import cx_Oracle
try:
	con=cx_Oracle.connect("scott/tiger@localhost/orcl")
	cur=con.cursor()
	#accept employee values
	empno=int(input("Enter Employee Number:"))
	ename=input("Enter Employee Name:")
	esal=float(input("Enter Employee Salary:"))
	compname=input("Enter Employee Company Name:")
	#design and execute query
	iq="insert into employee values (%d,'%s',%f,'%s')"
	cur.execute(iq %(empno,ename,esal,compname) )
	# (OR) cur.execute("insert into employee values (%d,'%s',%f,'%s')"
%(empno,ename,esal,compname) )
	con.commit()
	print("{}  Record Inserted Successfully in employee
table".format(cur.rowcount))

except cx_Oracle.DatabaseError as db:
	print("Problem in Database:",db)


#This accept the values of employee from KBD and insert into employee
table
#empinsertex3.py
import cx_Oracle,sys
while(True):
	try:
		con=cx_Oracle.connect("scott/tiger@localhost/orcl")
		cur=con.cursor()
		#accept employee values
		empno=int(input("\nEnter Employee Number:"))
		ename=input("Enter Employee Name:")
		esal=float(input("Enter Employee Salary:"))
		compname=input("Enter Employee Company Name:")
		#design and execute query
		iq="insert into employee values (%d,'%s',%f,'%s')"
		cur.execute(iq %(empno,ename,esal,compname) )
		# (OR) cur.execute("insert into employee values
(%d,'%s',%f,'%s')" %(empno,ename,esal,compname) )
		con.commit()
		print("{}  Record Inserted Successfully in employee
table".format(cur.rowcount))
		print("-"*50)
```

```
            ch=input("Do u want to insert another record(yes/no):")
            if(ch.lower()=="no"):
                  print("\nThanks for using this program")
                  sys.exit()

      except cx_Oracle.DatabaseError as db:
            print("Problem in Database:",db)

#This accept the values of employee from KBD and insert into employee
table
#empinsertex4.py---file name and treated as module name
import cx_Oracle,sys
def   empinsert():
      while(True):
            try:
                  con=cx_Oracle.connect("scott/tiger@localhost/orcl")
                  cur=con.cursor()
                  #accept employee values
                  empno=int(input("\nEnter Employee Number:"))
                  ename=input("Enter Employee Name:")
                  esal=float(input("Enter Employee Salary:"))
                  compname=input("Enter Employee Company Name:")
                  #design and execute query
                  iq="insert into employee values (%d,'%s',%f,'%s')"
                  cur.execute(iq %(empno,ename,esal,compname) )
                  # (OR) cur.execute("insert into employee values
(%d,'%s',%f,'%s')" %(empno,ename,esal,compname) )
                  con.commit()
                  print("{}  Record Inserted Successfully in employee
table".format(cur.rowcount))
                  print("-"*50)
                  ch=input("Do u want to insert another record(yes/no):")
                  if(ch.lower()=="no"):
                        print("\nThanks for using this program")
                        sys.exit()
            except cx_Oracle.DatabaseError as db:
                  print("Problem in Database:",db)


#empdemo.py
from empinsertex4 import empinsert
empinsert()

#This deletes the record from employee table
#empdeleteex1.py
import cx_Oracle
try:
      con=cx_Oracle.connect("scott/tiger@localhost/orcl")
      cur=con.cursor()
      #design the query and execute
      cur.execute("delete from employee where eno=333")
      con.commit()
      if(cur.rowcount>0):
            print("Employee Record removed ")
      else:
            print("Employee Record does not exists")
except cx_Oracle.DatabaseError as db:
      print("Problem in Database:",db)
```

===
2) DML statements (Data Manipulation Language)
===

=>The purpose of DML statements in Database softwares is that "To Manipulate records of table."

=>Manipulating records of a table is nothing inserting records , deleting records and updating records.

=>In Database softwares , we have 3 types of DML statements . They are

a) insert
b) delete
c) update

=>When we use any DML statement, we must commit (permanent changes)the data base by using commit() and rollback (undo the changes) the DML operation we use rollback(). commit() and rollback() are present in connection object.

---

a) insert:

=>This statement is used for inserting a record into a table.

=>Syntax:-

```
SQL>insert into employee values( val1 for col1, val2 for col2...val-n col-n)
```

=>Examples:

```
SQL> insert into employee values(222,'Renuka',7.7,'TCS');
```

---

b) delete:

=>This command is used for deleting the records.

```
=>Syntax1:-      SQL> delete from table-name
                    SQL> delete from table-name where condition list
=>Example1:-    SQL> delete from employee
=>Example2:-    SQL> delete from employee where eno=555
=>Example3:- SQL> delete from employee where sal>4.0;
```

---

c) update:

=>This command is used for updating the record values of table

=>Syntax: (Updating All Records)

```
SQL> update  TableName
          set
ExistingColName1=Expression1,ExistingColName2=Expression2...
```

=>Syntax: (Updating Perticular records)

```
SQL> update  TableName
          set
ExistingColName1=Expression1,ExistingColName2=Expression2...
       where  condition list
```

Example:

```
SQL> update  employee set sal=sal+sal*(10/100);
---------------------------
SQL> update employee set sal=sal+sal*(20/100) where eno=600;
---------------------------------------------------------------------------
---------------------------------------
```

```
#This Program updates sal and company name based employee number
#empupdate.py--file name and acts as module
import cx_Oracle
def   updateemprecord():
      try:
            con=cx_Oracle.connect("scott/tiger@localhost/orcl")
            cur=con.cursor()
            #accept employee number, sal and company name
            empno=int(input("Enter Employee Number:"))
            empsal=float(input("Enter Employee Salary for update:"))
            empcname=input("Enter Employee Comp Name for update:")
            #design and  execute the query
            cur.execute("update employee set sal=%f, cname='%s' where
eno=%d" %(empsal,empcname,empno) )
            con.commit()
            if(cur.rowcount>0):
                  print("Employee Record Values updated:")
            else:
                  print("Employee Record does not exists")
      except cx_Oracle.DatabaseError as db:
            print("Problem in database:",db)
```

```
#empupdatedemo.py--main program
from empupdate import updateemprecord
updateemprecord()
```

```
#This program accept employee number from KBD and Remove employee record
#empdel.py--File Name and acts module name
import cx_Oracle
def   deleteemployee():
      try:
            con=cx_Oracle.connect("scott/tiger@localhost/orcl")
            cur=con.cursor()
            #design the query and execute
            empno=int(input("Enter Employee Number:"))
            cur.execute("delete from employee where eno=%d" %empno)
            con.commit()
            if(cur.rowcount>0):
                  print("Employee Record Removed:")
            else:
                  print("Employee Record does not exists")
      except cx_Oracle.DatabaseError as db:
            print("Problem in database:",db)
```

```
#empdeldemo.py--main program
```

```
from empdel import deleteemployee
deleteemployee()
```

            ==
                  DRL statements (Data Retrieval Language)
            ==

=>The purpose of DRL statements is that "To read or extract the records data from table".
=>The DRL statement in Database softwares is "select"
=>Syntax:-

```
SQL> select colname1, colname-2...Colname-n  from table_name;

SQL> select colname1, colname-2...Colname-n  from table_name where cond list;
```

Examples:

```
            SQL>select * from employee;

            SQL>select * from employee where sal>4.0 and sal<6.0;
```

==

=>In python Programming, Once select query executed , all the records are placed in the object cur.
=>To extract the records from cur object, we have 3 Functions. They are
1) fetchone()
2) fetchmany(no.of records)
3) fetchall()

=>fetchone() is used for fetching one record at a time where cur object pointing and it return the record in the form of tuple.
Syntax:-   record=curobj.fetchone()
=>fetchmany(n): here ''n" represents number of records
Syntax:-   records=curobj.fetchmany(n)
1) if n<0 then we never get any records
2) if n==total number of records in table then we get all records
3) if n<total number of records then we get n records
4) if n>total number of records then we get all the records.

=>fetchall() is used for fetching all the records of table and it returns the records in the form of list of tuples.
Syntax:-   records=curobj.fetchall()
--X------------------------------------

```
#This program reads / selects all the records from employee table
#empselectex1.py
import cx_Oracle
con=cx_Oracle.connect("scott/tiger@127.0.0.1/orcl")
cur=con.cursor()
#Desig the query and execute
sq="select * from employee"
cur.execute(sq)
print("="*50)
print("Employee Details:")
print("="*50)
while(True):
```

```
      rec=cur.fetchone()
      if(rec==None):
            print("="*50)
            break
      else:
            for val in rec:
                  print("{}".format(val) , end=" ")
            print()
```

```
#This program reads / selects all the records from employee table
#empselectex2.py
import cx_Oracle
con=cx_Oracle.connect("scott/tiger@127.0.0.1/orcl")
cur=con.cursor()
#Desig the query and execute
sq="select * from employee"
cur.execute(sq)
print("="*50)
print("Employee Details:")
print("="*50)
records=cur.fetchmany(4)
for record in records:
      for val in record:
            print("{}".format(val),end="  ")
      print()
else:
      print("="*50)
```

```
#This program reads / selects all the records from employee table
#empselectex3.py
import cx_Oracle
con=cx_Oracle.connect("scott/tiger@127.0.0.1/orcl")
cur=con.cursor()
#Desig the query and execute
sq="select * from employee"
cur.execute(sq)
print("="*50)
print("Employee Details:")
print("="*50)
records=cur.fetchall()
for record in records:
      for val in record:
            print("{}".format(val),end="  ")
      print()
else:
      print("="*50)
```

```
# This Program accepts the table name and display all records along with
Column Names
#RecordsSelect.py
import cx_Oracle
def   getrecords():
      try:
            tname=input("Enter Table Name:")
```

```
            con=cx_Oracle.connect("scott/tiger@localhost/orcl")
            cur=con.cursor()
            cur.execute("select * from %s " %tname)
            #Code For Obtaining Col Names(Meta Data)
            print("="*50)
            for colname in [ metadata for metadata in cur.description]:
                  print(colname[0], end= "  ")
            print()
            print("="*50)
            #Code for obtaining Records
            records =cur.fetchall()
            for record in records:
                  for val in record:
                        print("{}".format(val),end="  ")
                  print()
            print("="*50)

      except cx_Oracle.DatabaseError:
            print("Table does not exists")

#main program
getrecords()
```

===

Communication between Python Program and MYSQL Database

===

=>To perform various database operations by using Python language, First we must learn steps for communication between python program and MYSQL Data base software.

Steps:

----

1) import mysql.connector
2) Python Program must get the connection from MYSQL Database.
3) Create an object of cursor
4) Design the Query, place the query in cursor and execute.
5) Process the result which is available in cursor object.
6. Python Program Closes the connection.

Step-1: import mysql.connector

=>If python Program want to communicate with MYSQL data base , First we must install mysql-connector module by using pip and later we must import in python program.

Example: import mysql.connector

Step-2: Python Program must get the connection from MYSQL Database.

=>If a python program want a connection from MYSQL Database, we must use a pre-defined function connect() which is present in mysql.connector module and it returns an object of <class,mysql.connector.Connection>

Syntax:- varname=mysql.connector.connect(host="DNS/IPAddress", user="User Name of MYSQL",

passwd="password of MYSQL"

)

=>"varname" is an object of <class,mysql.connector.connection.MySQLConnection'>
=>mysql.connector is called Module name
=>connect() is predefined function in mysql.connector module
=>Here "user name " of MYSQL DB is "root"
=>here "passwd " of MYSQL DB is "root"
=>Here"DNS (Domain Naming Service)" represents Name of Machine Where Database Softwares resides". The default DNS of every machine is "localhost".
=>Here "IP Address(Internet Protocol address)" represents Numerical Address of a machine where Database software resides. The default IPaddress of every computer is "127.0.0.1"(also know as Loop Back Address).

Example:-

```
conobj=mysql.connector.connect(host="localhost",
      user="root",
      passwd="root")
print("python program got connection from MYSQL")
```

---

3) Create an object of cursor

=>The purpose of creating an object of cursor is that "To caray the query from Python Program and brings the result from data base software and hand over to python program".
=>To create an object of cursor, we use a pre-defined called cursor() , which is present in conobj.
=>Syntax:- varname=connobj.cursor()
=>here "varname" is an object of <class 'mysql.connector.cursor.MySQLCursor'>
=>Here "connobj" is an object <class, mysql.connector.connection.MySQLConnection'>

Examples:

```
kvrcur=conobj.cursor()
print("Cursor object created ..")
```

---

4) Design the Query, place the query in cursor and execute.

=>A Query is request / Question to the database from python Program.
=>To execute any type of Query, we use a pre-defined Function called execute(), which is present in <class 'mysql.connector.cursor.MySQLCursor'>

=>Syntax:- curobj.execute("Query")

---

5) Process the result which is available in cursor object.

=>This process makes us to understand, retrieve the data from cursor object and display it on the console.
Example: Handling exception messages
              dealing with results.
----
6) Close the connection:

=>To close the connection manually, we write finally block.
Example:

```
try:
      -----------------
      -----------------
except .......:
---------------
finally:
      print("\nFinally Block")
      if(conobj!=None):
            conobj.close()
            print("Database Connection Closed")
```

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

```
#This Program obtains connection from MySQL Data base
#testmysqlcon.py
import mysql.connector
try:
      con=mysql.connector.connect(host="localhost",

                  user="root",

                  passwd="root")
      print("Type of con=",type(con))
      print("Python Program Got Connection from MySQL DB")
except mysql.connector.DatabaseError as db:
      print("Problem in MySQL:",db)

      #This Program obtains connection from MySQL Data base
#testmysqlcursor.py
import mysql.connector
try:
      con=mysql.connector.connect(host="localhost",

                  user="root",

                  passwd="root")
      print("\nPython Program Got Connection from MySQL DB")
      cur=con.cursor()
      print("\ncursor object created:")
      print("Type cur=",type(cur))
except mysql.connector.DatabaseError as db:
      print("Problem in MySQL:",db)
```

```
#This program create a database in MYSQL on the name of "batch4pm"
#dbcreate.py
import mysql.connector
try:
	con=mysql.connector.connect(host="localhost",

				user="root",

				passwd="root")
	cur=con.cursor()
	dq="create database KVR"
	cur.execute(dq)
	print("\nData base created in mysql--verify")
except mysql.connector.DatabaseError as db:
	print("Problem in MySQL", db)


	#This program create a database in MYSQL on the name of "batch4pm"
#tablecreate.py
import mysql.connector
try:
	con=mysql.connector.connect(host="localhost",

				user="root",

				passwd="root",

				database="kvr")
	cur=con.cursor()
	#create a table employee in batch4pm data base
	tq="create table student (sno int  primary key,name varchar(10) not
null , marks float not null  )"
	cur.execute(tq)
	print("\nTable Created in MYSQL--verify")
except mysql.connector.DatabaseError as db:
	print("Problem in MySQL", db)

#This program accept employee values from KBD and insert into employee
table
#insertrecord.py-----File Name and acts as Module Name
import mysql.connector,sys
def  employeerecord():
	while(True):
		try:
			con=mysql.connector.connect(host="localhost",

				user="root",

				passwd="root",

				database="batch4pm")
			cur=con.cursor()
			#accept employee values
			empno=int(input("\nEnter Employee Number:"))
			ename=input("Enter Employee Name:")
			esal=float(input("Enter Employee Salary:"))
```

```
                  #design and execute query
                  iq="insert into employee values (%d,'%s',%f)"
                  cur.execute(iq %(empno,ename,esal) )
                  # (OR) cur.execute("insert into employee values
(%d,'%s',%f)" %(empno,ename,esal) )
                  con.commit()
                  print("{}  Record Inserted Successfully in employee
table".format(cur.rowcount))
                  print("-"*50)
                  ch=input("Do u want to insert another record(yes/no):")
                  if(ch.lower()=="no"):
                        print("\nThanks for using this program")
                        sys.exit()
            except mysql.connector.DatabaseError as db:
                  print("Problem in Database:",db)


#insertrecorddemo.py
from  insertrecord import employeerecord
employeerecord()

#This program accept employee Number from KBD and delete from employee
table
#deleterecord.py-----File Name and acts as Module Name
import mysql.connector,sys
def  employeerecord():
      while(True):
            try:
                  con=mysql.connector.connect(host="localhost",

                        user="root",

                        passwd="root",

                        database="batch4pm")
                  cur=con.cursor()
                  #accept employee values
                  empno=int(input("\nEnter Employee Number:"))
                  #design and execute query
                  dq="delete from employee where eno=%d"
                  cur.execute(dq %empno)
                  # (OR) cur.execute("delete from employee where eno=%d"
%empno)
                  con.commit()
                  if(cur.rowcount>0):
                        print("Employee Record Deleted:")
                  else:
                        print("Employee Record Does not Exists:")
                  print("-"*50)
                  ch=input("Do u want to delete another record(yes/no):")
                  if(ch.lower()=="no"):
                        print("\nThanks for using this program")
                        sys.exit()
            except mysql.connector.DatabaseError as db:
                  print("Problem in Database:",db)

#deleterecorddemo.py
from  deleterecord import employeerecord
```

```
employeerecord()


#This program accept employee Number from KBD and update emp salary
#updaterecord.py-----File Name and acts as Module Name
import mysql.connector,sys
def  employeerecord():
      while(True):
            try:
                  con=mysql.connector.connect(host="localhost",

                        user="root",

                        passwd="root",

                        database="batch4pm")
                  cur=con.cursor()
                  #accept employee values
                  empno=int(input("\nEnter Employee Number:"))
                  hike=float(input("Enter the Hike Percentage:"))
                  #design and execute query
                  uq="update employee set sal=sal+sal*%f where eno=%d"
                  cur.execute(uq %(hike,empno) )
                  # (OR) cur.execute("update employee set sal=sal+sal*%f
where eno=%d" %(hike,empno))
                  con.commit()
                  if(cur.rowcount>0):
                        print("Employee Record Updated:")
                  else:
                        print("Employee Record Does not Exists:")
                  print("-"*50)
                  ch=input("Do u want to update another record(yes/no):")
                  if(ch.lower()=="no"):
                        print("\nThanks for using this program")
                        sys.exit()
            except mysql.connector.DatabaseError as db:
                  print("Problem in Database:",db)

#updaterecorddemo.py
from updaterecord import employeerecord
employeerecord()


# This Program accepts the table name and display all records along with
Column Names
#RecordsSelect.py
import mysql.connector
def    getrecords():
      try:
            tname=input("Enter Table Name:")
            con=mysql.connector.connect(host="localhost",

                        user="root",

                        passwd="root",

                        database="batch4pm")
            cur=con.cursor()
```

```
            cur.execute("select * from %s " %tname)
            #Code For Obtaining Col Names(Meta Data)
            print("="*50)
            for colname in [ metadata for metadata in cur.description]:
                  print(colname[0], end= "  ")
            print()
            print("="*50)
            #Code for obtaining Records
            records =cur.fetchall()
            for record in records:
                  for val in record:
                        print("{}".format(val),end="  ")
                  print()
            print("="*50)

      except mysql.connector.DatabaseError:
            print("Table does not exists")

#main program
getrecords()


#This program create a database in MYSQL on the name of "batch4pm"
#fundtranstablecreate.py
import mysql.connector
try:
      con=mysql.connector.connect(host="localhost",

                  user="root",

                  passwd="root",

                  database="batch4pm")
      cur=con.cursor()
      #create a table deposit in batch4pm data base
      tq="create table deposit (acno int  primary key,cname varchar(10)
not null , bal  float not null  )"
      cur.execute(tq)
      print("\nTable Created in MYSQL--verify")
except mysql.connector.DatabaseError as db:
      print("Problem in MySQL", db)


#This program accept depositors values from KBD and insert into deposit
table
#Fundstrnasinsert.py-----File Name
import mysql.connector,sys
def  customersrecord():
      while(True):
            try:
                  con=mysql.connector.connect(host="localhost",

                        user="root",

                        passwd="root",

                        database="batch4pm")
                  cur=con.cursor()
```

```
                #accept employee values
                ano=int(input("\nEnter Account Number:"))
                cname=input("Enter Customer Name:")
                bal=float(input("Enter Customer Balance:"))
                #design and execute query
                iq="insert into deposit values (%d,'%s',%f)"
                cur.execute(iq %(ano,cname,bal) )
                con.commit()
                print("{}  Record Inserted Successfully in deposit
table".format(cur.rowcount))
                print("-"*50)
                ch=input("Do u want to insert another record(yes/no):")
                if(ch.lower()=="no"):
                     print("\nThanks for using this program")
                     sys.exit()
          except mysql.connector.DatabaseError as db:
                print("Problem in Database:",db)

#main program
customersrecord()


#This Programs transfers the amount from Source Account to Destination
account.
#FundsTrans.py
import mysql.connector,sys
try:
     con=mysql.connector.connect(host="localhost",

                user="root",

                passwd="root",

                database="batch4pm")
     cur=con.cursor()
     #get all account details
     cur.execute("select * from deposit")
     print("-"*50)
     for colname in [ metadata for metadata in cur.description]:
          print(colname[0], end= "  ")
     print()
     recs=cur.fetchall()
     print("-"*50)
     for rec in recs:
          for val in rec:
                print("{}".format(val),end="    ")
          print()
     print("-"*50)
     found=False
     sracno=int(input("Enter Source Account Number from where u are
sending:"))
     for rec in recs:
          if(sracno==rec[0]):
                found=True
                srbal=rec[2]
                break
     if(found==False):
          print("{} is invalid Source Account Number:".format(sracno))
```

```
	else:
		sramt=float(input("Enter the Amount to transfer from Source 
Account:"))
		if( (sramt+500)>srbal):
			print("Source Account does not contain sufficient 
Funds:")
		else:
			found=False
			dstacno=int(input("Enter Destination Account Number:"))
			for rec in recs:
				if(dstacno==rec[0]):
					found=True
					break
			if(found==False):
				print("{} is invalid Destination Account 
Number:".format(dstacno))
			else:
				#update the account
				cur.execute("update deposit set bal=bal-%f where 
acno=%d" %(sramt,sracno))
				cur.execute("update deposit set bal=bal+%f where 
acno=%d" %(sramt,dstacno) )
				con.commit()
				print("\n From {} Account Number , INR {} 
Transfered into {} Account Number--Verify".format(sracno,sramt,dstacno))
				print("="*50)
				cur.execute("select * from deposit")
				recs=cur.fetchall()
				for colname in [ metadata for metadata in 
cur.description]:
					print(colname[0], end= "  ")
				print()
				print("="*50)
				for rec in recs:
					for val in rec:
						print("{}".format(val), end="  ")
					print()
				print("="*50)


except mysql.connector.DatabaseError as db:
			print("Problem in Database:",db)
```

==

Importance of Object Oriented Principles

==

=>In Real Time to develop any project, we must use a language and it can satisfy two types of Principles. They are

1) Procedure Oriented Principles (Functional Oriented).
2) Object Oriented Principles.

=>Python is one of Both Procedure and Object Oriented Programming.
=>Even though Python belongs to Both Procedure and Object Oriented Programming , every thing treated as objects.

"Benifits of Treating Every thing as object "

=>In Object, we can store large Volume of Data and achieves Platform Independency
(Python)
=>The confidential Data can be transfered between multiple remote machine in the
form cipher text (Encrypted format). So that security can be enhanced (Improved).
=>The Large of Volume of Data Transfered between Multiple Machines all at once in
the form of object and leads to effective communication.
=>All Values are available around the objects and provides effective memory
Management.
============================X===================================

==
List of Object Oriented Principles
==

=>To Say Python is one of the Object Oriented Programming Language, It has to Satisfy the following OOPs Principles.

1) Classes
2) Objects
3) Data Encapsulation
4) Data Abstraction
5) Inheritance
6) Polymorphism
7) Message Passing

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

=====================================
1) Classes
=====================================

=>The purpose of Classes Concept is that "To develop Programmer (or) Custom
Defined Data types and to develop any real time application ".
=>The Purpose of Developing Programmer (or) Custom Defined Data types is that "To
store Customized data and to perform cutomized operations."
=>To develop programmer defined data type by using classes concept, we use a
keyword called "class"
=>Every Class Name is considered as Programmer defined data type.

=>Definition of Class:

=>A Class is a collection of Data Members and Methods

=>When we define a class , Memory will not create for Data members and Methods but whose memory memory will be created when we create an object.

==
Syntax for dfining a class in python
==

```
class  <Class Name>:
      Class level Data Members
      def   instancemethodname(self,list of formal params if any):
              ------------------------------------------------
            ----Specify Instance Data Members----
            --->Perfoms Specific Operations-----
      @classmethod
      def  classlevelmethodname(cls, list of formal params if any):
            ------------------------------------------------
            ----Specify Class Level Data Members----
            --->Perfoms Class Level Operations-----
      @staticmethod
      def staticmethodname(list of formal params if any):
            ----------------------------------------------------
            -------Utility Operations---------------------
            ----------------------------------------------------
```

==

Types of Data Members in Class

==

=>In a class of Python, we can define two types of Two Data Members. They are

a) Instance Data Members
b) Class Level Data Members

----

a) Instance Data Members

=>Instance Data members are used for Storing Specific Values
=>Instance Data Members must be specified in 3 ways. They are

a) Through an object name
b) Though Instance Method Name
c) Though Constructors.

=>Instance Data Members are always available inside of object(also known as object level data members).
=>Instance Data Members must be accessed w.r.t object name or self

objname.instance data member name
self.instance data member name

----

b) Class Level Data Members

=>Class Level Data Members are used for Storing Common Values.
=>Class Level Data Members must be specified in two ways. They are

a) Inside of Class definition
b) Inside of Class Level Method Definition

=>Class Level Data Members are always availableto all the object bcoz they are common
=>Class Level Data Members must be accessed w.r.t Class Name or object name or self or cls.

```
                        ClassName.Class Level data member name
                        ObjectName.Class Level data member name
                        self.Class Level data member name
                        cls.Class Level data member name
======================X=============================
#human.py
class Human:
      country="INDIA"  # Class Level Data Member

#main program
emp=Human()
stu=Human()
tea=Human()
#add the data to emp object
emp.eno=10
emp.name="RS"
emp.sal=5.6
#add the data to student object
stu.sno=111
stu.sname="Ram"
stu.marks=33.33
stu.cname="OUCET"
#add the data to teacher object
tea.tno=222
tea.tname="DR"
print("----------------------------------------")
print("Employee Data")
print("----------------------------------------")
print("Employee Number={}".format(emp.eno))
print("Employee Name={}".format(emp.name))
print("Employee Salary={}".format(emp.sal))
print("Employee Country={}".format(emp.country))
print("----------------------------------------")
print("Student Data")
print("----------------------------------------")
print("Student Number={}".format(stu.sno))
print("Student Name={}".format(stu.sname))
print("Student Marks={}".format(stu.marks))
print("Student College Name={}".format(stu.cname))
print("Student Country={}".format(stu.country))
print("----------------------------------------")
print("Teacher Data")
print("----------------------------------------")
print("Teacher Number={}".format(tea.tno))
print("Teacher Name={}".format(tea.tname))
print("Teacher Country={}".format(Human.country))
print("------------------------------------")


#This program stores  student number, name and Marks by using classes and
objects
#StudEx1.py
class Student :pass

#main program
s1=Student()  # Object creation
print("content of s1 before adding =", s1.__dict__)   # {  }
#add the data to s1
```

```
s1.sno=100
s1.name="RS"
s1.sal=3.4
print("content of s1 after adding =", s1.__dict__)   # {--------- }
print("-----------------------------------------------------------------")
#create another object
s2=Student()
print("content of s2 before adding =", s2.__dict__)   # {  }
#add the data to s2
s2.sno=101
s2.name="DR"
s2.sal=6.7
print("content of s2 after adding =", s2.__dict__)   # {--------- }


#This program stores  student number, name and Marks by using classes and
objects
#StudEx2.py
class Student :
      crs="PYTHON PROG"  # Class Level Data member specification

#main program
s1=Student()  # Object creation
s2=Student()  # Object creation
# add the data to s1
s1.sno=10
s1.sname="Ram"
s1.marks=11.11

# add the data to s2
s2.sno=20
s2.sname="Rak"
s2.marks=22.22

#display the object of s1
print("-----------------------------------")
print("S1 Object Content:")
print("-----------------------------------")
print("Student Number=",s1.sno)# Instance Data Members accessing
print("Student Name=",s1.sname)# Instance Data Members accessing
print("Student Marks=",s1.marks)  # Instance Data Members accessing
print("Student Course=",Student.crs)  # Class Level Data member accessing
print("-----------------------------------")
print("S2 Object Content:")
print("-----------------------------------")
print("Student Number=",s2.sno) # Instance Data Members accessing
print("Student Name=",s2.sname) # Instance Data Members accessing
print("Student Marks=",s2.marks) # Instance Data Members accessing
print("Student Course=",Student.crs)  # Class Level Data member
print("-----------------------------------")
```

==

Syntax for dfining a class in python

==

```
class  <Class Name>:
      Class level Data Members
      def   instancemethodname(self,list of formal params if any):
              ------------------------------------------------
            ----Specify Instance Data Members----
            --->Perfoms Specific Operations-----
      @classmethod
      def  classlevelmethodname(cls, list of formal params if any):
            ------------------------------------------------
            ----Specify Class Level Data Members----
            --->Perfoms Class Level Operations-----
      @staticmethod
      def staticmethodname(list of formal params if any):
            ----------------------------------------------------
            -------Utility Operations---------------------
            ----------------------------------------------------
```

Explanation:

=>"class" is a keyword , which is used to develop Programmer-defined Data Types.
=> <class name> is one of valid variable name and treated as Class Name and Every Class Name is one of the Programmer-Defined Data Type.
=>In Class of Python , we can define two types of data members. They are
a) Instance Data Members
b) Class Level Data Members
=>In Class of Python , we can define Three types of Methods. They are
a) Instance Methods
b) Class Level Methods
c) Static Methods

-

```
#Program reading and writing student details by using OOPs
#StudEx3.py
class Student:pass


#main program
s1=Student()
s2=Student()
print("-----------------------------------------")
print("Enter First Student Details:")
print("-----------------------------------------")
s1.sno=int(input("Enter Student Number:"))
s1.sname=input("Enter Student Name:")
s1.marks=float(input("Enter Student Marks:"))
print("-----------------------------------------")
print("Enter Second Student Details:")
print("-----------------------------------------")
s2.sno=int(input("Enter Student Number:"))
s2.sname=input("Enter Student Name:")
s2.marks=float(input("Enter Student Marks:"))
print("-----------------------------------------")
print("\nFirst Student Details:")
```

```
print("-------------------------------------------")
print("Student Number={}".format(s1.sno))
print("Student Name={}".format(s1.sname))
print("Student Marks={}".format(s1.marks))
print("-------------------------------------------")
print("\nSecond Student Details:")
print("-------------------------------------------")
print("Student Number={}".format(s2.sno))
print("Student Name={}".format(s2.sname))
print("Student Marks={}".format(s2.marks))
print("-------------------------------------------")


#Program reading and writing student details by using OOPs
#StudEx4.py
class Student:
      def readstuddata(self):
            print("="*50)
            self.sno=int(input("Enter Student Number:"))
            self.sname=input("Enter Student Name:")
            self.marks=float(input("Enter Student Marks:"))
            print("="*50)

      def dispstuddata(self):
            print("="*50)
            print("Student Number={}".format(self.sno))
            print("Student Name={}".format(self.sname))
            print("Student Marks={}".format(self.marks))
            print("="*50)
#main program
s1=Student()
s2=Student()
print("Enter First Student Information:")
s1.readstuddata()  # Method Call
print("\nEnter Second Student Information:")
s2.readstuddata()   # Method Call
print("\nFirst Student Information")
s1.dispstuddata() # Method Call
print("\nSecond Student Information")
s2.dispstuddata() # Method Call



#Program reading and writing student details by using OOPs
#StudEx5.py
class Student:
      def readstuddata(self):
            print("="*50)
            self.sno=int(input("Enter Student Number:"))
            self.sname=input("Enter Student Name:")
            self.marks=float(input("Enter Student Marks:"))
            print("="*50)
            self.dispstuddata() # calling Instance Method from another
Instance Method

      def dispstuddata(self):
            print("="*50)
            print("Student Number={}".format(self.sno))
```

```
            print("Student Name={}".format(self.sname))
            print("Student Marks={}".format(self.marks))
            print("="*50)
#main program
print("First Student Information")
s1=Student()
s2=Student()
s1.readstuddata()
print("Second Student Information")
s2.readstuddata()
```

```
               ===========================================
                  Types of Methods in Class of Python
              ===========================================
=>In Python Programming, we can define 3 Types of Methods in side of
Class. They are

                  1) Instance Method
                  2) Class Level Method
                  3) Static Method.
------------------------------------------------------------------------------------------------------
1) Instance Method:
------------------------------------------------------------------------------------------------------
=>Instance Methods are used for Performing Specific Operatons on the data
of object
    and Hence Instance Methods are called Object Level Methods.
=>Instance Methods always Takes "self" as First Positional Parameters for
obtaining
    id of Current Class object.
=>Syntax:-
            def    InstanceMethodName(self, list of formal params):
                    -----------------------------------------
                  -------Specific Operations-------
                  -----------------------------------------
=>Instance Methods of a Class must be accessed w.r.t object name or self
            objectname.InstanceMethodName()
                        (or)
            self.InstanceMethodName()
--------------------------------------------------------------------------------
What is  self:
------------------
=>self is one of the implicit object used as a First formal parameter in
the definition of Instance Method
=>The self contains Id or memory address or reference of Current Class
object.
=>self is applicable for objects only.
==================================================================
```

```
#Program reading and writing student details by using OOPs
#StudEx3.py
class Student:pass


#main program
```

```
s1=Student()
s2=Student()
print("--------------------------------------------")
print("Enter First Student Details:")
print("--------------------------------------------")
s1.sno=int(input("Enter Student Number:"))
s1.sname=input("Enter Student Name:")
s1.marks=float(input("Enter Student Marks:"))
print("--------------------------------------------")
print("Enter Second Student Details:")
print("--------------------------------------------")
s2.sno=int(input("Enter Student Number:"))
s2.sname=input("Enter Student Name:")
s2.marks=float(input("Enter Student Marks:"))
print("--------------------------------------------")
print("\nFirst Student Details:")
print("--------------------------------------------")
print("Student Number={}".format(s1.sno))
print("Student Name={}".format(s1.sname))
print("Student Marks={}".format(s1.marks))
print("--------------------------------------------")
print("\nSecond Student Details:")
print("--------------------------------------------")
print("Student Number={}".format(s2.sno))
print("Student Name={}".format(s2.sname))
print("Student Marks={}".format(s2.marks))
print("--------------------------------------------")
```

```
#Program reading and writing student details by using OOPs
#StudEx4.py
class Student:
    def readstuddata(self):
        print("="*50)
        self.sno=int(input("Enter Student Number:"))
        self.sname=input("Enter Student Name:")
        self.marks=float(input("Enter Student Marks:"))
        print("="*50)

    def dispstuddata(self):
        print("="*50)
        print("Student Number={}".format(self.sno))
        print("Student Name={}".format(self.sname))
        print("Student Marks={}".format(self.marks))
        print("="*50)
#main program
s1=Student()
s2=Student()
print("Enter First Student Information:")
s1.readstuddata()  # Method Call
print("\nEnter Second Student Information:")
s2.readstuddata()   # Method Call
print("\nFirst Student Information")
s1.dispstuddata() # Method Call
print("\nSecond Student Information")
s2.dispstuddata() # Method Call
```

```
#Program reading and writing student details by using OOPs
```

```
#StudEx5.py
class Student:
      def readstuddata(self):
            print("="*50)
            self.sno=int(input("Enter Student Number:"))
            self.sname=input("Enter Student Name:")
            self.marks=float(input("Enter Student Marks:"))
            print("="*50)
            self.dispstuddata() # calling Instance Method from another
Instance Method

      def dispstuddata(self):
            print("="*50)
            print("Student Number={}".format(self.sno))
            print("Student Name={}".format(self.sname))
            print("Student Marks={}".format(self.marks))
            print("="*50)
#main program
print("First Student Information")
s1=Student()
s2=Student()
s1.readstuddata()
print("Second Student Information")
s2.readstuddata()
```

```
            ============================================
                 Types of Methods in Class of Python
            ============================================
=>In Python Programming, we can define 3 Types of Methods in side of
Class. They are

                 1) Instance Method
                 2) Class Level Method
                 3) Static Method.
----------------------------------------------------------------------------------------------------
1) Instance Method:
----------------------------------------------------------------------------------------------------
=>Instance Methods are used for Performing Specific Operatons on the data
of object
    and Hence Instance Methods are called Object Level Methods.
=>Instance Methods always Takes "self" as First Positional Parameters for
obtaining
    id of Current Class object.
=>Syntax:-
            def    InstanceMethodName(self, list of formal params):
                    -----------------------------------------
                 -------Specific Operations-------
                 -----------------------------------------
=>Instance Methods of a Class must be accessed w.r.t object name or self
            objectname.InstanceMethodName()
                        (or)
            self.InstanceMethodName()
--------------------------------------------------------------------------------
What is  self:
---------------------
```

=>self is one of the implicit object used as a First formal parameter in the definition of Instance Method
=>The self contains Id or memory address or reference of Current Class object.
=>self is applicable for objects only.

```
=====================================================================
2) Class Level Method:
---------------------------------------------
```

=>Class level Methods are used for Performing Class Level Operatons Such as Specifying Class Level Data Members and Performs operations on them (if required).
=>Class Level Methods always Takes "cls" as First Positional Parameters for obtaining Current Class Name.
=>Every Class Level Method must be preceded with a pre-defined decorator called @classmethod
=>Syntax:-

```
            @classmethod
            def    ClassLevelMethodName(cls, list of formal params):
                     ------------------------------------------
                  -------Specific Operations-------
                  ------------------------------------------
```

=>Every Class Level Method can be accessed w.r.t to Class Name or cls or object name or self

```
                  ClassName.Class Level method Name()
                              (OR)
                  cls.Class Level method Name()
                              (OR)
                  objectname.Class Level method Name()
                              (OR)
                  self.Class Level method Name()
-----------------------------------------
What is  cls  :
---------------------
```

=>cls is one of the implicit object used as a First formal parameter in the definition of Class Level Method
=>The cls contains Name of Current Class
=>cls is applicable for Class Level Data Members and Class Level Methods only.

```
=====================================================================
3) Static Method:
-----------------------------------
```

=>Static Methods are used for Performing Utility or Universal Operatons Such as caluclator, displaying the data of any obnject etc.
=>tatic Methods neither Takes "cls" nor takes "self "as First Positional Parameters but it may take another object(s) .
=>Every Static Method must be preceded with a pre-defined decorator called @staticmethod
=>Syntax:-

```
            @staticmethod
            def    StaticMethodName(list of formal params if any):
                     ------------------------------------------
                  -------Utility or Universal Operations-------
                  ------------------------------------------
```

=>Every Static can be accessed w.r.t to Corresponding Class Name or object name

```
                  ClassName.static method Name()
                              (OR)
```

```
                  objectname.static method  Name()
===========================X=======================================
```

```
#This program demosntrates the concept of Class Level Method
#classmethodex1.py
class Employee:
      @classmethod
      def   getcompaddr(cls):
            cls.cname="IBM"
            cls.addr="HYD"

      def   getempdata(self):
            self.eno=int(input("Enter Employee Number:"))
            self.ename=input("Enter Employee Name:")

      def  dispempdata(self):
            print("="*50)
            print("Employee Number:{}".format(self.eno))
            print("Employee Name:{}".format(self.ename))
            print("Employee Comp Name:{}".format(Employee.cname))
            print("Employee Comp Address:{}".format(Employee.addr))
            print("="*50)

#main program
Employee.getcompaddr()
e1=Employee()
e2=Employee()
print("Enter First Employee Data")
e1.getempdata()
print("Enter Second Employee Data")
e2.getempdata()
print("\nFirst Employee Data")
e1.dispempdata()
print("\nSecond Employee Data")
e2.dispempdata()

#This program demosntrates the concept of Class Level Method
#classmethodex2.py
class Employee:
      @classmethod
      def   getcompname(cls):
            Employee.cname="IBM"
            Employee.getcompplace() #One Class Level Method is calling
another class level

#method.

      @classmethod
      def getcompplace(cls):
            cls.addr="HYD"

      def   getempdata(self):
            self.eno=int(input("Enter Employee Number:"))
            self.ename=input("Enter Employee Name:")

      def  dispempdata(self):
            print("="*50)
```

```
            print("Employee Number:{}".format(self.eno))
            print("Employee Name:{}".format(self.ename))
            print("Employee Comp Name:{}".format(self.cname))
            print("Employee Comp Address:{}".format(self.addr))
            print("="*50)

#main program
Employee.getcompname()
e1=Employee()
e2=Employee()
print("Enter First Employee Data")
e1.getempdata()
print("Enter Second Employee Data")
e2.getempdata()
print("\nFirst Employee Data")
e1.dispempdata()
print("\nSecond Employee Data")
e2.dispempdata()


#This program demosntrates the concept of Class Level Method
#classmethodex2.py
class Employee:
      @classmethod
      def   getcompname(cls):
            Employee.cname="IBM"
            Employee.getcompplace() #One Class Level Method is calling
another class level

#method.

      @classmethod
      def getcompplace(cls):
            cls.addr="HYD"

      def   getempdata(self):
            self.eno=int(input("Enter Employee Number:"))
            self.ename=input("Enter Employee Name:")

      def  dispempdata(self):
            print("="*50)
            print("Employee Number:{}".format(self.eno))
            print("Employee Name:{}".format(self.ename))
            print("Employee Comp Name:{}".format(self.cname))
            print("Employee Comp Address:{}".format(self.addr))
            print("="*50)

#main program
Employee.getcompname()
e1=Employee()
e2=Employee()
print("Enter First Employee Data")
e1.getempdata()
print("Enter Second Employee Data")
e2.getempdata()
print("\nFirst Employee Data")
e1.dispempdata()
print("\nSecond Employee Data")
```

```
e2.dispempdata()

#This program demosntrates the concept of Class Level Method
#classmethodex4.py
class Employee:
      @classmethod
      def   getcompname(cls):
            Employee.cname="IBM"
      @classmethod
      def getcompplace(cls):
            cls.addr="HYD"

      def   getempdata(self):
            self.eno=int(input("Enter Employee Number:"))
            self.ename=input("Enter Employee Name:")

      def  dispempdata(self):
            print("="*50)
            print("Employee Number:{}".format(self.eno))
            print("Employee Name:{}".format(self.ename))
            print("Employee Comp Name:{}".format(self.cname))
            self.getcompplace() # calling Class Level Method from
Instance Method
            print("Employee Comp Address:{}".format(self.addr))
            print("="*50)

#main program
Employee.getcompname()
e1=Employee()
e2=Employee()
print("Enter First Employee Data")
e1.getempdata()
print("Enter Second Employee Data")
e2.getempdata()
print("\nFirst Employee Data")
e1.dispempdata()
print("\nSecond Employee Data")
e2.dispempdata()

#This program demosntrates the concept of Class Level Method
#classmethodex5.py---file name acts as module name
class Employee:
      @classmethod
      def   getcompname(cls):
            Employee.cname="IBM"
      @classmethod
      def getcompplace(cls):
            cls.addr="HYD"

      def   getempdata(self):
            self.eno=int(input("Enter Employee Number:"))
            self.ename=input("Enter Employee Name:")

      def  dispempdata(self):
            print("="*50)
            print("Employee Number:{}".format(self.eno))
            print("Employee Name:{}".format(self.ename))
            print("Employee Comp Name:{}".format(self.cname))
```

```
            self.getcompplace() # calling Class Level Method from
Instance Method
            print("Employee Comp Address:{}".format(self.addr))
            print("="*50)


#This program demosntrates the concept of Static Method
#StaticEx1.py
class Human:
      @staticmethod
      def dispdata(obj):
            print("-"*40)
            for k,v in obj.__dict__.items():
                  print("\t{}\t{}".format(k,v))
            print("-"*40)

#main program
emp=Human()
stu=Human()
tea=Human()
#add the data to emp object
emp.eno=10
emp.name="RS"
emp.sal=5.6
#add the data to student object
stu.sno=111
stu.sname="Ram"
stu.marks=33.33
stu.cname="OUCET"
#add the data to teacher object
tea.tno=222
tea.tname="DR"
print("-------------------------------------")
print("Employee Data")
Human.dispdata(emp)
print("Student Data")
Human.dispdata(stu)
print("Teacher Data")
Human.dispdata(tea)
print("--------------------------


##This program demosntrates the concept of Static Method
#StaticEx2.py
class Human:
      @staticmethod
      def dispdata(obj):
            print("-"*40)
            for k,v in obj.__dict__.items():
                  print("\t{}\t{}".format(k,v))
            print("-"*40)

class Employee:pass

class Student:pass
class Book:pass

#main program
```

```
e1=Employee()
e1.eno=10
e1.ename="Rossum"
e1.sal=4.5
s1=Student()
s1.stno=100
s1.sname="Ritche"
s1.marks=55.55
s1.cname="OUCET"
b1=Book()
b1.isbn=4567
b1.bname="Python Programming"
b1.price=456.78
b1.publication="TaTa MeGrahill"
b1.sale="ready"
#display the all objects data
h=Human()
print("Employee Data:")
h.dispdata(e1)
print("Student Data:")
h.dispdata(s1)
print("Book Data:")
Human.dispdata(b1)
```

```
#This program demosntrates the concept of calculator Method
#StaticEx3.py
import sys
class Calc:
      @staticmethod
      def  calculate(obj):
            match (obj.op):
                  case "+":

      print("sum({},{})={}".format(obj.a,obj.b,obj.a+obj.b))
                  case "-":
                        print("sub({},{})={}".format(obj.a,obj.b,obj.a-
obj.b))
                  case "*":

      print("mul({},{})={}".format(obj.a,obj.b,obj.a*obj.b))
                  case "/":
                        try:

      print("div({},{})={}".format(obj.a,obj.b,obj.a/obj.b))
                        except ZeroDivisionError:
                              print("\nDon't enter Zero for Den...")
                  case "//":
                        print("floor
Div({},{})={}".format(obj.a,obj.b,obj.a//obj.b))
                  case "%":

      print("mod({},{})={}".format(obj.a,obj.b,obj.a%obj.b))
                  case "**":

      print("expop({},{})={}".format(obj.a,obj.b,obj.a**obj.b))
                  case _:
```

```
                        print("{} is not Arithmetic 
Operator:".format(obj.op))

class Numbers:
        def   getvalues(self):
                try:
                        self.a=float(input("Enter First Value:"))
                        self.b=float(input("Enter Second Value:"))
                        self.op=input("Enter any Arithmetic Operator:")
                except ValueError:
                        print("Don't Enter strs/ special symbols/ alpha-
numerics")
                        sys.exit()


#main program
n=Numbers()
n.getvalues()
Calc.calculate(n)

#table.py
class Table:
        def   getnumber(self):
                self.n=int(input("Enter a Number:"))
        def  gettable(self):
                if(self.n<=0):
                        print("{} is invalid input:".format(self.n))
                else:
                        print("-"*50)
                        print("Mul Table for {}".format(self.n))
                        print("-"*50)
                        for i in range(1,11):
                                print("\t{} x {}={}".format(self.n,i,self.n*i))
                        print("-"*50)

#main program
t=Table()
t.getnumber()
t.gettable()

#moduletable.py---file name and acts as module name
class Table:
        def   getnumber(self):
                self.n=int(input("Enter a Number:"))
        def  gettable(self):
                self.getnumber()
                if(self.n<=0):
                        print("{} is invalid input:".format(self.n))
                else:
                        print("-"*50)
                        print("Mul Table for {}".format(self.n))
                        print("-"*50)
                        for i in range(1,11):
                                print("\t{} x {}={}".format(self.n,i,self.n*i))
                        print("-"*50)


#TableDemo.py---mainm program
```

```
from moduletable import Table
t=Table()
t.gettable()

#This program reads all the records of employee table by using classes
and objects
#employeerecs.py
import mysql.connector
class EmployeeRecords:
      def  getrecords(self):
            con=mysql.connector.connect(host="localhost",

                        user="root",

                        passwd="root",

                        database="batch4pm")
            cur=con.cursor()
            cur.execute("select * from employee")
            print("="*50)
            for colname in [colnames[0] for colnames in cur.description]:
                  print("{}".format(colname),end="  ")
            print()
            print("="*50)
            #get the records
            records=cur.fetchall()
            for record in records:
                  for rec in record:
                        print("{}".format(rec), end="  ")
                  print()
            print("="*50)

#main porogram
eo=EmployeeRecords()
eo.getrecords()


#This program reads employee values from KBD  and insert them in employee
table by using classes and objects.
#EmpInsert.py
import mysql.connector
class EmpInsert:
      def  getempvalues(self):
            self.empno=int(input("Enter Employee Number:"))
            self.ename=input("Enter Employee Name:")
            self.sal=float(input("Enter Employee Salary:"))

      def insertempdata(self):
            try:
                  con=mysql.connector.connect(host="localhost",

                              user="root",

                              passwd="root",

                              database="batch4pm")
                  cur=con.cursor()
```

```
                cur.execute("insert into employee values(%d, '%s', %f )"
%(self.empno,self.ename,self.sal) )
                con.commit()
                if(cur.rowcount>0):
                     print("\n{}  record inserted".format(cur.rowcount)
)
                else:
                     print("Unable to Insert into the Table:")
           except mysql.connector.DatabaseError as db:
                print("Problem in Data Base:", db)

#main program
eo=EmpInsert()
eo.getempvalues()
eo.insertempdata()

#student.py---file name and acts as module name
class Student:
     def   setstudvalues(self,sno,sname,smarks,cname="OUCET"):
           self.stno=sno
           self.sname=sname
           self.smarks=smarks
           self.cname=cname

     def dispstuddata(self):

     print("\t{}\t{}\t{}\t{}".format(self.stno,self.sname,self.smarks,se
lf.cname))


#this program reads student data and insert student in the stud.data file
by using pickling with Classes and Objects
#StudentPickDemo.py
import pickle,sys
from student import Student
class StudentPick:
     def  insertstuddata(self):
           with open("stud.data","ab") as fp:
                while(True):
                     try:
                          print("-"*50)
                          self.sno=int(input("Enter Student Number:"))
                          self.name=input("Enter Student Name:")
                          self.marks=float(input("Enter Student
Marks:"))
                          #create Student  class object
                          so=Student()

     so.setstudvalues(self.sno,self.name,self.marks)
                          #dump or save student data in file
                          pickle.dump(so,fp)
                          print("-"*50)
                          print("\nStudent Record Inserted into the
file Sucessfully:")
                          print("-"*50)
                          ch=input("Do u want to insert another
record(yes/no)")
                          if(ch.lower()=="no"):
```

```
                                print("Thanks for using this progrm")
                                sys.exit()
                        except ValueError:
                            print("Don't enter strs / symbols / alpha-
numeric for Student Number and Marks")
#main program
sp=StudentPick()
sp.insertstuddata()


#This program reads the students records from file by using un-pickling
with Classes and Objects.
#StudentUnPickDemo.py
import pickle
class StudentUnPick:
    def    getstudentrecords(self):
        try:
            with open("stud.data","rb") as fp:
                print("-"*50)
                print("\tStno\tName\tMarks\tCname")
                print("-"*50)
                while(True):
                    try:
                        obj=pickle.load(fp) # type of obj=
<class 'student.Student'>
                        obj.dispstuddata()
                    except EOFError:
                        print("-"*50)
                        break

        except FileNotFoundError:
            print("File does not exists:")


#main program
sup=StudentUnPick()
sup.getstudentrecords()
```

==============================
Constructors in Python
==============================

Index

=>Purpose of Constructors
=>Definition of Constructors
=>Syntax for Constructors
=>Rules for using Constructors
=>Types of Constructors
    a) Default Constructor
    b) Parameterized Constructors
=>Programming Examples

==============================
Constructors in Python

==============================
=>The purpose of Constructors is that " To Initlize the object ".
=>Initlizing the object is nothing but placing our own values without leaving an
object empty.

=>Definition of Constructor:

=>A constructor is a special method which is automatically or implicitly called by PVM during Object Creation and whose Role is to Initlize the object (placing our own values without leaving an object empty).

Syntax for Constructor:

```
      def  __init__(self, list of formal params if any):
            -----------------------------------
            -----------------------------------
            Block of Statements--Initlization
            -----------------------------------
            -----------------------------------
```

===
Rules for using Constructors:
===
=>The Name of the Constructor is always __init__(self,....)
=>Constructors are automatically or implicitly called by PVM during Object
creation.
=>Constructors will not return any value
=>Constructors will participate Inheritance
=>Constructors can be Overridden

===
Types of Constructors:
===
=>In Python Programming, we have two types of Constructors. They are
a) Default or Parameterless Constructor
b) Parameterized Constructor

---

a) Default or Parameterless Constructor

=>The purpose of Default or Parameterless Constructor is that " To initlize the multiple objects of same class with same values".
=>A constructor is said to be default if and only if It never takes any parameters(except self).

```
=>Syntax:      def     __init__(self):
                            ------------------------------
                       ------------------------------
                       Block of statements--Initlization
                       ------------------------------
                       ------------------------------
```

Examples:

```
#defaultcontex1.py
class Test:
      def   __init__(self):
            print("I am default Constructor:")
```

```
        self.a=10
        self.b=20
        print("Val of a={}".format(self.a))
        print("Val of b={}".format(self.b))

#main program
t1=Test()
t2=Test()
t3=Test()
```
==

b) Parameterized Constructor

=>The purpose of Default or Parameterized Constructor is that " To initlize the multiple objects of same class with different values".
=>A constructor is said to be Parameterized if and only if It always takes parameter(s) along self.

```
=>Syntax:      def     __init__(self,list of formal pareams):
                            ------------------------------
                     ------------------------------
                     Block of statements--Initlization
                     ------------------------------
                     ------------------------------
```

```
#paramcontex1.py
class Test:
    def   __init__(self,a,b):
        print("id current object:",id(self))
        print("I am Parameterized Constructor:")
        self.a=a
        self.b=b
        print("Val of a={}".format(self.a))
        print("Val of b={}".format(self.b))

#main program
t1=Test(10,20)
t2=Test(100,200)
t3=Test(1000,2000)
print("----------------------------------------")
k=int(input("Enter First Value:"))
v=int(input("Enter Second Value:"))
t4=Test(k,v)
```
==

Note: In Class of Python, we can't define both default and Parameterized constructors bcoz PVM can remember only latest constructor (due to its interpretation Process) . To full fill the need of both default and parameterized constructors , we define single constructor with default parameter mechanism.

```
#defparamconstex1.py
class Test:

    def   __init__(self,a=10,b=20):
        print("I am default / Parameterized Constructor:")
        self.a=a
        self.b=b
        print("Val of a={}".format(self.a))
```

```
            print("Val of b={}".format(self.b))

#main program
t1=Test()
t2=Test(100,200)
============================X===============================
```

```
#defaultcontex1.py
class Test:
      def   __init__(self):
            print("I am default Constructor:")
            self.a=10
            self.b=20
            print("Val of a={}".format(self.a))
            print("Val of b={}".format(self.b))

#main program
t1=Test()
t2=Test()
t3=Test()

#defparamconstex1.py
class Test:

      def   __init__(self,a=10,b=20):
            print("I am default / Parameterized Constructor:")
            self.a=a
            self.b=b
            print("Val of a={}".format(self.a))
            print("Val of b={}".format(self.b))

#main program
t1=Test()
t2=Test(100,200)

#Empex1.py
class Employee:
      def  __init__(self):   # Constructor
            self.eno=int(input("Enter Employee Number:"))
            self.ename=input("Enter Employee Name:")
            self.sal=float(input("Enter Employee Salary:"))
      def  dispempdetails(self):
            print("Employee Number:{}".format(self.eno))
            print("Employee Name:{}".format(self.ename))
            print("Employee Salary:{}".format(self.sal))


#main program
eo=Employee()
print(eo.__dict__)

#paramcontex1.py
class Test:
      def   __init__(self,a,b):
            print("id current object:",id(self))
            print("I am Parameterized Constructor:")
```

```
            self.a=a
            self.b=b
            print("Val of a={}".format(self.a))
            print("Val of b={}".format(self.b))

#main program
t1=Test(10,20)
t2=Test(100,200)
t3=Test(1000,2000)
print("------------------------------------------")
k=int(input("Enter First Value:"))
v=int(input("Enter Second Value:"))
t4=Test(k,v)

#Test.py
class Test:
      def __init__(self,a,b):
            print("i am from param. constructor:")



#main program
```

==

Destructors in Python

==

=>We know that Garbage Collector is one of the in-built program in python, which is running behind of every python program and whose is role is to collect un-used memory space and it improves the performnace of python based applications.
=>Every Garbage Collector Program is internally calling Destructor program
=>The destructor name in python is   def __del__(self).
=>The destructor always called by Garbage Collector when the program executed completed for de-allocating the memory space.where as constructor called By PVM implicitly when object is create d for initlizing the object.

=>When the program execution is completed, GC calls its own destructor to de-allocate the memory space of objects present in program.(automatically GC running)

=>We have two programming   conditions for calling GC forcefully and to make the garbage collector to call destructor Functions.

a) Make the object refereence as None

Syntax :   objname=None

b) delete the object by using del

Syntax:-   del objname

=>Syntax:

```
            def       __del__(self):
                       -----------------------
```

----

t1=Test(10,20)

```
#DestEx1.py
class Employee:
      def __init__(self):
            print("i am from constructor")
            self.eno=10
            self.ename="RS"
            print("{}\t{}".format(self.eno,self.ename))
      def __del__(self):
            print("GC calls destructor for de-allocating unused memory 
space")

#main program
eo1=Employee()

#DestEx2.py
class Employee:
      def __init__(self):
            print("i am from constructor")
            self.eno=10
            self.ename="RS"
            print("{}\t{}".format(self.eno,self.ename))
      def __del__(self):
            print("GC calls destructor for de-allocating unused memory 
space")

#main program
eo1=Employee()
eo2=Employee()

#DestEx2.py
import time
class Employee:
      def __init__(self):
            print("i am from constructor")
            self.eno=10
            self.ename="RS"
            print("{}\t{}".format(self.eno,self.ename))
      def __del__(self):
            print("GC calls destructor for de-allocating unused memory 
space")

#main program
eo1=Employee()
eo2=Employee()
eo3=Employee()
time.sleep(10)

#DestEx4.py
import time
class Employee:
      def __init__(self):
            print("i am from constructor")
```

```
            self.eno=10
            self.ename="RS"
            print("{}\t{}".format(self.eno,self.ename))
      def __del__(self):
            print("GC calls destructor for de-allocating unused memory
space")

#main program
eo1=Employee()
del  eo1   # GC calls   __del__(self)
eo2=Employee()
del eo2    # GC calls   __del__(self)
eo3=Employee()
```

```
#DestEx4.py
import time
class Employee:
      def __init__(self):
            print("i am from constructor")
            self.eno=10
            self.ename="RS"
            print("{}\t{}".format(self.eno,self.ename))
      def __del__(self):
            print("GC calls destructor for de-allocating unused memory
space")

#main program
eo1=Employee()
del  eo1   # GC calls   __del__(self)
eo2=Employee()
del eo2    # GC calls   __del__(self)
eo3=Employee()
```

```
DestEx5.py
import time
class Employee:
      def __init__(self):
            print("i am from constructor")
            self.eno=10
            self.ename="RS"
            print("{}\t{}".format(self.eno,self.ename))
      def __del__(self):
            print("GC calls destructor for de-allocating unused memory
space")

#main program
eo1=Employee()
eo2=eo1 # Deep Copy
eo3=eo2 # Deep Copy
print("No Loger Interested to maintain eo1  memory space")
time.sleep(10)
del eo1  # here GC will not call __del__(self) , bcoz eo2 points to that
object
print("No Loger Interested to maintain eo2  memory space")
time.sleep(10)
```

```
del eo2   # here GC will not  call __del__(self) , bcoz eo2 is no where
and no object points memory space
print("No Loger Interested to maintain eo3  memory space")
```

```
#DestEx6.py
import time
class Employee:
      def __init__(self):
            print("i am from constructor")
            self.eno=10
            self.ename="RS"
            print("{}\t{}".format(self.eno,self.ename))
      def __del__(self):
            print("GC calls destructor for de-allocating unused memory
space")

#main program
eo1=Employee()
eo2=Employee()
eo1=None   # GC calls   __del__(self)---forcefuly calling GC
eo2=None    # GC calls   __del__(self)---forcefuly calling GC
eo3=Employee()
#Automatically  calling GC
```

===

objects in Python

===

=>When we define a class, memory space is not created for Data Members and Methods but whose memory is created when we create an object w.r.t class name.
=>To do any Data Processing, It is mandatory to create an object.
=>To create an object, there must exists a class Definition otherwise we get NameError

Definition of object:

=>Instance of a class is called object (Instance is nothing but allocating sufficient memory space for the Data Members and Methods of a class)

Syntax for creating an object
----

varname=classname()

Examples: create an object of Student

so=Student()

Example:- create an object Employee

eo=Employee()

Differences Betwwen Classes and Objects

Class:

----
1) A class is a collection of Data Members and Methods
2) When we define a class, memory space is not created for Data Members and Methods and it can be treated as specification / model for real time application.
3) Definition of a perticular exists only once
4) When we develop any Program with OOPs principles, Class Loaded First only once in main memory.

Objects:
----
1) Instance of a class is called Object

2) When we create an object, we get the memory space for Data members and Methods.
3)w.r.t One class Definition, we can create multiple objects.
4)we can crate an object after loading the class definition otherwise we get NameError
================================X==================================

```
#Test.py
class Student1:
      def   disp1(self):
            print("disp1()--student1 class")
            s2=Student2 ()
            s2.disp()

class Student2:
      def  disp(self):
            print("disp2()--Student2 class")

#main program
s1=Student1()
s1.disp1()
```

==

Data Encapsulation and Data Abstraction

==

Data Encapsulation:

=>The Process of Hiding the confidential Information / Data / Methods from external Programmers / end users is called Data Encapsulation.
=>The Purpose of Encapsulation concept is that "To Hide Confidental Information / Features of Class (Data Members and Methods and constructors) ".
=>Data Encapsulation can be applied in three levels. They are
- a) At Data Members Level
- b) At Methods Level
- c) At Constructor Level

=>To implement Data Encapsulation in python programming, The Data Members , Methods and Constructors must be preceded with double under score (_ _)

Syntax1:-

```
class <ClassName>:
    def   methodname(self):
          self.__Data MemberName1=Value1
          self.__Data MemberName2=Value2
```

```
                                   ----------------------------------------
-----------
                                   self.__Data MemberName-n=Value-n


Syntax2:-                 class <ClassName>:
                             def    __methodname(self):
                                   self.Data MemberName1=Value1
                                   self.Data MemberName2=Value2
                                   ----------------------------------------
-----------
                                   self.Data MemberName-n=Value-n
Syntax3:-                 class <ClassName>:
                             def    ____init__(self):
                                   self.Data MemberName1=Value1
                                   self.Data MemberName2=Value2
                                   ----------------------------------------
-----------
                                   self.Data MemberName-n=Value-n
```

Example1:

```
#account.py----file name and treated as module name
class Account:
      def  getaccountdet(self):
            self.__acno=34567
            self.cname="Rossum"
            self.__bal=34.56
            self.bname="SBI"
            self.__pin=1234
            self.pincode=4444444
            #here  acno,bal and pin are encapsulated
```

Example2:

```
#account1.py----file name and treated as module name
class Account1:
      def  __getaccountdet(self):  #here  __getaccountdet() is made is
encapsulated
            self.acno=34567
            self.cname="Rossum"
            self.bal=34.56
            self.bname="SBI"
            self.pin=1234
            self.pincode=4444444
```

==
Data Abstraction:

=>The Process of retrieving / extracting Essential Details without considering Hidden Details is called Data Abstraction.

Example1:

```
#others.py---This Program access only cname,bname and pincode only
from account  import Account
ao=Account()
ao.getaccountdet()
```

```
#print("Account Number={}".format(ao.acno))  Not Possible to access
print("Account Holder Name={}".format(ao.cname))
#print("Account Bal={}".format(ao.bal))  Not Possible to access
print("Account Branch Name={}".format(ao.bname))
#print("Account PIN={}".format(ao.pin))   Not Possible to access
print("Account Branch Pin Code={}".format(ao.pincode))
```

Example2:

```
#others1.py--here we can't access method itself. so that we cant access
Instance Data Members.
from account1 import Account1
ao=Account1()
#ao.getaccountdet()---can't access
#print("Account Number={}".format(ao.acno))
#print("Account Holder Name={}".format(ao.cname))
#print("Account Bal={}".format(ao.bal))
#print("Account Branch Name={}".format(ao.bname))
#print("Account PIN={}".format(ao.pin))
#print("Account Branch Pin Code={}".format(ao.pincode))

#DataMemEncapEx1.py---File Name and acts as module name
class Account:
      def   setaccountdet(self):
            self.__acno=1234                  #Data member Level
Encapsulation
            self.cname="Rossum"
            self.__bal=45.67
            self.bname="SBI"
            self.__pin=3456


#DataAbstex1.py
from DataMemEncapEx1 import Account
ac=Account()
ac.setaccountdet()
#print("Account Number=", ac.acno)   not possible to access ,bcoz acno is
encapsulated
print("Account Name=", ac.cname)
#print("Account Bal=", ac.bal)    not possible to access ,bcoz bal is
encapsulated
print("Account Branch Name=", ac.bname)
#print("Account Pin=", ac.pin)      not possible to access ,bcoz pin is
encapsulated


#MethodEncapEx1.py---File Name and acts as module name
class Account:
      def   __setaccountdet(self):   #Method Level Encapsulation
            self.acno=1234
            self.cname="Rossum"
            self.bal=45.67
            self.bname="SBI"
            self.pin=3456
      def   custinfo(self):
            print("i am customer of bank")
```

```
#DataAbstEx2.py
from MethodEncapEx1 import Account
ac=Account()
#ac.setaccountdet()  not possible to access ,bcoz setaccountdet   is
encapsulated
ac.custinfo()

#ConstEncapEx1.py---File Name and acts as module name
class Account:
      def   ____init__(self):   #Constructor Level Encapsulation
            self.acno=1234
            self.cname="Rossum"
            self.bal=45.67
            self.bname="SBI"
            self.pin=3456
      def   custinfo(self):
            print("i am customer of bank")


#DataAbstEx3.py
from ConstEncapEx1 import Account
ac=Account()
"""print("Account Number=", ac.acno)      Not possible to access
print("Account Name=", ac.cname)
print("Account Bal=", ac.bal)
print("Account Branch Name=", ac.bname)
print("Account Pin=", ac.pin)     """



#StudentMarksDemo.py
import mysql.connector
class StudentMarksReport:
      def   __init__(self):
            self.sno=int(input("Enter Student Number:"))
            self.sname=input("Enter Student Name:")
            while(True):
                  self.sub1=float(input("Enter Subject-1 Marks:"))
                  if(self.sub1<=100) and (self.sub1>=0):
                        break
            while(True):
                  self.sub2=float(input("Enter Subject-2 Marks:"))
                  if(self.sub2<=100) and (self.sub2>=0):
                        break
            while(True):
                  self.sub3=float(input("Enter Subject-3 Marks:"))
                  if(self.sub3<=100) and (self.sub3>=0):
                        break

      def  decideresults(self):
            self.totmarks=self.sub1+self.sub2+self.sub3
            self.percent=(self.totmarks)/300 *100
            if(self.sub1<40) or (self.sub2<40) or (self.sub3<40):
                  self.grade="FAIL"
            else:
                  if(self.totmarks>=250) and (self.totmarks<=300):
                        self.grade="DISTINCTION"
                  elif(self.totmarks>=200) and (self.totmarks<=249):
```

```
                    self.grade="FIRST"
              elif(self.totmarks>=150) and (self.totmarks<=199):
                    self.grade="SECOND"
              elif(self.totmarks>=120) and (self.totmarks<=149):
                    self.grade="THIRD"

      def   storeindb(self):
            try:
                  con=mysql.connector.connect(host="localhost",

                        user="root",

                        passwd="root",

                        database="batch4pm" )
                  cur=con.cursor()
                  iq="insert into result values(%d,'%s',
%f,%f,%f,%f,%f,'%s') "
                  cur.execute(iq
%(self.sno,self.sname,self.sub1,self.sub2,self.sub3,self.totmarks,self.pe
rcent,self.grade) )
                  con.commit()
                  print("Student record inserted in table:")
            except mysql.connector.DatabaseError as db:
                  print("Prob in Database:",db)

#main program
so=StudentMarksReport()
so.decideresults()
so.storeindb()
```

===

Inheritance

===

=>Ihenritance is one of distinct features of OOPs
=>The purpose of Inheritance is that " To build Re-usable Applications in Python Programming".

=>Definition of Inheritance:

=>The Process obtaining Data members , Methods and Constructors (Features) of one class into
    another class is called Inheritance.
=>The class which is giving Data members , Methods and Constructors (Features) is called Super or
     Base or Parent Class.
=>The Class which is taking Data members , Methods and Constructors (Features) is called Sub or
     Derived or Child Class.
=>The Inheritance concept always follows Logical Memory Management. This Memory Management says that " Neither we write Source Code nor Takes Physical Memory Space ".

Advatnages of Inheritance:

=>When we develop any inheritance based application, we get the following advantages.

1. Application Development Time is Less
2. Application Memory Space is Less
3. Application Execution time is Fast / Less
4. Application Performance is enhanced (Improved)
5. Redundency (Duplication) of the code is minimized.

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

Types of Inheritances

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

=>Types of Inheritance is a pattern or model which makes us to understand, how the features are inherited from base class into derived classes.

=>In Python Programming, we have 5 types of Inheritances. They are

1) Single Inheritance
2) Multi Level Inheritance
3) Hierarchical Inheritance
4) Multiple Inheritance
5) Hybrid Inheritance

===

Inheriting the features of Base Class into Derived Class

===

=>To Inherit the features of Base class into Derived Class, we use the following Syntax:

```
class <class-name-1>:
        ---------------------------
      ---------------------------
class <class-name-2>:
        ---------------------------
      ---------------------------
class <class-name-n>:
        ---------------------------
      ---------------------------

class <class-name-n+1> (class-name-1,class-name-2,....,class-name-n>:
        ----------------------------------------------
      ----------------------------------------------
```

Explanation:

=><classname-1> <classname-2>.........<classname-n> represents Name of Base Classes

=><classname-n+1> represents derived class name.

=>When we develop any Inheritance Based Application, It is always recommended to create an object of Bottom Most derived Class bcoz It inherits all features of Base Class and Intermediate Base Classes.

=>For Every Class in Python, there exist an implicit pre-defined super class called "object" bcoz object class provides Garbage Collection facility.

```
#This program demonstrates  the concept of inheritance
#comp.py---File Name and acts as module name.
class Company:
      def setcompdet(self):
            self.cname=input("Enter Company Name:")
            self.cplace=input("Enter Company Place:")
      def dispcompdet(self):
            print("Company Name:{}".format(self.cname))
            print("Company Place:{}".format(self.cplace))

#emp.py----file name and acts as module name
from comp import  Company
class Employee(Company):
      def   getempdet(self):
            self.eno=int(input("Enter Employee Number:"))
            self.ename=input("Enter Emplyee Name:")
            self.sal=float(input("Enter Employee Salary:"))
            self.setcompdet()

      def  dispempdet(self):
            print("Employee Number:{}".format(self.eno))
            print("Emplyee Name: {}".format(self.ename))
            print("Emplyee Salary: {}".format(self.sal))
            self.dispcompdet()


#InhProg1.py
class Operation:
      def   addop(self,a,b):
            print("sum({},{})={}".format(a,b,a+b))



class Ravi(Operation):pass

class Mohan(Operation):pass

class Rajesh(Operation):pass

r=Ravi()
r.addop(10,20)
m=Mohan()
m.addop(100,200)
r1=Rajesh()
r1.addop(-3,-4)

#InhProg2.py
from op import Operation
class Ravi(Operation):pass

class Mohan(Operation):pass

class Rajesh(Operation):pass

r=Ravi()
r.addop(10,20)
m=Mohan()
m.addop(100,200)
```

```
r1=Rajesh()
r1.addop(-3,-4)

#This program demonstrates  the concept of inheritance
#InhProg3.py
class Company:
      def setcompdet(self):
            self.cname=input("Enter Company Name:")
            self.cplace=input("Enter Company Place:")
      def dispcompdet(self):
            print("Company Name:{}".format(self.cname))
            print("Company Place:{}".format(self.cplace))

class Employee(Company):
      def   getempdet(self):
            self.eno=int(input("Enter Employee Number:"))
            self.ename=input("Enter Emplyee Name:")
            self.sal=float(input("Enter Employee Salary:"))
      def  dispempdet(self):
            print("Employee Number:{}".format(self.eno))
            print("Emplyee Name: {}".format(self.ename))
            print("Emplyee Salary: {}".format(self.sal))

#main program
e=Employee()
e.getempdet()
e.setcompdet()
e.dispempdet()
e.dispcompdet()

#This program demonstrates  the concept of inheritance
#InhProg4.py
class Company:
      def setcompdet(self):
            self.cname=input("Enter Company Name:")
            self.cplace=input("Enter Company Place:")
      def dispcompdet(self):
            print("Company Name:{}".format(self.cname))
            print("Company Place:{}".format(self.cplace))

class Employee(Company):
      def   getempdet(self):
            self.eno=int(input("Enter Employee Number:"))
            self.ename=input("Enter Emplyee Name:")
            self.sal=float(input("Enter Employee Salary:"))
            self.setcompdet()

      def  dispempdet(self):
            print("Employee Number:{}".format(self.eno))
            print("Emplyee Name: {}".format(self.ename))
            print("Emplyee Salary: {}".format(self.sal))
            self.dispcompdet()

#main program
e=Employee()
e.getempdet()
print("----------------------------------")
e.dispempdet()
```

```
#InhProg5.py
from emp import Employee
eo=Employee()
eo.getempdet()
eo.dispempdet()

#InhProg6.py
class Univ:
      def  getunivdet(self):
            self.uname=input("Enter University Name:")
            self.uloc=input("Enter University Location:")
      def   dispunivdet(self):
            print("University  Details:")
            print("-"*50)
            print("University Name:{}".format(self.uname))
            print("University Location:{}".format(self.uloc))
            print("-"*50)

u=Univ()
u.getunivdet()
u.dispunivdet()

#InhProg7.py
from stud import Student
s=Student()
s.getstuddet()
s.getcollegedet()
s.getunivdet()

s.dispstuddet()
s.dispcollegedet()
s.dispunivdet()

#stud.py--file name and acts as module name
from College import College
class Student(College):
      def  getstuddet(self):
            self.sno=int(input("Enter Student Number:"))
            self.sname=input("Enter Student Name:")
            self.marks=float(input("Enter Student Marks:"))
      def dispstuddet(self):
            print("-"*50)
            print("Student Details:")
            print("-"*50)
            print(" Student Number:{}".format(self.sno))
            print(" Student Name:{}".format(self.sname))
            print(" Student Marks:{}".format(self.marks))
            print("-"*50)

#Univ.py---File Name and acts as Module name
class Univ:
      def  getunivdet(self):
            self.uname=input("Enter University Name:")
            self.uloc=input("Enter University Location:")
      def   dispunivdet(self):
            print("University  Details:")
            print("-"*50)
```

```
            print("University Name:{}".format(self.uname))
            print("University Location:{}".format(self.uloc))
            print("-"*50)
```

            ==
                  Method Overriding in Python
            ==

=>Method Overriding=Method Heading is same + Method Body is Different

(OR)

=>The process of re-defining the original method of base class into various derived classes for performing different operations is called Method Overriding.

=>To use Method Overriding in python program we must apply Inheritance Principle.

=>Method Overriding used for implementing Polymorphism Principle.

Examples:

```
#methodoverex1.py
class Circle:
      def  draw(self):  # original Method
            print("Drawing Circle")

class Rect(Circle):
      def draw(self):    # overridden Method
            print("Drawing Rect:")
            super().draw()

class Square(Rect):
      def  draw(self): # overridden Method
            print("Drawing Square:")
            super().draw()

#main program
so=Square()
so.draw()
```

```
#teacher.py
class Teacher:
      def   readsub(self):
            print("Teacher advises to read 2 hours")

class LazyStudent(Teacher):
      def readsub(self):
            print("LazyStudent never read at all")
class PerfectStudent(Teacher):
      def readsub(self):
            print(" Perfect Student 2hrs reading and practicing")

ls=LazyStudent()
ls.readsub()
ps=PerfectStudent()
ps.readsub()
```

```
#MethodOverrideex1.py
```

```python
class Dog:
	def  noise(self):  # Original Method
		print("Dog makes a noise as BOW BOW")
class Cat(Dog):
	def  noise(self):  # Overridden Method
		super().noise()
		print("Cat makes a noise as MEW MEW")

class Cow(Cat):
	def  noise(self):   # Overridden Method
		super().noise()
		print("Cow makes a noise as Amba Amba")


#main program
c=Cow()
c.noise()


#MethodOverrideex2.py
class Dog:
	def  noise(self):  # Original Method
		print("Dog makes a noise as BOW BOW")
class Cat(Dog):
	def  noise(self):  # Overridden Method
		print("Cat makes a noise as MEW MEW")

class Cow(Cat):
	def  noise(self):   # Overridden Method
		print("Cow makes a noise as Amba Amba")
		Cat.noise(self)
		Dog.noise(self)
#main program
c=Cow()
c.noise()

#MethodOverrideEx3.py
class Circle:
	def  draw(self): # Orginal Method
		print("Drawing Circle:")

class Rect(Circle):
	def  draw(self): # Overridden Method
		print("Drawing Rect:")


#main program
r=Rect()
r.draw()

#teacher.py
class Teacher:
	def   readsub(self):
		print("Teacher advises to read 2 hours")

class LazyStudent(Teacher):
	def readsub(self):
		print("LazyStudent never read at all")
```

```
            super().readsub()
class PerfectStudent(Teacher):
      def readsub(self):
            print("Perfect Student 2hrs reading and practicing")
            super().readsub()

ls=LazyStudent()
ls.readsub()
print("-----------------------------------------------------")
ps=PerfectStudent()
ps.readsub()
```

==
Polymorphism in Python
==

=>Polymorphism is one of the distinct features of OOPs
=>The purpose of Polymorphism   is that "Efficient Utilization Memory--Less Memory space is achieved".

=>Def. of Polymorphism:

=>The Process of Representing "One Form in multiple Forms " is called Polymorphism.
=>The Polymorphism Principle is implemented(Bring into action) by Using "Method Overriding" feature of all OO Programming Languages.
=>In The definition of polymorphism, "One Form" represents "Original Method" and multiple forms represents Overridden Methods.
=>A "Form" is nothing but existence of a Method. if the method is existing in base class then it is called "Original Method(one form)" and if the method existing derived class then it is called "Overridden Method(multiple Forms)".

---X--

===
Number of approaches to call original methods from Overridden methods
===

=>We have two approches to call original method / constructors of base class from overridden method / constructors of derived class. They are

1) By using   super()
2) By using Class Name

----

1) By using   super():

=>super() is one of the pre-defined function, which is used for calling super class original method / constructor from overridden method / constructors of derived class.

Syntax1:-   super().methodname(list of values if any)

Syntax2:-   super().__init__(list of values if any)

=>with super() we are able to call only immediate base class method but uanble to call Specified method of base Class . To do this we must use class name approach.

----

2) By using Class Name:

=>By using ClassName approach, we can call any base class method / constructor name from the context of derived class method / constructor names.

```
Syntax1:-        ClassName.methodname(self, list of values if any)
Syntax2:-        ClassName.__init__(self, list of values if any)
```

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

```
#This program calculates area of different Figures such as  Circle and
Square
#polyex1.py
class Circle:
      def   area(self):  # Original Method
            self.r=float(input("Enter Radious:"))
            self.ac=3.14*self.r**2
            print("Area of Circle={}".format(self.ac))

class Square(Circle):
      def  area(self):  # Overridden Method
            self.s=float(input("Enter Side:"))
            self.sa=self.s**2
            print("Area of Square={}".format(self.sa))
            Circle.area(self)


#main program
s=Square()
s.area()

#This program calculates area of different Figures such as  Circle and
Square
#polyex2.py
class Circle:
      def   area(self):  # Original Method
            self.r=float(input("Enter Radious:"))
            self.ac=3.14*self.r**2
            print("Area of Circle={}".format(self.ac))

class Square:
      def  area(self):  # Overridden Method
            self.s=float(input("Enter Side:"))
            self.sa=self.s**2
            print("Area of Square={}".format(self.sa))
class Rect(Square,Circle):
      def  area(self):   # Overridden Method
            self.l=float(input("Enter length:"))
            self.b=float(input("Enter breadth:"))
            self.ar=self.l*self.b
            print("Area of Rect={}".format(self.ar))
            print("------------------------------------------------")
            Circle.area(self)
            Square.area(self)
```

```
#main program
r=Rect()
r.area()

#polyex3.py
class Test:
      def  __init__(self):  # Orginal Constructor
            print("Test Class Constructor")

class Sample(Test):
      def  __init__(self):  # Overridden Constructor
            super().__init__()
            print("Sample Class Constructor:")



#main program
s=Sample()


#AbstClassEx.py
class Banking:   # Here Baking class is called Abstract Class
      def   openac(self):pass          # Null Body Methods
      def   deposit(self,amt):pass
      def   loan(self,name,lamt):pass

class Ravi(Banking):
      def openac(self):
            print("Ravi Opened Saving Account in SBI")
class  Person(Banking):
      def  loan(self,name,lamt):
            print("{} Taken {} as loan and went out of 
India".format(name,lamt))

#main program
r=Ravi()
r.openac()
print("-"*40)
p=Person()
p.loan("VMalya",2.3)
p.loan("NModi",4.5)


#WithInh.py
class C1:
      def   x(self):
            print("C1-x()")

class C2(C1):
      def   y(self):
            print("C2-y()")
class C3(C1):
      def   z(self):
            print("C3-z()")
class C4(C2,C3):
      def   k(self):
            print("C4-k()")
```

```
#main program
o4=C4()
o4.k()
o4.z()
o4.y()
o4.x()

#WithInhex1.py
class C1:
      def   x(self):
            print("C1-x()")
class C2(C1):
      def   y(self):
            print("C2-y()")
class C3(C1):
      def   z(self):
            print("C3-z()")
class C4(C2,C3):
      def   k(self):
            print("C4-k()")
            super().y()
            super().z()
            super().x()

#main program
o4=C4()
o4.k()

#withployex2.py
class India:
      def   countrytype(self):
            print("India is developing Country")
      def  lang(self):
            print("Indians can speak multiple languages:")

class USA:
      def   countrytype(self):
            print("USA is Developed Country")
      def  lang(self):
            print("USA Citizens can speak English languages:")


#main program
io1=India()
uo=USA()
for obj in (io1,uo):   # object level polymorphism
      obj.countrytype()
      obj.lang()

      #withployex2.py
class India:
      def   countrytype(self):
            print("India is developing Country")
      def  lang(self):
            print("Indians can speak multiple languages:")

class USA:
```

```
	def   countrytype(self):
		print("USA is Developed Country")
	def  lang(self):
		print("USA Citizens can speak English languages:")


#main program
io1=India()
uo=USA()
for obj in (io1,uo):   # object level polymorphism
	obj.countrytype()
	obj.lang()


	#withployex4.py
def    show(*values):
	print("-"*50)
	for val in values:
		print("\t{}".format(val), end="")
	print()
	print("-"*50)


#main program
show()
show(10)
show(10,20)
show(10,20,30)
show(10,20,30,40)
show("Python","Java","DotNet","Django","Data Scienece")


#WithPolyex1.py
class C1:
	def   x(self):
		print("C1-x()")
class C2(C1):
	def   x(self):
		print("C2-x()")
class C3(C1):
	def   x(self):
		print("C3-x()")
class C4(C3,C2):
	def   x(self):
		print("C4-x()")
		C2.x(self)
		C3.x(self)
		C1.x(self)
#main program
o4=C4()
o4.x()
```

==
Regular Expressions
==

=>Regular Expressions is one of the Programming Language Independent Concept.
=>Regular Expressions are used Data Validation Purpose and builds robust applications in project development.

=>Real Time Products / Applications uses Regular Expressions :

=>All the Language Compilers and Interpreters
=>All kind Electronic Circuits
=>All kind Universal Protocals (Http, https, smtp,nmpt.pop,pop2..etc)
=>All types of Operating Systems
=>Used in Search Patterns.

===

Definition of Regular Expression:

=>Regular Expression of the search pattern (combination alphabets,digits and special symbols), which is used to serach in the given data for searching / matching / finding and obtains desired Result.

=>To deal with regular expressions programming, we must use a pree-defined called "re".

===

Pre-defined Functions in re module

===

=>The 're' module contains the follwing essential Functions.

---

1) finditer():

Syntax:-   varname=re.finditer("search-pattern","Given data")
=>here varname is an object of type <class,'Callable_Itetaror'>

=>This function is used for searching the search pattern in given data iteratively and it returns table of entries which contains start index , end index and matched value based on the search pattern.

---

2) group():

=>This function is used obtaining matched value by the findIter()
Syntax:-   varname=matchtabobj.group()

---

3) start():

=>This function is used obtaining starting index of matched value
Syntax: varname=matchobj.start()

---

4) end():

=>This function is used obtaining end index+1 of matched value
Syntax: varname=matchobj.end()

----
5) search():

Syntax:-   varname=re.search("search-pattern","Given data")
=>here varname is an object of <class,'match'>

=>This function is used for searching the search pattern in given data for first occuence / match only.
=>if the search pattern found in given data then it returns an object of match which contains matched value and start and end index values and it indicates search is successful.
=>if the search pattern not found in given data then it returns None and it indicates search is un-successful

----
6) findall():

Syntax:-   varname=re.findall("search-pattern","Given data")
=>here varname is an object of <class,'list'>

=>This function is used for searching the search pattern in entire given data and find all occurences / matches and it returns all the matched values in the form an object <class,'list'>

```
#RegExprex1.py
import re
gd="Python is an oop lang. Python is also Functional Programming Lang."
sp="Python"
mtab=re.finditer(sp,gd) # here matb is of type < class,
"callabel_iterator">
for omt in mtab:
      print("Start Index: {}  End Index:{}
Value:{}".format(omt.start(),omt.end(),omt.group()))

#RegExprex2.py
#Program for finding number of occurences of word "Python" in given data.
import re
gd="Python is an oop lang. Python is also Functional Programming Lang."
sp="Python"
noc=0
mtab=re.finditer(sp,gd)
print("-"*50)
for omt in mtab:
      noc=noc+1
      print("Start Index: {}  End Index:{}
Value:{}".format(omt.start(),omt.end(),omt.group()))
```

```
else:
      print("-"*50)
      print("Number of Occurences of '{}'={}".format(sp,noc))

#RegExprex3.py
#Program for finding number of occurences of word "Python" in given data.
import re
gd="Python is an oop lang. Python is also Functional Programming Lang."
sp="one"
noc=0
matchlist=re.findall(sp,gd)
print("Number of occurences of '{}'={}".format(sp,len(matchlist)))


#RegExprex4.py
#Program for finding number of occurences of word "Python" in given data.
import re
gd="Python is an oop lang. Python is also Functional Programming Lang."
sp="Python"
noc=0
matchinfo=re.search(sp,gd)
if (matchinfo!=None):
      print("{} Found In given Data:".format(sp))
else:
      print("{} does not found in given data".format(sp))

#RegExprex5.py
import re
gd=input("Enter a line of text:")
sp=input("Enter which word u want search:")
matchinfo=re.search(sp,gd)
if(matchinfo!=None):
      print("{} found in Given Data and search is successful
:".format(sp))
else:
      print("{} does not exists and search is un-successful".format(sp))
```

==

Programmer-defined character Classes in Regular Expressions

==

=>The purpose of Programmer-defined character Classes in Regular Expressions to prepare Search
Pattern to search in givan data for obtaining desired result.

=>Syntax: [Search Pattern]

1. [abc]----->searches for either 'a' or 'b' or 'c' only

2. [^abc]---->searches for all except 'a' or 'b' or 'c'

3. [a-z]------>searches for all Lower Case Alphabets only

4. [^a-z]----->searches for all except Lower Case Alphabets

5. [A-Z]---->searches for all Upper Case Alphabets only

6. [^A-Z]---->searches for all except Upper Case Alphabets only

7. [A-Za-z]---->Searcher for all Upper Case and lower case Alphabets only

8. [^A-Za-z]---->Searcher for all except Upper Case and lower case Alphabets only

9. [0-9]------>searches for all digits only

10. [^0-9]----->searches for all except Digits

11. [A-Za-z0-9]---->searches Alpha-numerics (Alphabets and digits)

12. [^A-Za-z0-9]---->searches for all special symbols (except Alpha-numerics)

```
      #Program for searching either 'a' or 'b' or 'c' only
#RegExprEx6.py
import re
matchtab=re.finditer("[abc]","cAaU#2RQk8%b6^WoP")
for onematch in matchtab:
      print("start Index:{}  End Index:{}
Value:{}".format(onematch.start(),onematch.end(),onematch.group()))

"""
Output
E:\KVR-PYTHON-4PM\REG EXPR>py RegExprEx6.py
start Index:0  End Index:1  Value:c
start Index:2  End Index:3  Value:a
start Index:11  End Index:12  Value:b

"""

#Program for searching for all   except  'a' or 'b' or 'c' only
#RegExprEx7.py
import re
matchtab=re.finditer("[^abc]","cAaU#2RQk8%b6^WoP")
for onematch in matchtab:
      print("start Index:{}  End Index:{}
Value:{}".format(onematch.start(),onematch.end(),onematch.group()))


"""
E:\KVR-PYTHON-4PM\REG EXPR>py RegExprEx7.py
start Index:1  End Index:2  Value:A
start Index:3  End Index:4  Value:U
start Index:4  End Index:5  Value:#
start Index:5  End Index:6  Value:2
start Index:6  End Index:7  Value:R
start Index:7  End Index:8  Value:Q
start Index:8  End Index:9  Value:k
start Index:9  End Index:10  Value:8
start Index:10  End Index:11  Value:%
start Index:12  End Index:13  Value:6
start Index:13  End Index:14  Value:^
start Index:14  End Index:15  Value:W
start Index:15  End Index:16  Value:o
```

```
start Index:16  End Index:17  Value:P
"""

#Program for searching for all  lower case alphabets
#RegExprEx8.py
import re
matchtab=re.finditer("[a-z]","cAaU#2RzQk8%b6^WoP")
for onematch in matchtab:
      print("start Index:{}  End Index:{}
Value:{}".format(onematch.start(),onematch.end(),onematch.group()))


"""
E:\KVR-PYTHON-4PM\REG EXPR>py RegExprEx8.py
start Index:0  End Index:1  Value:c
start Index:2  End Index:3  Value:a
start Index:7  End Index:8  Value:z
start Index:9  End Index:10  Value:k
start Index:12  End Index:13  Value:b
start Index:16  End Index:17  Value:o
"""

#Program for searching for all  except lower case alphabets
#RegExprEx9.py
import re
matchtab=re.finditer("[^a-z]","cAaU#2RzQk8%b6^WoP")
for onematch in matchtab:
      print("start Index:{}  End Index:{}
Value:{}".format(onematch.start(),onematch.end(),onematch.group()))

"""
E:\KVR-PYTHON-4PM\REG EXPR>py RegExprEx9.py
start Index:1  End Index:2  Value:A
start Index:3  End Index:4  Value:U
start Index:4  End Index:5  Value:#
start Index:5  End Index:6  Value:2
start Index:6  End Index:7  Value:R
start Index:8  End Index:9  Value:Q
start Index:10  End Index:11  Value:8
start Index:11  End Index:12  Value:%
start Index:13  End Index:14  Value:6
start Index:14  End Index:15  Value:^
start Index:15  End Index:16  Value:W
start Index:17  End Index:18  Value:P
"""

#Program for searching for all  Upper Case Alphabets
#RegExprEx10.py
import re
matchtab=re.finditer("[A-Z]","cAaU#2RzQk8%b6^WoP")
for onematch in matchtab:
      print("start Index:{}  End Index:{}
Value:{}".format(onematch.start(),onematch.end(),onematch.group()))

"""
E:\KVR-PYTHON-4PM\REG EXPR>py RegExprEx10.py
start Index:1  End Index:2  Value:A
start Index:3  End Index:4  Value:U
```

```
start Index:6  End Index:7  Value:R
start Index:8  End Index:9  Value:Q
start Index:15  End Index:16  Value:W
start Index:17  End Index:18  Value:P
"""#Program for searching for all except Upper Case Alphabets
#RegExprEx11.py
import re
matchtab=re.finditer("[^A-Z]","cAaU#2RzQk8%b6^WoP")
for onematch in matchtab:
      print("start Index:{}  End Index:{}
Value:{}".format(onematch.start(),onematch.end(),onematch.group()))

"""
E:\KVR-PYTHON-4PM\REG EXPR>py RegExprEx11.py
start Index:0  End Index:1  Value:c
start Index:2  End Index:3  Value:a
start Index:4  End Index:5  Value:#
start Index:5  End Index:6  Value:2
start Index:7  End Index:8  Value:z
start Index:9  End Index:10  Value:k
start Index:10  End Index:11  Value:8
start Index:11  End Index:12  Value:%
start Index:12  End Index:13  Value:b
start Index:13  End Index:14  Value:6
start Index:14  End Index:15  Value:^
start Index:16  End Index:17  Value:o
"""

#Program for searching for all lower and  Upper Case Alphabets
#RegExprEx12.py
import re
matchtab=re.finditer("[A-Za-z]","cAaU#2RzQk8%b6^WoP")
for onematch in matchtab:
      print("start Index:{}  End Index:{}
Value:{}".format(onematch.start(),onematch.end(),onematch.group()))

"""
E:\KVR-PYTHON-4PM\REG EXPR>py RegExprEx12.py
start Index:0  End Index:1  Value:c
start Index:1  End Index:2  Value:A
start Index:2  End Index:3  Value:a
start Index:3  End Index:4  Value:U
start Index:6  End Index:7  Value:R
start Index:7  End Index:8  Value:z
start Index:8  End Index:9  Value:Q
start Index:9  End Index:10  Value:k
start Index:12  End Index:13  Value:b
start Index:15  End Index:16  Value:W
start Index:16  End Index:17  Value:o
start Index:17  End Index:18  Value:P
"""

#Program for searching for all  except lower and  Upper Case Alphabets
#RegExprEx13.py
import re
matchtab=re.finditer("[^A-Za-z]","cAaU#2RzQk8%b6^WoP")
for onematch in matchtab:
```

```
	print("start Index:{}  End Index:{}
Value:{}".format(onematch.start(),onematch.end(),onematch.group()))



"""
E:\KVR-PYTHON-4PM\REG EXPR>py RegExprEx13.py
start Index:4  End Index:5  Value:#
start Index:5  End Index:6  Value:2
start Index:10  End Index:11  Value:8
start Index:11  End Index:12  Value:%
start Index:13  End Index:14  Value:6
start Index:14  End Index:15  Value:^

"""

#Program for searching for all digits
#RegExprEx14.py
import re
matchtab=re.finditer("[0-9]","cAaU#2RzQk8%b6^WoP")
for onematch in matchtab:
	print("start Index:{}  End Index:{}
Value:{}".format(onematch.start(),onematch.end(),onematch.group()))

""""
E:\KVR-PYTHON-4PM\REG EXPR>py RegExprEx14.py
start Index:5  End Index:6  Value:2
start Index:10  End Index:11  Value:8
start Index:13  End Index:14  Value:6
"""

#Program for searching for all  except  digits
#RegExprEx15.py
import re
matchtab=re.finditer("[^0-9]","cAaU#2RzQk8%b6^WoP")
for onematch in matchtab:
	print("start Index:{}  End Index:{}
Value:{}".format(onematch.start(),onematch.end(),onematch.group()))


"""
E:\KVR-PYTHON-4PM\REG EXPR>py RegExprEx15.py
start Index:0  End Index:1  Value:c
start Index:1  End Index:2  Value:A
start Index:2  End Index:3  Value:a
start Index:3  End Index:4  Value:U
start Index:4  End Index:5  Value:#
start Index:6  End Index:7  Value:R
start Index:7  End Index:8  Value:z
start Index:8  End Index:9  Value:Q
start Index:9  End Index:10  Value:k
start Index:11  End Index:12  Value:%
start Index:12  End Index:13  Value:b
start Index:14  End Index:15  Value:^
start Index:15  End Index:16  Value:W
start Index:16  End Index:17  Value:o
start Index:17  End Index:18  Value:P
"""
```

```
#Program for searching for all alpha-numerics( except special symbols)
#RegExprEx16.py
import re
matchtab=re.finditer("[A-Za-z0-9]","cAaU#2RzQk8%b6^WoP")
for onematch in matchtab:
      print("start Index:{}  End Index:{}
Value:{}".format(onematch.start(),onematch.end(),onematch.group()))

"""
E:\KVR-PYTHON-4PM\REG EXPR>py RegExprEx16.py
start Index:0  End Index:1  Value:c
start Index:1  End Index:2  Value:A
start Index:2  End Index:3  Value:a
start Index:3  End Index:4  Value:U
start Index:5  End Index:6  Value:2
start Index:6  End Index:7  Value:R
start Index:7  End Index:8  Value:z
start Index:8  End Index:9  Value:Q
start Index:9  End Index:10  Value:k
start Index:10  End Index:11  Value:8
start Index:12  End Index:13  Value:b
start Index:13  End Index:14  Value:6
start Index:15  End Index:16  Value:W
start Index:16  End Index:17  Value:o
start Index:17  End Index:18  Value:P
"""

#Program for searching for all Special Symbols except alpha-numerics
#RegExprEx17.py
import re
matchtab=re.finditer("[^A-Za-z0-9]","cA^aU#2Rz Qk8%b6^WoP")
for onematch in matchtab:
      print("start Index:{}  End Index:{}
Value:{}".format(onematch.start(),onematch.end(),onematch.group()))

"""
E:\KVR-PYTHON-4PM\REG EXPR>py RegExprEx17.py
start Index:2  End Index:3  Value:^
start Index:5  End Index:6  Value:#
start Index:9  End Index:10  Value:
start Index:13  End Index:14  Value:%
start Index:16  End Index:17  Value:^
"""

#Program for searching for space character
#RegExprEx18.py
import re
matchtab=re.finditer("\s","c A^aU#2R zQk8%b6^WoP")
for onematch in matchtab:
      print("start Index:{}  End Index:{}
Value:{}".format(onematch.start(),onematch.end(),onematch.group()))


"""
E:\KVR-PYTHON-4PM\REG EXPR>py RegExprEx18.py
start Index:1  End Index:2  Value:
start Index:9  End Index:10  Value:
```

"""

==
Pre-defined character Classes in Regular Expressions
==

=>Pre-defined character Classes in Regular Expressions are available in Python softawre as pre-defined API
=>The purpose of Pre-defined character Classes in Regular Expressions to prepare Search
Patterns to search in givan data for obtaining desired result.

=>Syntax: "\pre-defined character Classes"

=>The following the essential Pre-defined character Classes in Regular Expressions.

1) \s--------->searches for only space character
2) \S-------->searches for all except space character
3) \w-------->Searches for word character or alpha-numerics only (except special symbols) or [A-Za-z0-9]
4) \W------->Searches for special symbols (except alpha-numerics) or [^A-Za-z0-9]
5) \d-------->Searches for digit only [0-9]
6)\D-------->searches for all except digits[^0-9]

```
#Program for searching for all except  space character
#RegExprEx19.py
import re
matchtab=re.finditer("\S","c A^aU#2R zQk8%b6^WoP")
for onematch in matchtab:
      print("start Index:{}  End Index:{}
Value:{}".format(onematch.start(),onematch.end(),onematch.group()))


#Program for searching for all word characters except  special symbols
#RegExprEx20.py
import re
matchtab=re.finditer("\w","c A^aU#2R zQk8%b6^WoP")
for onematch in matchtab:
      print("start Index:{}  End Index:{}
Value:{}".format(onematch.start(),onematch.end(),onematch.group()))



#Program for searching for all  special symbols  except word characters
#RegExprEx21.py
import re
matchtab=re.finditer("\W","c A^aU#2R zQk8%b6^WoP")
for onematch in matchtab:
      print("start Index:{}  End Index:{}
Value:{}".format(onematch.start(),onematch.end(),onematch.group()))

#Program for searching for all digits
#RegExprEx22.py
import re
```

```
matchtab=re.finditer("\d","c A^aU#2R zQk 78 %b6^WoP")
for onematch in matchtab:
      print("start Index:{}  End Index:{}
Value:{}".format(onematch.start(),onematch.end(),onematch.group()))

#Program for searching for all  except  digits
#RegExprEx23.py
import re
matchtab=re.finditer("\D","c A^aU#2R zQk8%b6^WoP")
for onematch in matchtab:
      print("start Index:{}  End Index:{}
Value:{}".format(onematch.start(),onematch.end(),onematch.group()))



#Program for searching for exactly one k
#RegExprEx24.py
import re
matchtab=re.finditer("k","akaakkaakkkaka")
for onematch in matchtab:
      print("start Index:{}  End Index:{}
Value:{}".format(onematch.start(),onematch.end(),onematch.group()))


#Program for searching for either  one k or more k's
#RegExprEx25.py
import re
matchtab=re.finditer("k+","akaakkaakkkaka")
for onematch in matchtab:
      print("start Index:{}  End Index:{}
Value:{}".format(onematch.start(),onematch.end(),onematch.group()))
```

===
Quantifiers in Regular Expressions
===

=>Quantifiers in Regular Expressions are used for searching number of occurences of the specified value (alphabets or digits or special symbols) used in search pattern to search in the given data and obtains desired result.

```
1)  "k"------->It search for only one 'k' at a time
2) "k+"------>It search for either one 'k' more 'k' s
3) "k*"------>It search for either zero 'k'  or one 'k'  and  more 'k' s
4) "k?"----->>It search for either zero 'k'  or one 'k'
5)  ". " ---->It searches for all
```

Note:

```
\ddd  or   d{3}-----searches for 3 digits
\dd.\dd----searhes for 2 integer values and 2 decimal values
\d{2,4}----searches for min 2 digit number and max 4 digit number.
[A-Za-z]+---searches one alphabet   or More alphabets.

#Program for searching for either zero k or  one k or more k's
#RegExprEx26.py
import re
```

```
matchtab=re.finditer("k*","akaakkaakkkaka")
for onematch in matchtab:
      print("start Index:{}  End Index:{}
Value:{}".format(onematch.start(),onematch.end(),onematch.group()))

""""
E:\KVR-PYTHON-4PM\REG EXPR>py RegExprEx26.py
start Index:0  End Index:0  Value:
start Index:1  End Index:2  Value:k
start Index:2  End Index:2  Value:
start Index:3  End Index:3  Value:
start Index:4  End Index:6  Value:kk
start Index:6  End Index:6  Value:
start Index:7  End Index:7  Value:
start Index:8  End Index:11  Value:kkk
start Index:11  End Index:11  Value:
start Index:12  End Index:13  Value:k
start Index:13  End Index:13  Value:
start Index:14  End Index:14  Value:
      """

      #Program for searching for either zero k or  one k
#RegExprEx27.py
import re
matchtab=re.finditer("k?","akaakkaakkkaka")
for onematch in matchtab:
      print("start Index:{}  End Index:{}
Value:{}".format(onematch.start(),onematch.end(),onematch.group()))


#Program for searching for all
#RegExprEx28.py
import re
matchtab=re.finditer(".","akaakk333aakkkaka")
for onematch in matchtab:
      print("start Index:{}  End Index:{}
Value:{}".format(onematch.start(),onematch.end(),onematch.group()))

      #Program for mobile number validation
#mobilenumbervalid.py
import re
while(True):
      mno=input("Enter Ur Mobile Number:")
      if(len(mno)==10):
            result=re.search("\d{10}" ,mno)
            if(result!=None):
                  print("{} , Ur Mobile Number is Valid".format(mno))
                  break
      else:

            print("\nUr mobile should contain 10 digits in length")



#Program for searching Names and Marks of the strudents in given data
#NamesMarkslistex1.py
import re
```

```
gd="Rohit got 56 marks , Raman got 77 marks, Rocky got 88 marks, Ganesh got 99 marks, Anju got 66 marks Senapathi got 68 marks and Rossum got 11 marks and Kkgupta got 58 marks and Sagar got 48 marks"
nameslist=re.findall("[A-Z][a-z]+", gd)
print("-"*50)
print("Names of Students:")
print("-"*50)
for name in nameslist:
      print("\t{}".format(name))
print("-"*50)
markslist=re.findall("\d{2}",gd)
print("Marks of Students:")
print("-"*50)
for marks in markslist:
      print("\t{}".format(marks))
print("-"*50)
print("Student Names\tStudent Marks")
print("-------------------------------------------------------")
for sn,sm in zip(nameslist,markslist):
      print("\t{}\t\t{}".format(sn,sm))
print("-------------------------------------------------------")


#Program for searching Names and Marks of the strudents information by reading text document from file (studentsinfo.data)
#NamesMarkslistex2.py
import re
try:
      with  open("studentsinfo.data","r") as fp:
            filedata=fp.read()
            nameslist=re.findall("[A-Z][a-z]+",filedata)
            markslist=re.findall("\d{2}",filedata)
            print("-"*50)
            print("Student Names\tStudent Marks")
            print("-------------------------------------------------------")
            for sn,sm in zip(nameslist,markslist):
                  print("\t{}\t\t{}".format(sn,sm))
            print("-------------------------------------------------------")

except FileNotFoundError:
      print("File does not exists:")

#This program read the information from file and obtains email-ids by Reg Exp
#mailsvalid.py
import re
try:
      with  open("mailsinfo.data","r") as fp:
            filedata=fp.read()
            mailslist=re.finditer("\S+@\S+", filedata)
            print("-"*60)
            print("Students mails")
            print("-"*60)
            for mail in mailslist:
                  print("\t{}".format(mail.group()))
            print("-"*60)
except FileNotFoundError:
      print("File does not exists:")
```

```
                  ===================================
                        Multi Threading in Python
                  ===================================
Index:
---------
=>Purpose of Multi Threading
=>Definition of Thread
=>Types of Applications
      1) Process Based Applications
      2) Thread Based Applications
=>Module Name required for Multi Threading
                       1) threading---Thread
=>Number of approaches for developing Thread Based Application
=>Programming Examples
---------------------------------------------------
=>Dead Locks Concept
=>Dead Locks Elimination Concept
=>Module Name required for Dead Locks
                       1) threading---Lock
=>Programming Examples
--------------------------------------------------------------------------
            ========================================
                 Introduction to Multi Threading
            ========================================
=>In the context of OS, we have a concept called Multi Tasking.
=>The main purpose of Multi Threading in python is that "To achieve the
Concurrent Execution ( Simultaneous / Parallel Execution ) "
=>Multi Threading is one of the specilized form of Multi Tasking of OS.
=>In this context , we have two types of applications. They are
            a) Process Based Applications
            b) Thread Based Applications.
---------------------------------------------
a) Process Based Applications:
---------------------------------------------
=>Process Based Applications execution environment always contains single
thread.
=>Process Based Applications provides Sequential Execution
=>Process Based Applications takes more execution time
=>Process Based Applications are treated as Heavy weight components.
Examples:- The application of C,CPP...etc are comes under Process Based
                             Applications.
Examples: The default Execution Environment of Python is also comes Under
Process
                  Based Application.
--------------------------------------------------------------------------
-----------------------------
b) Thread Based Applications:
----------------------------------------------
=>Thread Based Applications execution environment always contains
multiple  threads for performing multiple Operations concurrently.
=>Thread Based Applications provides concurrent execution.
=>Thread Based Applications takes Less execution time
=>Thread Based Applications are treated as Light weight components.
Examples:-- The application of PYTHON, JAVA...etc are comes under Thread
Based Applications.
```

Thread Based Applications

=>Definition of thread:

=>A flow of control is called thread

=>The purpose of thread is that "To execute any type of operation concurrently"

(or)

=>The purpose of thread is that "To perform the operations whose logic is written in the form of Functions / Methods "

=>When we write any python program, there exists two types of threads. They are

a) sub thread (or) child thread (or) Fore Ground Thread
b) main thread

=>The purpose of sub therad (or) Fore Ground Thread is that to execute the operations concurrently whose logic is written in the form of methods / functions.

=>The purpose of main thread is that to monitor the execution status of Sub Thread(s)

=>By default, we have single main thread only.

=>Programatically, There is a possibility of creating multiple sub threads and recommeded to have single main thread.

```
#threaddemo1.py
import threading
dftname=threading.current_thread().name
print("\ndefault Name  of thread={}".format(dftname))
print("Program execution started")
print("Hello Multi Threading program")
print("First class Multi Threading")
print("Program execution ended")
```

```
#threaddemo2.py
import threading
def  hello():
      tname1=threading.current_thread().name
      print("\nHello() executed by {}".format(tname1))
      print("i am from hello()")

def  hi():
      tname2=threading.current_thread().name
      print("\nHi() executed by {}".format(tname2))
      print("i am from hi()")
def  show():
      tname3=threading.current_thread().name
      print("\nshow() executed by {}".format(tname3))
      print("i am from show()")

#main program
dftname=threading.current_thread().name
print("\ndefault Name  of thread in main program={}".format(dftname))
hello()
hi()
show()
```

```
#threaddemo3.py
import time
import threading
def  squares(lst):
      tname1=threading.current_thread().name
      print("\nsqaures() executed by {}".format(tname1))
      for val in lst:
            print("square({})={}".format(val,val**2))
            time.sleep(1)

def  cubes(lst):
      tname2=threading.current_thread().name
      print("\ncubes() executed by {}".format(tname2))
      for val in lst:
            print("cubes({})={}".format(val,val**3))
            time.sleep(1)

#main program
bt=time.time()
dftname=threading.current_thread().name
print("\ndefault Name  of thread in main program={}".format(dftname))
lst=[2,8,-4,6,9,12,67,25]
squares(lst)
cubes(lst)
et=time.time()
print("\nExecution of time non-threading application={}".format(et-bt))
```

```
#threaddemo4.py
import time
import threading
def  squares(lst):
      tname1=threading.current_thread().name
      print("\nsqaures() executed by {}".format(tname1))# Thread-1
      for val in lst:
            print("square({})={}".format(val,val**2))
            time.sleep(1)

def  cubes(lst):
      tname2=threading.current_thread().name
      print("\ncubes() executed by {}".format(tname2)) # Thread-2
      for val in lst:
            print("cubes({})={}".format(val,val**3))
            time.sleep(1)

#main program
bt=time.time()
dftname=threading.current_thread().name
print("\ndefault Name  of thread in main program={}".format(dftname))
lst=[2,8,-4,6,9,12,67,25]
st1=threading.Thread(target=squares,args=(lst,) ) # creating child thread
rt1=threading.Thread(target=cubes,args=(lst,) )  # creating child thread
st1.name="Rossum"
rt1.name="Ranjan"
#send child threads to execute functions
st1.start()
rt1.start()
st1.join()
```

```
rt1.join()
et=time.time()
print("\nExecution of time threading application={}".format(et-bt))
```

==

Module Name for developing thread based applications

==

=>To develop any thread based applications, must use a pre-defined module called "threading".

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

=>Details of threading

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

=>Functions in threading Module

----

1) current_thread():

=>This Function is used for finding thread name which is running

```
      Syntax:-    varname=threading.current_thread().name
```

----

2) active_count():

=>This Function is used for counting number threads which running

```
      Syntax:      varname=threading.active_count()
```

=>Class Name in threading Module: Thread:

----

1) Thread(target,args): This Constructor is used for creating an object of child thread by specifying target function which is executed by child thread and also specifying values passing to target function in the form of tuple.

```
Syntax:-    childthreadname=threading.Thread(target=functioname,
args=(val1,val2..val-n)
Example:     t1=threading.Thread(target=generate,args=(10,) )
```

----

2) setName(str)   or name

This function is used for setting user-friendly name to thread instead of giving default thread name

```
           Syntax:      childthreadname.setName(str)
                             (OR)
           Syntax:   childthreadname . name=str

Examples:        t1.setName("Rossum")
                            or
                      t1.name="Rossum"
```

----

3) getName()   (or) name

=>This function is used for obtaing name of thread.

```
    =>Syntax:-     threadobj.getName()
                      (or)
                      threadobj.name
Example: -     t1=threading.Thread(target=multable,args=(19,))
                     print(t1.getName()) # Thread-1
                      (or)
                    print(t1.name)  #  Thread-1
```

----
4) run()
----
5) start():
This function is used for dispatching or sending the child thread to targeted function by passing the values as args in the form of tuple

```
Syntax:   childthreadname.start()
Example:    t1.start()
```

----
6) is_alive()
----
7) join(): This function is used for making the child threads to join with main thread after their completion.

```
Syntax:    childthreadname1.join()
           childthreadname2.join()
           ------------------------------------
           childthreadname-n.join()
```

===

Number of approaches to develop thread based applications

===

=>In Python Programming, we have 3 types of approaches to develop thread based applications. They are

1. By using Functional Programming Approach
2. By using Sub Class of Thread Class   (with Inheritance)
2. By Using Non-Class sub class of Thread class (without Inheritance)

----
1. By using Functional Programming Approach

Step-1: import threading module
Step-2:   define a function which contains logic executed by Child Thread
Step-3: create an object of therad class and it is called child thread.
Step-4: Dispatch the child thread to excuted the targeted Function.

Example:

```
import threading,time
def  generate(n):
      print("Number of Numbers:{}".format(n))
      ctname=threading.current_thread().name
      print("Name of Child Thread=",ctname)
      print("-"*50)
      for i in range(1,n+1):
            print("\tValue of i={}".format(i))
            time.sleep(1)
      print("-"*50)
```

```
#main porogram
mtname=threading.current_thread().name
print("Name of main thread={}".format(mtname))
t1=threading.Thread(target=generate, args=(10,) )#creating child thread
#t1.setName("Rs")
t1.name="ROssum"
t1.start()   # distaching the child thread
print("Number of active threads=",threading.active_count())
t1.join()
print("Line-23, Number of active threads=",threading.active_count())
```

===

2. By using Sub Class of Thread Class   (with Inheritance)

```
#defaultthreadex.py
import threading
tname=threading.current_thread().name
noc=threading.active_count()
print("Number of active threads=",noc)
print("default thread name=",tname)
print("This is a thread based program")
print("Hyd")
```

```
#program displaying 1 to 10 number after each and evevry second by using
threads
#approachex1.py
import threading,time
def  generate(n):
      print("Number of Numbers:{}".format(n))
      ctname=threading.current_thread().name
      print("Name of Child Thread=",ctname)
      print("-"*50)
      for i in range(1,n+1):
            print("\tValue of i={}".format(i))
            time.sleep(1)
      print("-"*50)

#main porogram
mtname=threading.current_thread().name
print("Name of main thread={}".format(mtname))
t1=threading.Thread(target=generate, args=(10,) )#creating child thread
#t1.setName("Rs")
t1.name="ROssum"
t1.start()   # dispaching the child thread
print("Number of active threads=",threading.active_count())
t1.join()
print("Line-23, Number of active threads=",threading.active_count())
```

```
#Program generating mul table by using thread( use functional approach)
#approachex12.py
import threading,time
def  multable(n):
      tname=threading.current_thread().name
      print("Name of child thread in multable()=",tname)
      if(n<=0):
            print("{} is invalid input:".format(n))
      else:
```

```
            print("-"*50)
            print("Mul Table for {}".format(n))
            print("-"*50)
            for i in range(1,11):
                  print("\t {} x {} = {}".format(n,i,n*i))
                  time.sleep(1)
            print("-"*50)
#main program
print("Number of active threads in this program before
start=",threading.active_count())
t1=threading.Thread(target=multable, args=(int(input("Enter a
number:")),))
print("Defult child thread name=",t1.name)  # getting child thread name
t1.name="Hyd"  # setting user-friendly thread name
print("Execution status of t1 before start=",t1.is_alive()) # False
t1.start()
print("Execution status of t1 after start=",t1.is_alive()) # True
print("Number of active threads in this
program=",threading.active_count())
t1.join()
print("\nLine-27-->Execution status of t1 after
completion=",t1.is_alive()) # True
print("\nLine-28-->Number of active threads in this
program=",threading.active_count())


#Approachno2.py
import threading  # step-1
#           step-2             step-3
class Hyd(threading.Thread):
      def  run(self):  #step-4
            print("i am from run()")
            print("Therad based Application")

#main program
print("Name of main thread=",threading.current_thread().name)
h=Hyd() # here 'h'  is an object of Hyd and considered as Child thread
print("execution status of h before start=",h.is_alive())
h.start()
print("execution status of h after start=",h.is_alive())

#CharGenEx1.py---Approch-1
import threading,time
def    chargeneration():
      line=input("Enter a line of text:")
      print("="*50)
      print("Given Line:{}".format(line))
      print("="*50)
      for ch in line:
            print("\t\t{}".format(ch))
            time.sleep(1)
      print("="*50)

#main program
t1=threading.Thread(target=chargeneration)
t1.start()

#CharGenEx2.py---Approch-2
```

```
import threading,time
class Char(threading.Thread):
      def  run(self):
            line=input("Enter a line of text:")
            l=list(line)
            print("="*50)
            print("Given Line:{}".format(line))
            print("="*50)
            for ch in line:
                  print("\t\tCharacter :{}  and Occurences={}".format(ch,
l.count(ch)))
                  time.sleep(1)
            print("="*50)

#main program
ch=Char()
ch.start()

#CharGenEx3.py---Approch-3
import threading,time
class Character:
      def  genchar(self,line):
                  l=list(line)
                  print("="*50)
                  print("Given Line:{}".format(line))
                  print("="*50)
                  for ch in line[::-1]:
                        print("\t\tCharacter :{}  and
Occurences={}".format(ch, l.count(ch)))
                        time.sleep(1)
                  print("="*50)

#main program
t1=threading.Thread(target=Character().genchar,args=(input("Enter a
line:"),))
t1.start()


#Program will display 1 to n numbers by using threads with OOPs
(Inheritance)
#NumGenEx1.py
import time
from threading import Thread  # step-1
#                 step-2      step-3
class Numbers(Thread):
      def   run(self):  # Overridden run()----Step-4
            n=int(input("Enter Number of Numbers to generate:"))
            if(n<=0):
                  print("{} is invalid input:".format(n))
            else:
                  print("Number within:{}".format(n))
                  for i in range(1,n+1):
                        print("\tVal of i={}".format(i))
                        time.sleep(1)
#main program
n=Numbers() # creating child thread---Step-5
n.start() # step-6
```

```
#Program will display 1 to n numbers by using threads with OOPs (without
Inheritance)
#NumGenEx2.py
import threading,time #step-1
class Numbers:  #step-2
      def  generate(self,n):  #step-3
            print("Name of child
thread=",threading.current_thread().name) # Thread-1
            if(n<=0):
                  print("{} is invalid input:".format(n))
            else:
                  print("="*60)
                  print("Number within:{}".format(n))
                  print("="*60)
                  for i in range(1,n+1):
                        print("\tvalue of i={}".format(i))
                        time.sleep(1)
                  else:
                        print("="*60)

#main program
n=Numbers() #step-4
t1=threading.Thread(target=n.generate,args=(int(input("Enter a
number:")),) ) #step-5
t1.start() #step-6

#Program will display n to 1 numbers by using threads with OOPs (with out
Inheritance)
#NumGenEx3.py
import threading,time
class Numbers:
      def  generate(self,n):
            print("Name of child
thread=",threading.current_thread().name) # Thread-1
            if(n<=0):
                  print("{} is invalid input:".format(n))
            else:
                  print("="*60)
                  print("Number within:{}".format(n))
                  print("="*60)
                  for i in range(n,0,-2):
                        print("\tvalue of i={}".format(i))
                        time.sleep(1)
                  else:
                        print("="*60)

#main program
#n=Numbers()
t1=threading.Thread(target=Numbers().generate,args=(int(input("Enter a
number:")),) )
t1.start()
```

==

Synchronization in Multi Threading

(OR)

Locking concept in Threading

==

=>When multiple threads are operating / working on the same resource(function / method) then by default we get dead lock result / race condition / wrong result / non-thread safety result.
=>To overcome this dead lock problems, we must apply the concept Synchronization concept.
=>The advantage of synchronization concept is that to avoid dead lock result and provides Thread Safety Result.
=>In Python Programming, we can obtain synchronization concept by using locking and un-locking concept.

=>Steps for implementing Synchronization Concept:

----

1) obtain / create an object of Lock class, which is present in threading module.

Stntax:-

```
lockobj=threading.Lock()
```

2) To obtain the lock on the sharable resource, we must use acquire()

Syntax:

```
lockobj.acquire()
```

Once current object acquire the lock, other objects are made wait until curent object releases the lock.

3) To un-lock the sharable resource/current object, we must use release()

Syntax:

```
lockobj.release()
```

Once current object releases the lock, other objects are permitted into shrable resource.

This process of aquiring the releasing the lock will be continued until all the objects completed their execution.

```
#nonlockingex1.py
import threading , time
def  multable(n):
	print("-"*50)
	print("Child Thread Name=",threading.current_thread().name)
	print("Mul Table for {}  ".format(n))
	for i in range(1,11):
		print("{} x {}={}".format(n,i,n*i))
		time.sleep(1)
	print("-"*50)
#main program
t1=threading.Thread(target=multable,args=(5,))
t2=threading.Thread(target=multable,args=(15,))
t3=threading.Thread(target=multable,args=(19,))
t4=threading.Thread(target=multable,args=(7,))
t1.start()
t2.start()
t3.start()
t4.start()

#nonlockingex2.py
import threading , time
```

```
class MulTab(threading.Thread):
      def  setvalue(self,n):
            self.n=n
      def  run(self):
            print("-"*50)
            print("Child Thread Name=",threading.current_thread().name)
            print("Mul Table for {}  ".format(self.n))
            for i in range(1,11):
                  print("{} x {}={}".format(self.n,i,self.n*i))
                  time.sleep(1)
            print("-"*50)
#main program
#create multiple child threads
t1=MulTab()
t2=MulTab()
t3=MulTab()
t4=MulTab()
#set values
t1.setvalue(12)
t2.setvalue(14)
t3.setvalue(2)
t4.setvalue(19)
#dispatch the therads
t1.start()
t2.start()
t3.start()
t4.start()


#nonlockingex3.py
import threading , time
class MulTab:
      def __init__(self,n):
            self.n=n
      def  multable(self):
            print("-"*50)
            print("Child Thread Name=",threading.current_thread().name)
            print("Mul Table for {}  ".format(self.n))
            for i in range(1,11):
                  print("{} x {}={}".format(self.n,i,self.n*i))
                  time.sleep(1)
            print("-"*50)
#main program
MulTab.getlockobj()
m1=MulTab(15)
m2=MulTab(4)
m3=MulTab(15)
#create multiple child threads
t1=threading.Thread(target=m1.multable)
t2=threading.Thread(target=m2.multable)
t3=threading.Thread(target=m3.multable)

#dispatch the therads
t1.start()
t2.start()
t3.start()

#lockingex1.py
```

```
import threading , time
k=threading.Lock() # Step-1
def  multable(n):
      k.acquire() # step-2
      print("-"*50)
      print("Child Thread Name=",threading.current_thread().name)
      print("Mul Table for {}  ".format(n))
      for i in range(1,11):
            print("{} x {}={}".format(n,i,n*i))
            time.sleep(1)
      print("-"*50)
      k.release() # Step-3
#main program
t1=threading.Thread(target=multable,args=(5,))
t2=threading.Thread(target=multable,args=(16,))
t3=threading.Thread(target=multable,args=(13,))
t4=threading.Thread(target=multable,args=(27,))
t1.start()
t2.start()
t3.start()
t4.start()


#lockingex2.py
import threading , time
class MulTab(threading.Thread):
      L=threading.Lock()  # Class Level Data Member--Step-1
      def  setvalue(self,n):
            self.n=n
      def  run(self):
            MulTab.L.acquire()  # step-2
            print("-"*50)
            print("Child Thread Name=",threading.current_thread().name)
            print("Mul Table for {}  ".format(self.n))
            for i in range(1,11):
                  print("{} x {}={}".format(self.n,i,self.n*i))
                  time.sleep(1)
            print("-"*50)
            MulTab.L.release() # Step-3
#main program
#create multiple child threads
t1=MulTab()
t2=MulTab()
t3=MulTab()
t4=MulTab()
#set values
t1.setvalue(12)
t2.setvalue(14)
t3.setvalue(2)
t4.setvalue(19)
#dispatch the therads
t1.start()
t2.start()
t3.start()
t4.start()

#lockingex3.py
import threading , time
```

```
class MulTab:
      @classmethod
      def    getlockobj(cls):
            cls.L=threading.Lock()
      def __init__(self,n):
            self.n=n
      def  multable(self):
            self.L.acquire()
            print("-"*50)
            print("Child Thread Name=",threading.current_thread().name)
            print("Mul Table for {}  ".format(self.n))
            for i in range(1,11):
                  print("{} x {}={}".format(self.n,i,self.n*i))
                  time.sleep(1)
            print("-"*50)
            self.L.release()
#main program
MulTab.getlockobj()
m1=MulTab(15)
m2=MulTab(4)
m3=MulTab(15)
#create multiple child threads
t1=threading.Thread(target=m1.multable)
t2=threading.Thread(target=m2.multable)
t3=threading.Thread(target=m3.multable)

#dispatch the therads
t1.start()
t2.start()
t3.start()
```

===

Numpy

===

Introduction to Numpy:

=>Numpy stands for Numerical Python.
=>Numpy is one of the pre-defined third party module / Library.
=>To use numpy as a part of our python program, we must install numpy module explicitly by using a tool called pip and it present in (C:\Users\nareshit\AppData\Local\Programs\Python\Python39\Scripts)
=>Syntax for installing any module:

```
pip    install    module-name
```

=>Example: Install numpy module

```
pip   install   numpy
```

=>To use numpy as part of our program, we must import numpy module.
=>A Numpy module is a collection of Variables, Functions and Classes.

===

History of Numpy:

=>Numpy was developed by studying existing module called "Numeric Library"(origin for development of numpy module)
=>The Numeric Library was developed by JIM HUNGUNIAN
=>The Numeric Library was not able to solve complex maths calculations.

=>Numpy module developed by TRAVIS OLIPHANT
=>Numpy Module developed in the year 2005
=>Numpy Module developed in C and PYTHON languages.
==
Uses of NumPy:
----
1) An alternative for the lists and arrays in Python and NumPy arrays are stored at one continuous place in memory unlike lists, so processeing, accessing and manipulate them very efficiently.

2) NumPy maintains minimal memory:

3) Using NumPy for multi-dimensional arrays:

4) Mathematical operations with NumPy are easy.

===
Python Traditional List VS Numpy Module
===

Similarities of python Traditional List VS Numpy Module:

=>An object of list used to store multiple values of same type or different type and both types (unique +duplicates) in single object.
=>In Numpy Programming, the data is organized in the object of "ndarray", which is one of the pre-defined class in numpy module.
=>The objects of numpy and list are mutable (changes can takes place)

Differences between Python Traditional List and Numpy Module:

=>An object of list contains both homogeneous and hetrogeneous values where as an object of ndarray of numpy can store only similar type of values(even we store different values, internally they are treated as similar type).
=>On the object of list, we can't perform Vector Operations. where as on the object of ndarray, we can perform Vector based operations.
=>In large sampling of data, List based applications takes more memory space where ndarray object takes less memory space.
=>List based applications are not effiecient where as numpy based applications are efficient.
=>List object can't perform complex mathematical operations where as an object of ndarray can perform complex mathematical operations.
==========================X===================================

===
Python Traditional List VS Numpy Module
===

Similarities of python Traditional List VS Numpy Module:

=>An object of list used to store multiple values of same type or different type and both types (unique +duplicates) in single object.
=>In Numpy Programming, the data is organized in the object of "ndarray", which is one of the pre-defined class in numpy module.

=>The objects of numpy and list are mutable (changes can takes place)

Differences between Python Traditional List and Numpy Module:

=>An object of list contains both homogeneous and hetrogeneous values where as an object of ndarray of numpy can store only similar type of values(even we store different values, internally they are treated as similar type).
=>On the object of list, we can't perform Vector Operations. where as on the object of ndarray, we can perform Vector based operations.
=>In large sampling of data, List based applications takes more memory space where ndarray object takes less memory space.
=>List based applications are not effiecient where as numpy based applications are efficient.
=>List object can't perform complex mathematical operations where as an object of ndarray can perform complex mathematical operations.
==========================X==================================

ndarray

=>'ndarray' is one of the pre-defined class present in numpy module
=>An object of 'ndarray' allows us to store the data in the form of single (or) one dimensional and multi dimesional in the entire numpy module.
=>To create an object of ndarray, we have 7 approaches.

1) array()
2) arange()
3) zeros()
4) ones()
5) full()
6) eye()
7) identity()

=>All the above functions are present in numpy module.

---

1) array():

=>It is used for connverting any object type of python into an object of ndarray.
=>Syntax:-

```
varname=numpy.array(object, dtype)
```

=>varname is represents an object of ndarray.
=>numpy is a module name
=>array() is a pre-defined function present in numpy module.
=>object can be any Collection Types (list, tuple,set,frozenset,dict..)...etc

---

Examples:

```
>>> a=10
>>> b=np.array(a)
>>> b.dtype-------------dtype('int32')
>>> b.ndim----------------0
```

```
>>> b.shape------------- ()
>>> l1=[10,20,30,40]
>>> print(l1,type(l1))----------[10, 20, 30, 40] <class 'list'>
>>> a=np.array(l1)
>>> print(a, type(a))---------[10 20 30 40] <class 'numpy.ndarray'>
>>> a-------array([10, 20, 30, 40])
>>> print(a.ndim)----1
>>> print(a.shape)----(4,)
------------------------------------------
>>> l1=[10,20,30,40]
>>> a=np.array(l1,dtype='float')
>>> print(a,type(a))----[10. 20. 30. 40.] <class 'numpy.ndarray'>
>>> a--------------array([10., 20., 30., 40.])
>>> l1=[12.3,34.5,56.78]
>>> a=np.array(l1,dtype='float')
>>> print(a, type(a))--------[12.3  34.5  56.78] <class 'numpy.ndarray'>
>>> a------------array([12.3 , 34.5 , 56.78])
>>> print(a.ndim)-------1
>>> print(a.shape)-------(3,)
>>> print(a.dtype)------float64
-----------------
>>> l1=[10,10.25,24,23.45,30]
>>> a=np.array(l1)
>>> print(a)------[10.   10.25 24.   23.45 30.  ]
>>> a---------array([10.  , 10.25, 24.  , 23.45, 30.  ])
>>> print(a.dtype)-------float64
>>> l2=[10,20,30]
>>> a=np.array(l2)
>>> a---------array([10, 20, 30])
>>> print(a.dtype)--------int32
---------------------------------
>>> l1=["RS","RT","JG"]
>>> a=np.array(l1)
>>> a
array(['RS', 'RT', 'JG'], dtype='<U2')
>>> print(a.dtype)----<U2
>>> l1=["Rossum","RT","JG"]
>>> a=np.array(l1)
>>> a
array(['Rossum', 'RT', 'JG'], dtype='<U6')
----------------------------------
>>> l1=[10,"KVR",23.45,True,2+3j]
>>> a=np.array(l1)
>>> print(a)----['10' 'KVR' '23.45' 'True' '(2+3j)']
>>> print(a.ndim)------1
>>> print(a.shape)----(5,)
>>> print(a.dtype)---- <U64
-----------------------------------------------------------------
>>> l1=[ [10,20], [30,40] ]
>>> a=np.array(l1)
>>> a------->array([[10, 20],
                        [30, 40]])
>>> print(a.ndim)
2
>>> print(a.shape)
(2, 2)
>>> print(a.dtype)----------------int32
-------------------------------------
```

```
>>> l1=[[10,20,30], [40,50,60],[70,80,90]]
>>> a=np.array(l1)
>>> a
array([[10, 20, 30],
       [40, 50, 60],
       [70, 80, 90]])
>>> print(a.ndim)
2
>>> print(a.shape)
(3, 3)
>>> print(a.dtype)
int32
-------------------------------------------------------
>>> l1=[[10,20,30], [40,50,60]]
>>> a=np.array(l1)
>>> a
array([[10, 20, 30],
       [40, 50, 60]])
>>> print(a.ndim)
2
>>> print(a.shape)
(2, 3)
>>> b=a.reshape(3,2)
>>> b
array([[10, 20],
       [30, 40],
       [50, 60]])
>>> print(b.ndim)
2
>>> print(b.shape)
(3, 2)
----------------------------------------
>>> l1=[[[10,20],[30,40]],[[50,60],[70,80]] ]
>>> a=np.array(l1)
>>> a
array([[[10, 20],
             [30, 40]],

            [[50, 60],
             [70, 80]]])
>>> print(a.ndim)
3
>>> print(a.shape)
(2, 2, 2)
>>> print(a[0])
                  [[10 20]
                   [30 40]]
>>> print(a[1])
                  [[50 60]
                   [70 80]]
---------------------------------------------------------------------
```

2) arange():

=>Syntax:-   ndarrayobjname=numpy.arange(begin,end,step, dtype)

=>This is function is used for generating 1-Dimensional Array of Values but we can't create 2-Dimensional Array.

=>To convert 1-Dimensional Array of Values of ndarray object into 2-Dimensional Array, we use reshape().

```
---------------------
=>Examples:
---------------------
>>> a=np.arange(9)
>>> print(a)----------[0 1 2 3 4 5 6 7 8]
>>> a-------array([0, 1, 2, 3, 4, 5, 6, 7, 8])
>>> print(type(a))----<class 'numpy.ndarray'>
>>> print(a.ndim)
1
>>> print(a.shape)
(9,)
>>> print(a.dtype)
int32
>>> b=a.reshape(3,3)
>>> print(b)
[[0 1 2]
 [3 4 5]
 [6 7 8]]
>>> b
array([[0, 1, 2],
       [3, 4, 5],
       [6, 7, 8]])
>>> print(b.ndim,b.shape,b.dtype)----2  (3, 3)   int32
---------------------------------------
>>> a=np.arange(10,22)
>>> print(a)
[10 11 12 13 14 15 16 17 18 19 20 21]
>>> a
array([10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21])
>>> print(a.ndim,a.shape)
1 (12,)
>>> b=a.reshape(4,3)
>>> b
array([[10, 11, 12],
       [13, 14, 15],
       [16, 17, 18],
       [19, 20, 21]])
>>> c=a.reshape(3,4)
>>> c
array([[10, 11, 12, 13],
       [14, 15, 16, 17],
       [18, 19, 20, 21]])
>>> print(b.ndim,b.shape)
2 (4, 3)
>>> print(c.ndim,c.shape)
2 (3, 4)
>>> d=a.reshape(2,6)
>>> d
array([[10, 11, 12, 13, 14, 15],
       [16, 17, 18, 19, 20, 21]])
>>> e=a.reshape(6,2)
>>> e
array([[10, 11],
       [12, 13],
       [14, 15],
       [16, 17],
```

```
       [18, 19],
       [20, 21]])
>>> f=a.reshape(12,1)
>>> f
array([[10],
       [11],
       [12],
       [13],
       [14],
       [15],
       [16],
       [17],
       [18],
       [19],
       [20],
       [21]])
```
----

3) zeros():

=>This function is used for building zero matrix (or) creating ndarray objct with zeros by specfying its shape.

Syntax:-

```
      ndarrayobj=numpy.zeros(shape,dtype)
```

Here shape can be either 1-dimensional (or) 2-dimensional
here specfying dtype is optional.

Examples:

```
>>> a=np.zeros(6)
>>> a
array([0., 0., 0., 0., 0., 0.])
>>> b=a.reshape(3,2)
>>> b
array([[0., 0.],
       [0., 0.],
       [0., 0.]])
>>> c=b.reshape(2,3)
>>> c
array([[0., 0., 0.],
       [0., 0., 0.]])
>>> a=np.zeros(12,dtype=int)
>>> a
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
>>> b=a.reshape(3,4)
>>> c=a.reshape(4,3)
>>> print(b)
[[0 0 0 0]
 [0 0 0 0]
 [0 0 0 0]]
>>> print(c)
[[0 0 0]
 [0 0 0]
 [0 0 0]
 [0 0 0]]
>>> print(type(a),type(b),type(c))
<class 'numpy.ndarray'> <class 'numpy.ndarray'> <class 'numpy.ndarray'>
```

```
>>> a=np.zeros((3,3) )
>>> a=np.zeros(shape=(3,3),dtype=int )
>>> a
array([[0, 0, 0],
       [0, 0, 0],
       [0, 0, 0]])
>>> a=np.zeros(shape=(2,3),dtype=int )
>>> a
array([[0, 0, 0],
       [0, 0, 0]])
>>> a=np.zeros(shape=(4,2),dtype=int )
>>> a
array([[0, 0],
       [0, 0],
       [0, 0],
       [0, 0]])
------------------------------------------------------------------------
------------------------
```

4)ones()

=>This function is used building a matrix with 1's (or) creating an object ndarray by initlizing with all 1's.

=>Syntax:-   ndarrayobj=numpy.ones(shape,dtype)

Examples:

```
>>> a=np.ones(6)
>>> print(a,type(a))
[1. 1. 1. 1. 1. 1.] <class 'numpy.ndarray'>
>>> print(a.ndim,a.shape,a.dtype)
1 (6,) float64
>>> print(a.reshape(3,2))
[[1. 1.]
 [1. 1.]
 [1. 1.]]
>>> print(a.reshape(2,3))
[[1. 1. 1.]
 [1. 1. 1.]]
>>> a=np.ones(8,dtype=int)
>>> print(a,type(a))
[1 1 1 1 1 1 1 1] <class 'numpy.ndarray'>
>>> print(a.reshape(4,2))
[[1 1]
 [1 1]
 [1 1]
 [1 1]]
>>> a=np.ones( (3,4),dtype=int)
>>> a
array([[1, 1, 1, 1],
       [1, 1, 1, 1],
       [1, 1, 1, 1]])
>>> print(a.reshape(4,3))
[[1 1 1]
 [1 1 1]
 [1 1 1]
 [1 1 1]]
>>> a=np.ones( (2,3,4),dtype=int)
```

```
>>> a
array([[[1, 1, 1, 1],
        [1, 1, 1, 1],
        [1, 1, 1, 1]],

       [[1, 1, 1, 1],
        [1, 1, 1, 1],
        [1, 1, 1, 1]]])
>>> a=np.ones( (3,3,4),dtype=int)
>>> a
array([[[1, 1, 1, 1],
        [1, 1, 1, 1],
        [1, 1, 1, 1]],

       [[1, 1, 1, 1],
        [1, 1, 1, 1],
        [1, 1, 1, 1]],

       [[1, 1, 1, 1],
        [1, 1, 1, 1],
        [1, 1, 1, 1]]])
>>> print(a.ndim)
3
>>> print(a.shape)
(3, 3, 4)
```

----

5) full():

=>Syntax:-   ndarrayobj=numpy.full(shape, fill_value, dtype)

=>This function is used for generating a matrix by specifying user choice value (or) building an object of ndarray with our value.

=>"fill_value" is programmer-defined value

Examples:

```
>>> a=np.full(3,4,dtype=int)
>>> print(a, type(a))
[4 4 4] <class 'numpy.ndarray'>
>>> a=np.full(12,6,dtype=int)
>>> print(a, type(a))
[6 6 6 6 6 6 6 6 6 6 6 6] <class 'numpy.ndarray'>
>>> print(a.reshape(4,3))
[[6 6 6]
 [6 6 6]
 [6 6 6]
 [6 6 6]]
>>> a.reshape(3,4)
array([[6, 6, 6, 6],
       [6, 6, 6, 6],
       [6, 6, 6, 6]])
>>> a=np.full((4,5),8, dtype=int)
>>> a
array([[8, 8, 8, 8, 8],
       [8, 8, 8, 8, 8],
       [8, 8, 8, 8, 8],
       [8, 8, 8, 8, 8]])
```

```
>>> a=np.full((3,2,2),8, dtype=int)
>>> a
array([[[8, 8],
        [8, 8]],

       [[8, 8],
        [8, 8]],

       [[8, 8],
        [8, 8]]])
```

----

6) eye()

Syntax:-   ndarrayobj=numpy.eye(N,M=None,K=0,dtype)

=>Here N represents No. of Rows

=>Here M represents No. of Columns. If we don't specify the M value then N
value will be considered as M value.

=>If we take M value explicitly then It will form Possible Identity matrix (NXM)and remaining elements filled with zeros.

=>Here K represents Principal Diagnal

(if K=0 then it is Pricipal Diagnal and it is default)

(if K=-1,-2...then it is considered as bellow Principal Diagnal)

(if K=1,2 ...then it is considered as above Principal Diagnal)

Examples:

```
>>> np.eye(3)
array([[1., 0., 0.],
       [0., 1., 0.],
       [0., 0., 1.]])
>>> np.eye(3,dtype=int)
array([[1, 0, 0],
       [0, 1, 0],
       [0, 0, 1]])

>>> np.eye(3,4,dtype=int)
array([[1, 0, 0, 0],
            [0, 1, 0, 0],
            [0, 0, 1, 0]])

>>> np.eye(4,3,dtype=int)
array([[1, 0, 0],
            [0, 1, 0],
            [0, 0, 1],
            [0, 0, 0]])
>>> np.eye(5,6,dtype=int)
array([[1, 0, 0, 0, 0, 0],
       [0, 1, 0, 0, 0, 0],
       [0, 0, 1, 0, 0, 0],
       [0, 0, 0, 1, 0, 0],
       [0, 0, 0, 0, 1, 0]])
>>> np.eye(5,6,k=-1,dtype=int)
array([[0, 0, 0, 0, 0, 0],
       [1, 0, 0, 0, 0, 0],
       [0, 1, 0, 0, 0, 0],
       [0, 0, 1, 0, 0, 0],
       [0, 0, 0, 1, 0, 0]])
```

```
>>> np.eye(5,6,k=-2,dtype=int)
array([[0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0],
       [1, 0, 0, 0, 0, 0],
       [0, 1, 0, 0, 0, 0],
       [0, 0, 1, 0, 0, 0]])
>>> np.eye(5,6,k=-3,dtype=int)
array([[0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0],
       [1, 0, 0, 0, 0, 0],
       [0, 1, 0, 0, 0, 0]])
>>> np.eye(5,6,k=1,dtype=int)
array([[0, 1, 0, 0, 0, 0],
       [0, 0, 1, 0, 0, 0],
       [0, 0, 0, 1, 0, 0],
       [0, 0, 0, 0, 1, 0],
       [0, 0, 0, 0, 0, 1]])
>>> np.eye(5,6,k=2,dtype=int)
array([[0, 0, 1, 0, 0, 0],
       [0, 0, 0, 1, 0, 0],
       [0, 0, 0, 0, 1, 0],
       [0, 0, 0, 0, 0, 1],
       [0, 0, 0, 0, 0, 0]])
>>> np.eye(5,6,k=3,dtype=int)
array([[0, 0, 0, 1, 0, 0],
       [0, 0, 0, 0, 1, 0],
       [0, 0, 0, 0, 0, 1],
       [0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0]])
>>> np.eye(5,6,k=4,dtype=int)
array([[0, 0, 0, 0, 1, 0],
       [0, 0, 0, 0, 0, 1],
       [0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0]])
```

----

7)identity()

=>This function generates only Square Identity Matrix

Syntax:-   ndarrayobj=numpy.identity(n,dtype)

Here 'n' represent nxn identity matrix and it will be considered as Rows and columns.

Example:

```
>>> a=np.identity(3)
>>> print(a, type(a))
[[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]] <class 'numpy.ndarray'>
>>> a
array([[1., 0., 0.],
       [0., 1., 0.],
       [0., 0., 1.]])
>>> a=np.identity(3,dtype=int)
```

```
>>> a
array([[1, 0, 0],
       [0, 1, 0],
       [0, 0, 1]])
>>> a=np.identity(4,dtype=int)
>>> a
array([[1, 0, 0, 0],
       [0, 1, 0, 0],
       [0, 0, 1, 0],
       [0, 0, 0, 1]])
```
==

```
#performance.py
import numpy as np
import sys
l1=[10,20,30,40,50]
print("Type of l1=",type(l1))
a=np.array(l1)
print("Type of a=",type(a))
print("------------------------------------")
print("Memory Size of l1=", sys.getsizeof(l1))
print("Memory Size of a=", sys.getsizeof(a))

#performance1.py
import numpy as np
import sys
l1=[10,20,30,40,50,56,78,89,34,56,78,99,123,45,67,89,34,56,234,567,45,234
,56,78,10,20,30,40,50,56,78,89,34,56,78,99,123,45,67,89,34,56,234,567]
print("Type of l1=",type(l1))
a=np.array(l1)
print("Type of a=",type(a))
print("------------------------------------")
print("Memory Size of l1=", sys.getsizeof(l1))
print("Memory Size of a=", sys.getsizeof(a))

#performance2.py
import numpy as np
import sys
l1=[10,20,30,40,50,56,78,89,34,56,78,99,123,45,67,89,34,56,234,567,45,234
,56,78,10,20,30,40,50,56,78,89,34,56,78,99,123,45,67,89,34,56,234,567,"Py
thon"]
print("Type of l1=",type(l1))
a=np.array(l1,dtype="object")
print("Type of a=",type(a))
print("------------------------------------")
print("Memory Size of l1=", sys.getsizeof(l1))
print("Memory Size of a=", sys.getsizeof(a))
```

===
Numpy--Arithmetic Operations
===

=>On the objects of ndarray, we can apply all types of Arithmetic Operators.
=>To perform Arithmetic Operations on the objects of ndarray in numpy programming, we use the following functions.

a) add()
b) subtract()
c) multiply()

```
            d) dot()
            e) divide()
            f) floor_divide()
            g) mod()
            h) power()
```
=>All the arithmetic Function can also be perfomed w.r.t Arithmetic Operators.

----

a) add():

Syntax:- varname=numpy.add(ndarrayobj1, ndarrayobj2)

=>This function is used for adding elements of ndarrayobj1, ndarrayobj2 and result can be displayed

Examples:

```
>>> l1=[ [10,20],[30,40] ]
>>> l2=[[1,2],[3,4]]
>>> a=np.array(l1)
>>> b=np.array(l2)
>>> a
      array([[10, 20],
             [30, 40]])
>>> b
      array([[1, 2],
             [3, 4]])
>>> c=np.add(a,b)
>>> c
            array([[11, 22],
                     [33, 44]])
```

```
 >>> x=np.array([[1,2,3],[4,5,6]])
>>> x
            array([[1, 2, 3],
                   [4, 5, 6]])
>>> y=np.array([4,4,4])
>>> y
            array([4, 4, 4])
>>> z=x+y
>>> z
      array([[ 5,  6,  7],
             [ 8,  9, 10]])
>>> z=np.add(x,y)
>>> z
      array([[ 5,  6,  7],
             [ 8,  9, 10]])
>>> x
      array([[1, 2, 3],
             [4, 5, 6]])
>>> k=np.array([[2,3],[4,5]])
>>> k
      array([[2, 3],
             [4, 5]])
>>> kvr=np.add(x,k)----ValueError: operands could not be broadcast
together                                with shapes (2,3) (2,2)
```

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

```
>>> l2=[[1,2],[3,4]]
>>> a=np.array(l1)
>>> b=np.array(l2)
>>> a
           array([[10, 20],
                  [30, 40]])
>>> b
           array([[1, 2],
                  [3, 4]])
>>> c=a+b  # we used operator + instead of add()
>>> c
      array([[11, 22],
             [33, 44]])
```

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

b) subtract()

Syntax:-   varname=numpy.subtract(ndarrayobj1, ndarrayobj2)

=>This function is used for subtracting elements of ndarrayobj1, ndarrayobj2 and result can be displayed

Examples:

```
>>> l1=[[10,20],[30,40]]
>>> l2=[[1,2],[3,4]]
>>> a=np.array(l1)
>>> b=np.array(l2)
>>> a
array([[10, 20],
       [30, 40]])
>>> b
array([[1, 2],
       [3, 4]])
>>> c=np.subtract(a,b)
>>> c
array([[ 9, 18],
       [27, 36]])
-----------------------------------
>>> d=a-b    # we used operator - instead of subtract()
>>> d
array([[ 9, 18],
       [27, 36]])
```

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

c) multiply():

Syntax:-   varname=numpy.multiply(ndarrayobj1, ndarrayobj2)

=>This function is used for performing element-wise multiplication of ndarrayobj1, ndarrayobj2 and result can be displayed

Examples:

```
>>> l1=[[1,2],[3,4]]
>>> l2=[[5,6],[4,3]]
>>> a=np.array(l1)
>>> b=np.array(l2)
>>> a
array([[1, 2],
       [3, 4]])
>>> b
array([[5, 6],
```

```
       [4, 3]])
>>> c=np.multiply(a,b)
>>> c
array([[ 5, 12],
       [12, 12]])
----------------------------------------------
>>> e=a*b   # we used operator * instead of multiply()
>>> e
array([[ 5, 12],
       [12, 12]])
-----------------------------------------
```

d) dot()
=>To perform Matrix Multiplication, we use dot()

Syntax:-   varname=numpy.dot(ndarrayobj1, ndarrayobj2)

=>This function is used for performing actual matrix multiplication of ndarrayobj1, ndarrayobj2 and result can be displayed
Examples:

Examples:

```
>>> l1=[[1,2],[3,4]]
>>> l2=[[5,6],[4,3]]
>>> a=np.array(l1)
>>> b=np.array(l2)
>>> a
      array([[1, 2],
             [3, 4]])
>>> b
      array([[5, 6],
             [4, 3]])
>>> d=np.dot(a,b)
>>> d
      array([[13, 12],
             [31, 30]])
----------------------------------------------------------------------------
```

e) divide()

Syntax:-   varname=numpy.divide(ndarray1,ndarry2)
=>This function is used for performing element-wise division of ndarrayobj1, ndarrayobj2 and result can be displayed

```
>>> l1=[[10,20],[30,40]]
>>> l2=[[1,2],[3,4]]
>>> a=np.array(l1)
>>> b=np.array(l2)
>>> a
      array([[10, 20],
             [30, 40]])
>>> b
      array([[1, 2],
             [3, 4]])
>>> c=np.divide(a,b)
>>> c
      array([[10., 10.],
             [10., 10.]])
---------------------------------------------------------------
```

```
>>> d=a/b    # we used operator / instead of divide()
>>> d
      array([[10., 10.],
             [10., 10.]])
```
----

f) floor_divide()

Syntax:-   varname=numpy.floor_divide(ndarray1,ndarry2)

=>This function is used for performing element-wise floor division of ndarrayobj1, ndarrayobj2 and result can be displayed

```
>>> l1=[[10,20],[30,40]]
>>> l2=[[1,2],[3,4]]
>>> a=np.array(l1)
>>> b=np.array(l2)
>>> a
      array([[10, 20],
             [30, 40]])
>>> b
      array([[1, 2],
             [3, 4]])
>>> c=np.floor_divide(a,b)
>>> c
      array([[10, 10],
             [10, 10]])
---------------------------------------------------------------
>>> d=a//b    # we used operator // instead of floor_divide()
>>> d
      array([[10, 10],
             [10, 10]])
```
----

g) mod()

Syntax:-   varname=numpy.mod(ndarray1,ndarry2)

=>This function is used for performing element-wise modulo division of ndarrayobj1, ndarrayobj2 and result can be displayed

----

Examples:

```
>>> l1=[[10,20],[30,40]]
>>> l2=[[1,2],[3,4]]
>>> a=np.array(l1)
>>> b=np.array(l2)
>>> a
      array([[10, 20],
             [30, 40]])
>>> b
      array([[1, 2],
             [3, 4]])
>>> c=np.mod(a,b)
>>> c
      array([[0., 0.],
             [0., 0.]])
```

=>We can also do with operator %

```
>>> e=a%b
>>> e
```

```
     array([[0, 0],
              [0, 0]],     dtype=int32)
```
----

h) power():

Syntax:-   varname=numpy.power(ndarray1,ndarry2)

=>This function is used for performing element-wise exponential of ndarrayobj1, ndarrayobj2 and result can be displayed

----

```
>>> l1=[[10,20],[30,40]]
>>> l2=[[1,2],[3,4]]
>>> a=np.array(l1)
>>> b=np.array(l2)
>>> a
     array([[10, 20],
            [30, 40]])
>>> b
     array([[1, 2],
            [3, 4]])
>>>c=np.power(a,b)
>>>print(c)
     array([[     10,     400],
            [  27000, 2560000]],
```

```
>>> f=a**b   # Instead of using  power() we can use ** operator
>>> f
     array([[     10,     400],
            [  27000, 2560000]],    dtype=int32)
```
==================================X======================

==

Numpy--Statstical Functions

==

=>The most essential Numpy--Statstical Functionsare

- a) amax()
- b) amin()
- c) mean()
- d) median()
- e) var()
- f) std ()

----

a) amax() b) amin()

=>These functions are used for finding max and min elements from given ndarray object

Syntax1:-   numpy.amax(array1) # here without axis all the elements of matix
                        numpy.amin(array2) # will be considered

Syntax1:-   numpy.amax(array1, axis=0) # here axis =0 represents Columns of matrix numpy.amin(array2,axis=1) # here axis =1 represents Rows of matrix

```
>>> a=np.array([[10,20,30],[40,50,60],[12,13,14]])
>>> print(a)
[[10 20 30]
 [40 50 60]
 [12 13 14]]
>>> a
array([[10, 20, 30],
       [40, 50, 60],
       [12, 13, 14]])
>>> np.amax(a)
60
>>> np.amin(a)
10
>>> np.amax(a,axis=0)
array([40, 50, 60])
>>> np.amax(a,axis=1)
array([30, 60, 14])
>>> np.amin(a,axis=0)
array([10, 13, 14])
>>> np.amin(a,axis=1)
array([10, 40, 12])
```

===

c) mean():

=>mean is nothing but sum of all elements of ndarray divided by total number of elements.

Examples:

```
>>> a=np.array([[2,1,3],[6,5,4],[3,5,2]])
>>> print(a)
[[2 1 3]
 [6 5 4]
 [3 5 2]]
>>> mr=np.mean(a)
>>> print("mean=",mr)
mean= 3.4444444444444446
>>> cm=np.mean(a,axis=0)
>>> print("column mean=",cm)
column mean= [3.66666667 3.66666667 3.        ]
>>> rmr=np.mean(a,axis=1)
>>> print("row mean=",rmr)
row mean= [2.         5.         3.33333333]
```

===

d) median():

=>Selecting the center element after sorting in ascending oder.
=>If number of elements are EVEN then sort them ascending order and find sum of two middle elements/2
=>If number of elements are ODD then sort them ascending order and take middle element

Examples:

```
>>> a=np.array([[2,1,3],[6,5,4],[3,5,2]])- #   1  2  2  3  3  4  5   5
6
>>> print(a)                                  # here middle element=3
[[2 1 3]
 [6 5 4]
 [3 5 2]]
>>> print(np.median(a))
3.0
>>> a=np.array([[2,1],[6,5],[3,5]])    #    1  2  3  5  5  6
>>> print(a)                 # here middle elements are 3,5 and whose
(3+5)/2= 4.0
[[2 1]
 [6 5]
 [3 5]]
>>> print(np.median(a))
4.0
>>> a=np.array([[2,1,3],[6,5,4],[3,5,2]])
>>> print(np.median(a))
3.0
>>> print(a)
[[2 1 3]
 [6 5 4]
 [3 5 2]]
>>> print(np.median(a,axis=0))
[3. 5. 3.]
>>> print(np.median(a,axis=1))
[2. 5. 3.]
======================================================================
```

e) var():

The formula variance= square(xi-mean)/ total number of elements
                       here xi represents each element of ndarray
object.
            (or)
abs(xi-mean)^2 / total no of elements

Examples:

```
>>> a=np.array([[2,1],[4,3],[3,5]])
>>> print(a)
[[2 1]
 [4 3]
 [3 5]]
>>> print("mean=",np.mean(a))
mean= 3.0
>>> print("var=",np.var(a))----------------var= 1.6666666666666667
>>> print("col var=",np.var(a,axis=0))-----------col var= [0.66666667
2.66666667]
>>> print("row var=",np.var(a,axis=1))--------row var= [0.25 0.25 1.  ]
================================================================
```

f) std () :

The formula for std= sqrt(var)

Examples:

```
>>> a=np.array([[2,1],[4,3],[3,5]])
>>> print("mean=",np.mean(a))
mean= 3.0
>>> print("var=",np.var(a))
var= 1.6666666666666667
>>> print("std=",np.std(a))
std= 1.2909944487358056
>>> print("std=",np.std(a,axis=0))
std= [0.81649658 1.63299316]
>>> print("std=",np.std(a,axis=1))
std= [0.5 0.5 1. ]
```

==
Numpy--selecting the elements based on condition
(OR)
Creating Filter Directly From Array
==

=>To select any element from ndarray object, we the two approaches. They are

----
Approach-1:

=>Prepare Boolean Array (It contains True or False. True represents Condition
satisfied and False represents Condition not satisfied]

Syntax:- varname=ndarrayobject with condition

varname is called boolean array.

=>Pass the Boolean Array to the ndarray object. so that we can get those elements from ndarray which satisfies with the entry True(or) we can get those elements from ndarray corresponding True entries of Boolean array.

Syntax: ndarray[Boolean Array]

Approach-2:

=>In this approach, we directly pass Boolean array values to the ndarray for getting required elements based on condition.

Syntax: ndarray[ndarrayobject with condition]

Examples:

Q1) Select the Possitive Elements from ndarray

```
>>> import numpy as np
>>> l=[10,21,-34,23,-45,30,-40]
>>> print(l)--------------[10, 21, -34, 23, -45, 30, -40]
>>> a=np.array(l)
>>> a------------array([ 10,  21, -34,  23, -45,  30, -40])
>>> b=a>0    # Boolean Array
>>> print(b)----[ True  True False  True False  True False]
>>> a[b]-------array([10, 21, 23, 30])
```

```
===================OR=========================
>>> a[a>0]-----------array([10, 21, 23, 30])
---------------------------------------------------------------------
Q2) Select the Negative Elements from ndarray
      >>> l=[10,21,-34,23,-45,30,-40]
      >>> a=np.array(l)
      >>> a----------  array([ 10,  21, -34,  23, -45,  30, -40])
      >>> b=a<0  # Boolean Array
      >>> b----  array([False, False,  True, False,  True, False,  True])
      >>> a[b]-------  array([-34, -45, -40])
            ================OR=============
      >>> a[a<0]--------------    array([-34, -45, -40])
----------------------------------------------
```

```
            ==========================================
                 Numpy Searching Arrays
            ==========================================
```

=>We can search an array for a certain value, and return the indexes that get a match otherwise we get empty array.
=>To search an array, use the where() function.
=>Syntax: varname=numpy.where(Ndarray object with condition)

Exmaples:

```
Find the indexes where the value is 4:
import numpy as np
arr = np.array([1, 2, 3, 4, 5, 4, 4])
x = np.where(arr == 4)
print(x)  #  (array([3, 5, 6]),)
```

=>The example above will return a tuple: (array([3, 5, 6],)
=>Which means that the value 4 is present at index 3, 5, and 6.

```
=>Find the indexes where the values are even:
import numpy as np
arr = np.array([1, 2, 3, 4, 5, 6, 7, 8])
x = np.where(arr%2 == 0)
print(x)  # (array([1, 3, 5, 7]),)
-------------------------------------------------------------------------------------------
=>Find the indexes where the values are odd:
import numpy as np
arr = np.array([1, 2, 3, 4, 5, 6, 7, 8])
x = np.where(arr%2 == 1)
print(x)   # (array([0, 2, 4, 6]),)
-----------------------------------------------------------------------
```

```
#program eye
#eyeex1.py
import numpy as np
n=int(input("Enter Rows of matrix:"))
m=int(input("Enter Columns of matrix:"))
a=np.eye(n,m,dtype="int")
print(a)

#program matrix multiplication
#matrixmul.py
```

```
import numpy as np
l1=[10,20,30,40]
l2=[1,2,3,4]
#convert into ndarray
a=np.array(l1)
b=np.array(l2)
mat1=a.reshape(2,2)
mat2=b.reshape(2,2)

mat3=mat1*mat2
mat4=np.dot(mat1,mat2)
print("First Matrix:")
for row in mat1:
      print("\t{}".format(row))
print("Second Matrix:")
for row in mat2:
      print("\t{}".format(row))
print("Element Matrix Multiplication:")
for row in mat3:
      print("\t{}".format(row))
print("Orginal Matrix Multiplication:")
for row in mat4:
      print("\t{}".format(row))

#Program  for obtainin Pos and Neg Values
#ndfilter.py
import numpy as np
print("Enter List of values:")
lst=[int(val) for val in input().split()]
a=np.array(lst)
print("Given Elements")
print(a)
print("-----------------------------")
print("Possitive Elements={}".format(a[a>0]))
print("-----------------------------")
print("Negative Elements={}".format(a[a<0]))


#Program  for obtainin Even  and Odd Values
#ndfilter1.py
import numpy as np
print("Enter List of values:")
lst=[int(val) for val in input().split()]
a=np.array(lst)
print("Given Elements")
print(a)
print("-----------------------------")
print("Even  Elements={}".format(a[a%2==0]))
print("-----------------------------")
print("Odd Elements={}".format(a[a%2!=0]))

#performance.py
import numpy as np
import sys
l1=[10,20,30,40,50]
print("Type of l1=",type(l1))
a=np.array(l1)
print("Type of a=",type(a))
```

```
print("-------------------------------------")
print("Memory Size of l1=", sys.getsizeof(l1))
print("Memory Size of a=", sys.getsizeof(a))

#performance1.py
import numpy as np
import sys
l1=[10,20,30,40,50,56,78,89,34,56,78,99,123,45,67,89,34,56,234,567,45,234
,56,78,10,20,30,40,50,56,78,89,34,56,78,99,123,45,67,89,34,56,234,567]
print("Type of l1=",type(l1))
a=np.array(l1)
print("Type of a=",type(a))
print("-------------------------------------")
print("Memory Size of l1=", sys.getsizeof(l1))
print("Memory Size of a=", sys.getsizeof(a))

#performance2.py
import numpy as np
import sys
l1=[10,20,30,40,50,56,78,89,34,56,78,99,123,45,67,89,34,56,234,567,45,234
,56,78,10,20,30,40,50,56,78,89,34,56,78,99,123,45,67,89,34,56,234,567,"Py
thon"]
print("Type of l1=",type(l1))
a=np.array(l1,dtype="object")
print("Type of a=",type(a))
print("-------------------------------------")
print("Memory Size of l1=", sys.getsizeof(l1))
print("Memory Size of a=", sys.getsizeof(a))
```

===

Numpy---Basic Indexing

===

==>If we want to access Single element of 1D,2D and N-D arrays we must use the concept of Basic Indexing.

=>Accessing Single Element 1D-Array :

=>Syntax:- ndarrayname [Index]

=>Here 'index' can be either either +ve or -ve indexing

Examples:

```
>>> a=np.array([10,20,30,40,50,60])
>>> a
array([10, 20, 30, 40, 50, 60])
>>> a[0]
10
>>> a[3]
40
```

=>Accessing single Element of 2D :

=>Syntax:- ndarrayobj[row index,column index]

Examples:-

```
>>>import numpy as np
>>> a=np.array([10,20,30,40,50,60])
>>> b=a.reshape(2,3)
>>> b
array([[10, 20, 30],
       [40, 50, 60]])
>>> b[0,0]
10
>>> b[0,1]
20
>>> b[1,2]
60
```

==
=>Accessing single Element of 3D :

Syntax:-   ndarrayobj[Index of matrix , row index , column index]

Examples:

```
>>> a=np.array([10,20,30,40,50,60,70,80])
>>> b=a.reshape(2,2,2)
>>> b
array([[[10, 20],
        [30, 40]],

       [[50, 60],
        [70, 80]]])

>>> b[0,0,0]-----------10
>>> b[-1,0,0]---------50
>>> b[-2,1,1]---------40
```

==
Numpy---Advanced Indexing
==

==>If we want to access multiple elements, which are not in order (arbitrary elements) of 1D,2D and N-D arrays we must use the concept of Advanced Indexing.
=>If we want access the elements based on some condition then we can't use basic indexing and Basic Slicing Operations. To fullfill such type of requirements we must use advanced Indexing.

=>Accessing Multiple Arbitrary Elements ---1D :

=>Syntax:-   ndarrayname [x]

=>Here 'x' can be either ndarray or list which represents required indexes of arbitrary elements.

Examples:

```
>>> lst=[10,20,30,40,50,60,70,80,90]
>>> a=np.array(lst)
>>> print(a)----------------[10 20 30 40 50 60 70 80 90]
#access 10   30  and 80 elements
# here indexes of 10 30 and 80  are  0 2 7
>>> indexes=np.array([0,2,7]) # here [0,2,7] are indexes of 10  30 and 80
>>> print(indexes)---------[0 2 7]
>>> print(a[indexes])--------------[10 30 80]
      (OR)
>>> ind=[0,2,7]  # prepare the list of indexes of arbitray
elements(10,30,80) of ndarray and pass to ndarray
>>> print(a[ind]) -----------[10 30 80]
```

Examples:

```
Q1-->Access  20  30 80 10 10 30
>>> lst=[10,20,30,40,50,60,70,80,90]
>>> a=np.array(lst)
>>> print(a)----------------[10 20 30 40 50 60 70 80 90]
>>> ind=[1,2,7,0,0,2] # [1,2,7,0,0,2] are the indexes of 20 30 80 10 10
30
>>> print(a[ind])----------------[20 30 80 10 10 30]
```

=>Accessing Multiple Arbitrary Elements ---2D :

=>Syntax:- ndarrayobj[[row indexes],[column indexes]]

Examples:-

```
>>>import numpy as np
>>>mat=np.array([ [1,2,3,4],[5,6,7,8],[9,10,11,12],[13,14,15,16] ] )
>>> print(mat)
[[ 1  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]
 [13 14 15 16]]
```

Q1) Access the principle diagnal elements 1 6 11 16

```
Ans:-      mat[ [0,1,2,3],[0,1,2,3] ]
=>When the above statement is executed, The PVM takes internally as
             mat[ (0,0), (1,1), (2,2),(3,3) ]--------  1  6   11  16

>>> mat[ [0,1,2,3],[0,1,2,3] ]-----------array([ 1,  6, 11, 16])
```

Q2) Access the elements 6 14

```
Ans:      mat[[1,3],[1,1]]
=>When the above statement is executed, The PVM takes internally as
        mat[ (1,1),(3,1) ]

>>> mat[[1,3],[1,1]]-----------array([ 6, 14])
```

==

=>Accessing Multiple Arbitrary Elements ---3D :

```
Syntax:-       ndarray[ [Indexes of 2Dmatrix],[row indexes],[column
indexes]  ]
-------------

----------------
Examples:
---------------
>>>import numpy as np
>>>l1=[ [ [1,2,3,4],[5,6,7,8],[9,10,11,12] ],[
[13,14,15,16],[17,18,19,20],[21,22,23,24] ]  ]
>>>mat3d=np.array(l1)
>>>print(mat3d)
>>> print(mat3d)
[[[ 1  2  3  4]
  [ 5  6  7  8]
  [ 9 10 11 12]]

 [[13 14 15 16]
  [17 18 19 20]
  [21 22 23 24]]]
>>> mat3d.ndim
3
>>> mat3d.shape
(2, 3, 4)
----------------------------------------
Q1) Access the elements 1  14  24
Ans:-    mat3d[ [0,1,1], [0,0,2], [0,1,3]  ]

When the above statement is executed, Internally PVM takes as follows.
=>mat3d[ (0,0,0),(1,0,1),(1,2,3) ]-Gives-->1  14  24


Q1) Access the elements   10  16
>>> mat3d[[-2,-1],[-1,-3],[-3,-1]]----------array([10, 16])


      ================================================
      Numpy---Indexing and Slicing Operations of 1D,2D and 3D array
      ================================================
------------------------------------
1D Arrays Slicing:
------------------------------------
Syntax:-   1dndrrayobj[begin:end:step]
-----------------------
Examples:
-----------------------
>>> a=np.array([10,20,30,40,50,60,70])
>>> a------------array([10, 20, 30, 40, 50, 60, 70])
>>> a[::-1]-----------array([70, 60, 50, 40, 30, 20, 10])
>>> a[::]-----------array([10, 20, 30, 40, 50, 60, 70])
------------------------------------
2D Arrays Slicing:
------------------------------------
Syntax:- ndrrayobj[i,j]
      here 'i' represents  Row Index
      here 'j' represents Column Index

Syntax:-       2dndrrayobj[slice1, slice2]
```

```
Syntax:-       2dndrrayobj[begin:end:step, begin:end:step]

-------------------------------------------------------------------
Examples:
-------------------------------------------------------------------
>>> a=np.array([[10,20,30],[40,50,60]])
>>> a
array([[10, 20, 30],
       [40, 50, 60]])
>>> a[0,0]
10
>>> a[0:,0:1]
array([[10],
       [40]])
>>> a[0:,1:2]
array([[20],
       [50]])
>>> a[1:,:]
array([[40, 50, 60]])
==============================================================
3D Arrays Slicing
----------------------------
Syntax:-       3dndrrayobj[i,j,k]

      here 'i' represents Which 2D matrix ( Matrix Number-->0 1 2 3 4
5...... )
      here 'j' represents which Rows in that 2D matrix
      here 'k' represents which Columns in that 2D matrix
                        (OR)
Syntax:-       3dndrrayobj[slice1, slice2, slice3 ]
                        (OR)
Syntax:-       3dndrrayobj[begin:end:step, begin:end:step, begin:end:step
]
----------------------
Examples:
----------------------
>>> lst=[ [ [1,2,3],[4,5,6],[7,8,9] ],[ [13,14,15],[16,17,18],[19,20,21]
] ]
>>> print(lst)
[[[1, 2, 3], [4, 5, 6], [7, 8, 9]], [[13, 14, 15], [16, 17, 18], [19, 20,
21]]]
>>> arr2=np.array(lst)
>>> print(arr2)
[[[ 1  2  3]
  [ 4  5  6]
  [ 7  8  9]]

 [[13 14 15]
  [16 17 18]
  [19 20 21]]]
>>> arr2.ndim
3
>>> arr2.shape
(2, 3, 3)
>>> arr2[:,:,0:1]
array([[[ 1],
        [ 4],
```

```
        [ 7]],

       [[13],
        [16],
        [19]]])
>>> arr2[:,:,:1]
array([[[ 1],
        [ 4],
        [ 7]],

       [[13],
        [16],
        [19]]])
>>> arr2[: , 0:2, 1:3]
array([[[ 2,  3],
        [ 5,  6]],

       [[14, 15],
        [17, 18]]])
>>> arr2[: , :2, 1:]
array([[[ 2,  3],
        [ 5,  6]],

       [[14, 15],
        [17, 18]]])
```


```
#oddeventhreadex.py
import threading,time
def   even(n):
      ctname=threading.current_thread().name
      for i in range(2,n+1,2):
            print("Val Generated by {}={}".format(ctname,i))
            time.sleep(1)

def  odd(n):
      ctname=threading.current_thread().name
      for i in range(1,n+1,2):
            print("Val Generated by {}={}".format(ctname,i))
            time.sleep(1)

#main program
n=int(input("Enter a Number:"))
t1=threading.Thread(target=even , args=(n,))  # creating child thread
t2=threading.Thread(target=odd ,  args=(n,)) # creating child thread
#dispatch the threads
t2.start()
t1.start()
print("Number of active threads=",threading.active_count()) # 3
t1.join()
t2.join()
print("Number of active threads=",threading.active_count())# 1

#TrainsResr.py
import threading
class Train:
```

```
      L=threading.Lock()
      def __init__(self,n):
            self.seats=n

      def   reservation(self,nos):
            self.L.acquire()
            if(nos>self.seats):
                  print("{} unable get {}
seats:".format(threading.current_thread().name, nos))
            else:
                  self.seats=self.seats-nos
                  print("{} Reserved {}
seats:".format(threading.current_thread().name, nos))
            self.L.release()

#main program
t=Train(int(input("Enter Number Seats:")))
t1=threading.Thread(target=t.reservation, args=(100,))
t2=threading.Thread(target=t.reservation, args=(15,))
t3=threading.Thread(target=t.reservation, args=(23,))
t4=threading.Thread(target=t.reservation, args=(5,))
#dispatch the threads
t1.start()
t2.start()
t3.start()
t4.start()
```

==

Pandas

==

Introduction to Pandas:

=>Pandas is an open source Python Library / Module providing high performance and data
manipulation and Analysis Tool.
=>The word PANDAs derived from PANel DAta
=>The pandas concept developed by WES MCKinney in the year 2008.
=>The Traditional Python Programming does not contain any Module for Data Analysis and
Now Python Programming uses Pandas as an data anaysis tool.
=>Python Pandas can be used in wide range of fields like Financial Services, Statistics , retail
maketing sectors..etc as data analysis tool
=>pandas module developed in C and Python Languages.

Instalation of Pandas:

=>The standard python software / Distribution (CPYTHON) does not contain any module for data analysis and now we are using third party module called PANDAS and whose module name is pandas
=>Programatically to use pandas as part of our python program, we must install pandas module by using pip tool.

Syntax:-   pip install   module name

Example:-   pip install   pandas

-

Key Features of Pandas:-----> Series DataFrame
----
1) Fast and Efficient Data Frame with default and costomized indexing
2) Tools for loading the data in in-memory data objects(objects of Series,   DataFrame)
3) We can access the data from pandas by using Labeled Based Slicing and indexing.
4) Columns from in-memory data objects(objects of Series, DataFrame) can be deleted and inserted
----

===
Data Structures used in Pandas
===

=>In Pandas programming, we can store the data in 2 types of Data structures. They are.

a) Series
b) DataFrame

====================================
Series
====================================

=>It is a One-Dimensional Labelled Array Capable of Storing / Holding Homogeneous data of any type (Integer, String, float,.........Python objects etc).
=>The Axis Labels are collectively called Index.
=>Pandas Series is nothing but a column value in excel sheet.
=>Pandas Series Values are Mutable.
=>Pandas Series contains Homogeneous Data (Internally even we store different types values , They are treated as object type)

Creating an Series

=>A Series object can be created by using the folowing Syntax:
Syntax:-

----

varname=pandas.Series(object, index, dtype)

Explanation:-

=>Here varname is an object of <class, pandas.core.series.Series >
=>pandas is module name
=>Series() is pre-defined Function in pandas module and it is used for creating an object of Series class.
=>'object' can either list,ndarray,dict .....etc
=>'index' represnets the position of values present Series object. The default value of Index starts from 0 to n-1, Here n represents number of values in Series object. Programatically we can give our own Index Values.
=>'dtype' represents data type (Ex:- int32, ,int64, float32, float64...etc)

```
Examples:-    Create a series for 10 20  30 40 50 60

>>> import pandas as pd
>>> import numpy as np
>>> lst=[10,20,30,40,50,60]
>>> s=pd.Series(lst)
>>> print(s,type(s))
                       0    10
                       1    20
                       2    30
                       3    40
                       4    50
                       5    60
dtype: int64 <class 'pandas.core.series.Series'>
----------------------------
>>> lst=[10,20,30,40,50,60]
>>> s=pd.Series(lst,dtype=float)
>>> print(s,type(s))
           0    10.0
           1    20.0
           2    30.0
           3    40.0
           4    50.0
           5    60.0
dtype: float64 <class 'pandas.core.series.Series'>
-------------------------------------------------------------------
>>> lst=["Rossum","Gosling","Travis","MCKinney"]
>>> a=np.array(lst)
>>> a ---- ---array(['Rossum', 'Gosling', 'Travis', 'MCKinney'],
dtype='<U8')
>>> print(a, type(a))--['Rossum' 'Gosling' 'Travis' 'MCKinney'] <class
'numpy.ndarray'>
>>> s=pd.Series(a)
>>> print(s,type(s))
           0      Rossum
           1     Gosling
           2      Travis
           3    MCKinney
dtype: object <class 'pandas.core.series.Series'>
-----------------------------------------------------------------------
>>>lst=[10,"Rossum",34.56,"Author"]
>>> s=pd.Series(lst)
>>> print(s,type(s))
           0        10
           1    Rossum
           2     34.56
           3    Author
dtype: object <class 'pandas.core.series.Series'>
--------------------------------------------------------------------------
-------------------
      Creating an Series object with Programmer-defined Index
--------------------------------------------------------------------------
-------------------
>>> lst=[10,"Rossum",34.56,"Author"]
>>> print(lst)--------[10, 'Rossum', 34.56, 'Author']
>>> s=pd.Series(lst,index=["Stno","Name","Marks","Desg"])
>>> print(s)
           Stno         10
```

```
           Name     Rossum
           Marks     34.56
           Desg     Author
           dtype: object
>>> print(s["Stno"])-------10
>>> lst=["Rossum","Gosling","Travis","MCKinney"]
>>> s=pd.Series(lst,index=[100,200,300,400])
>>> print(s,type(s))
                 100      Rossum
                 200     Gosling
                 300      Travis
                 400    MCKinney
dtype: object <class 'pandas.core.series.Series'>
```

Creating a Series object from dict

=>A dict object can be used for creating a series object
=>If we use dict object in Series() then keys can be taken as Indices (Or Indexes)
automatically and corresponding values of dict can be taken as data.

Examples:

```
>>> import pandas as pd
>>> d1={"sub1":"Python","sub2":"Java","sub3":"Data Science","sub4":"ML"}
>>> print(d1)--{'sub1': 'Python', 'sub2': 'Java', 'sub3': 'Data Science', 'sub4': 'ML'}
>>> s=pd.Series(d1)
>>> print(s)
           sub1          Python
           sub2            Java
           sub3    Data Science
           sub4              ML
           dtype: object
>>> d2={"RS":2.3,"JG":1.2,"MCK":4.5,"TOLI":2.4}
>>> print(d2)---{'RS': 2.3, 'JG': 1.2, 'MCK': 4.5, 'TOLI': 2.4}
>>> s=pd.Series(d2)
>>> print(s)
           RS      2.3
           JG      1.2
           MCK     4.5
           TOLI    2.4
           dtype: float64
```
==========================X==================

==

DataFrame in Pandas

==

=>A DataFrame is 2-Dimensional Data Structure to organize the data .
=>In Otherwords a DataFrame Organizes the data in the Tabular Format, which is
nothing but Collection of Rows and Columns.
=>The Columns of DataFrame can be Different Data Types or Same Type
=>The Size of DataFrame can be mutable.

```
               ================================================
                    Number of approaches to create DataFrame
               ================================================
=>To create an object of DataFrame, we use pre-defined DataFrame() which
is present in pandas Module and returns an object of DataFrame class.
=>We have 5 Ways to create an object of DataFrame. They are
           a) By using list / tuple
           b) By using dict
           c) By using Series
           d) By using ndarray of numpy
           e) By using CSV File (Comma Separated Values)
------------------------------------------------------------------------
=>Syntax for creating an object of DataFrame in pandas:
-------------------------------------------------------------------------
--
      varname=pandas.DataFrame(object,index,columns,dtype)
----------------
Explanation:
----------------
=>'varname' is an object of <class,'pandas.core.dataframe.DataFrame'>
=>'pandas.DataFrame()' is a pre-defined function present in pandas module
and it is  used to create an object of DataFrame for storing Data sets.
=>'object' represents list (or) tuple (or) dict (or) Series (or) ndarray
(or) CSV file
=>'index' represents Row index and whose default indexing starts from
0,1,...n-1
     where 'n' represents number of values in DataFrame object.
=>'columns' represents Column index whose default indexing starts from
0,1..n-1
    where n number of columns.
=>'dtype' represents data type of values of Column Value.
=======================================================
Creating an object DataFrame by Using list / tuple
-------------------------------------------------------------------------
----------
>>>import pandas as pd
>>>lst=[10,20,30,40]
>>>df=pd.DataFrame(lst)
>>>print(df)
                0
           0  10
           1  20
           2  30
           3  40
------------------------------------
lst=[[10,20,30,40],["RS","JS","MCK","TRV"]]
df=pd.DataFrame(lst)
print(df)
            0   1    2    3
      0  10  20   30   40
      1  RS  JS  MCK  TRV
--------------------------------------------
lst=[[10,'RS'],[20,'JG'],[30,'MCK'],[40,'TRA']]
df=pd.DataFrame(lst)
print(df)
          0     1
      0  10   RS
```

```
      1  20   JG
      2  30  MCK
      3  40  TRA
--------------------------------------------------
lst=[[10,'RS'],[20,'JG'],[30,'MCK'],[40,'TRA']]
df=pd.DataFrame(lst, index=[1,2,3,4],columns=['Rno','Name'])
print(df)

         Rno Name
      1   10   RS
      2   20   JG
      3   30  MCK
      4   40  TRA
-------------------------------------------
tpl=( ("Rossum",75), ("Gosling",85), ("Travis",65),
("Ritche",95),("MCKinney",60) )
df=pd.DataFrame(tpl, index=[1,2,3,4,5],columns=['Name','Age'])
print(df)
           Name    Age
      1    Rossum   75
      2   Gosling     85
      3    Travis      65
      4    Ritche      95
      5  MCKinney  60
------------------------------------------------------------------------
------------
Creating an object DataFrame by Using dict object
--------------------------------------------------------------------
=>When we create an object of DataFrame by using Dict , all the keys are
taken as Column Names and Values of Value are taken as Data.
-----------------
Examples:
-----------------
>>> import pandas as pd
>>>
dictdata={"Names":["Rossum","Gosling","Ritche","McKinney"],"Subjects":["P
ython","Java","C","Pandas"],"Ages":[65,80,85,55]  }
>>> df=pd.DataFrame(dictdata)
>>> print(df)
                 Names   Subjects    Ages
            0    Rossum   Python   65
            1   Gosling     Java         80
            2    Ritche        C         85
            3  McKinney   Pandas   55
>>> df=pd.DataFrame(dictdata,index=[1,2,3,4])
>>> print(df)
                 Names    Subjects      Ages
            1    Rossum   Python          65
            2   Gosling     Java             80
            3    Ritche        C                85
            4  McKinney   Pandas        55
----------------------------------------------------------------------
Creating an object DataFrame by Using Series object
----------------------------------------------------------------------
>>> import pandas as pd
>>> sdata=pd.Series([10,20,30,40])
>>> df=pd.DataFrame(sdata)
>>> print(df)
```

```
          0
     0  10
     1  20
     2  30
     3  40
>>> sdata=pd.Series({"IntMarks":[10,20,30,40],"ExtMarks":[80,75,65,50]})
>>> print(sdata)
IntMarks    [10, 20, 30, 40]
ExtMarks    [80, 75, 65, 50]
dtype: object

>>> df=pd.DataFrame(sdata)
>>> print(df)
                        0
      IntMarks  [10, 20, 30, 40]
      ExtMarks  [80, 75, 65, 50]
>>> ddata={"IntMarks":[10,20,30,40],"ExtMarks":[80,75,65,50]}
>>> df=pd.DataFrame(ddata)
>>> print(df)
   IntMarks  ExtMarks
0        10        80
1        20        75
2        30        65
3        40        50
--------------------------------------------------------------------------
----------------------
Creating an object DataFrame by Using ndarray object
-----------------------------------------------------------------------
>>> import numpy as np
>>> l1=[[10,60],[20,70],[40,50]]
>>> a=np.array(l1)
>>> df=pd.DataFrame(a)
>>> print(df)
                 0   1
           0  10  60
           1  20  70
           2  40  50
>>> df=pd.DataFrame(a,columns=["IntMarks","ExtMarks"])
>>> print(df)
              IntMarks  ExtMarks
           0        10        60
           1        20        70
           2        40        50
--------------------------------------------------------------------------
-----------------
e) By using CSV File(Comma Separated Values)
------------------------------------------------------------------
import pandas as pd1
df=pd1.read_csv("D:\KVR-JAVA\stud.csv")
print("type of df=",type(df)) #type of df= <class
'pandas.core.frame.DataFrame'>
print(df)
--------------------- OUTPUT--------------------
        stno     name        marks
      0    10   Rossum 45.67
      1    20  Gosling 55.55
      2    30   Ritche       66.66
      3    40   Travis       77.77
```

```
      4    50      KVR        11.11
```

Misc Operations on DataFrame

```
>>> data={"First":[10,20,30,40],"Second":[1.4,1.3,1.5,2.5]}
>>> print(data,type(data))
{'First': [10, 20, 30, 40], 'Second': [1.4, 1.3, 1.5, 2.5]} <class
'dict'>
>>> df=pd.DataFrame(data)
>>> print(df)
   First  Second
0     10     1.4
1     20     1.3
2     30     1.5
3     40     2.5
>>> df["Third"]=df["First"]+df["Second"]
>>> print(df)
   First  Second  Third
0     10     1.4   11.4
1     20     1.3   21.3
2     30     1.5   31.5
3     40     2.5   42.5
----------------------------------------------------
>>> df["Total"]=df["First"]+df["Third"]
>>> print(df)
   First  Second  Third  Total
0     10     1.4   11.4   21.4
1     20     1.3   21.3   41.3
2     30     1.5   31.5   61.5
3     40     2.5   42.5   82.5

>>> df.pop("Total")
0    21.4
1    41.3
2    61.5
3    82.5
Name: Total, dtype: float64
>>> print(df)
   First  Second  Third
0     10     1.4   11.4
1     20     1.3   21.3
2     30     1.5   31.5
3     40     2.5   42.5
```

==

Working with CSV Files with Pandas

==

=>CSV stands for Comma Separated Values
=>CSV file is one of the Simple file format used for storing Tabular data such as spread sheet or data base
=>CSV files stores Tabular data (Numbers and text) in plain text.
=>Each line of CSV is a data record. Each record contains contains collection
of values separated by comma

=>CSV files must be saved on some file name with an extension .csv (internally treated as excel sheet)
=>To deal with CSV file, we must import a pre-defined module called "csv"

Examples:

```
                 stud.csv
                     --------------------
                  stno,sname,marks
                 10,Rossum,34.56
                 20,Gosling,45.67
                 30,Ritche,56.78
                 40,Kinney,66.67
                 50,Oliphant,66.99
```

```
#noncsv.py
try:
	with open("E:\KVR-PYTHON-4PM\CSV\stud.csv") as fp:
		records=fp.readlines()
		for record in records:
			print(record,end="")
except FileNotFoundError:
	print("File does not exists")
```

```
#readcsv.py
import csv   #  in csv  module, we have reader()
try:
	with open("E:\KVR-PYTHON-4PM\CSV\stud.csv","r") as fp:
		print("="*50)
		csvreader=csv.reader(fp)
		for record in csvreader:
			for val in record:
				print("\t{}".format(val),end="")
			print()
		else:
			print("="*50)
except FileNotFoundError:
	print("File does not exists")
```

```
#pandascsv.py
import pandas as p
df=p.read_csv("E:\KVR-PYTHON-4PM\CSV\stud.csv")
print(df)
```

sno	sname	marks
10	RS	33.33
20	TR	55.55
30	DR	66.56
40	DJ	77.77
50	RT	66.66
60	DW	55.55
70	WE	77.11
80	RT	44.44

```
=======================================================
           Accesssing the Data of DataFrame
=======================================================
1) DataFrameobj.head(no.of rows)
2) DataFrameobj.tail(no.of rows)

3) DataFrameobj.describe()
4) DataFrameobj.shape
5) DataFrameobj[start:stop:step]

6) DataFrameobj["Col Name"]

7) DataFrameobj[ ["Col Name1","Col Name-2"...."Col Name-n"] ]

8) DataFrameobj[ ["Col Name1","Col Name-2"...."Col Name-n"]]
[start:stop:step]
9) DataFrameobj.iterrows()
===================================================
Understabding loc() ----- here start and stop index Included and
                                         Col Names can be used(but not
column numbers]
---------------------------------------------------------------------------
-------------
1) DataFrameobj.loc[row_number]
2) DataFrameobj.loc[row_number,[Col Name,.........] ]
3) DataFrameobj.loc[start:stop:step]
4) DataFrameobj.loc[start:stop:step,["Col Name"] ]
5) DataFrameobj.loc[start:stop:step,["Col Name1", Col Name-2......."] ]
6) DataFrameobj.loc[start:stop:step,"Col Name1" : Col Name-n"]
---------------------------------------------------------------------------
-----------------------------------

Understabding iloc() ----- here start index included and stop index
excluded and
                                         Col Numbers  must be used(but
not column names]
---------------------------------------------------------------------------
-------------
1) DataFrameobj.iloc[row_number]
2) DataFrameobj.iloc[row_number,Col Number.........]
3) DataFrameobj.iloc[row_number,[Col Number1,Col Number2............] ]
3) DataFrameobj.iloc[row start:row stop, Col Start: Col stop]
4) DataFrameobj.iloc[row start:row stop,Col Number ]
5) DataFrameobj.iloc[ [row number1, row number-2.....] ]
6) DataFrameobj.iloc[ row start: row stop , [Col Number1,Col
Number2............] ]
6) DataFrameobj.iloc[ : , [Col Number1,Col Number2............] ]
=======================================================================
           Adding Column Name to Data Frame
=======================================================================
1)  dataframeobj['new col name']=default value
2)  dataframeobj['new col name']=expression
=======================================================================
           Removing Column Name from Data Frame
=======================================================================
1)dataframe.drop(columns="col name")
2)dataframe.drop(columns="col name",inplace=True)
=======================================================================
```

```
            sorting the dataframe data
===========================================================================
1) dataframeobj.sort_values("colname")
2) dataframeobj.sort_values("colname",ascending=False)
3) dataframeobj.sort_values(["colname1","col name2",...col name-n] )
===========================================================================
            knowing duplicates in dataframe data
===========================================================================
1) dataframeobj.duplicated()---------------gives boolean result
===========================================================================
            Removing duplicates from dataframe data
===========================================================================
1) dataframeobj.drop_duplicates()
2) dataframeobj.drop_duplicates(inplace=True)
===========================================================================
            Data Filtering and Conditional Change
===========================================================================
1) dataframeobj.loc[ simple condition]

                  Ex:    df.loc[ df["maths"]>75 ]

2) dataframeobj.loc[ compund condition]
                  Ex:   df.loc[ (df["maths"]>60) & (df["maths]<85) ]
Ex: df.loc[ (df["percent"]>=60)  & (df["percent"]<=80),["grade"]]="First"
# cond updattion.

Special Case:
3) dataframeobj.loc[simple condition.str.contains(str)]
4) dataframeobj.loc[simple condition.str.startswith(str)]
5) dataframeobj.loc[simple condition.str.endswith(str)]
===========================================================================



htno  name  telugu     english    hindi maths science     social
100   Ramesh      50    60    66    98    66    55
101   Rajesh      45    67    34    67    66    78
102   Rossum      56    88    56    99    44    77
103   Raji  56    78    34    56    88    55
104   Kalyan      51    63    62    93    67    51
105   Karthik     48    62    39    68    65    88
106   Kambli      53    81    59    92    48    73
107   Praveen     46    88    74    86    78    45
108   Ganesh      53    62    76    88    76    35
109   Nags  55    77    44    77    86    58
106   Kambli      53    81    59    92    48    73
110   Biswa 66    48    86    95    48    47
111   Ritchi      66    68    64    76    98    75
100   Ramesh      50    60    66    98    66    55



                  ==================================================
                          Network Programming in Python
                 ==================================================
```

=>The purpose of network programming is that "To share the data between multiple Machine whcih are present in the network.
=>A Network is a collection of autonomous interconnected computers connected with server."
=>In network programming, we can write two types of programs. They are

1. Server Side Program.
2. Client Side Program.

Def.of Server Side Program:

=>A Server Side Program is one, which is accepting Client request, Process the client request and gives response back to the client.

Def.of Client Side Program:

=>A Client Side Program is one, which is sending a request to server and receives response from Server Side Program.

Def. of DNS (Domaining Naming Service):

=>The DNS is the name of the Physical Machine, where the Server Side Program Resides.
=>The default name of DNS is "localhost"

Def. of IP Address (Internet Protocal Address):

=>An IP Address is one of the four parts numerical address of a physical machine, where the server side program resides.
=>The default IP Address of every computer is 127.0.0.1 (loop back address)

Def. of Port Number

=>A Port Number is one of the numerical id, where the server side program is running.
==
Steps for Developing Server Side Program

Step-1: import socket module.
Step-2: Every Server Side Program must run at Certain DNS / IP Address(Residing) and port number(running)
Step-3: Every Server Side Program must be configured in such way that how many client(s) can make request at a time.
Step-4: Every Server Side Program ACCEPT the Client Side Program request.
Step-5: Every Server Side Program must READ Client Side Program request,PROCESS the client side program request (decode the request)
Step-6: Server Side Program must SEND the result to Client Side program (encode the result)

Note:- As long as Client Side Program makes a request, Server Side Program Performs step-(4),step-(5)
and Step-(6)
==
Steps for Developing Client Side Program
==
Step-1: import socket module
Step-2: Every Client Side program must get a connection from Server Side Program by passing DNS (or
IP Address) and Port Number.
Step-3: Every Client Side program must SEND a request(encode) to the server side program.
Step-4: Every Client Side program must RECEIVE the response (decode) from Server Side Program
----
Note:- If the Client Side Program want to make multiple Request and receives multiple responses then
Client Side Program must reapeat step-(3) and Step-(4)
====================================X======================================

===
Module Required for dealing with Network Programming
===

=>To deal with network programming, we use a pre-defined module called "socket" and it present python
itself. (No Need to install with pip).

=>The pre-defined Functions in socket module
----
1) socket()
2) bind()
3) listen()
4) accept()
5) recv() with decode()
6) send() with encode()
7) connect()

```
#program for Client side operations.
#Client1.py
import socket
s=socket.socket()
s.connect(("localhost",9999))
print("CSP get Connection from SSP")
s.send("Hello Server".encode())
sdata=s.recv(1024).decode()
print("Server Data =",sdata)
```

```
#program for server side operations.
#Server1.py
import socket
s=socket.socket()
s.bind(("localhost",9999))
s.listen(2)
print("\nSSP is Ready to accept any CSP request")
```

```
while(True):
      clientsock, clientaddr=s.accept()
      print("Client Socket object=",type(clientsock))
      print("Client Socket address {} and
type{}=".format(clientaddr,type(clientaddr)))
      print("---------------------------------------------")
      cdata=clientsock.recv(1024).decode()
      print("Client Data at Sever=",cdata)
      clientsock.send("Hello client".encode())

#client side program  accept the values from KBD , send to server and its
square.
#ClientSquare.py
import socket
irfan=socket.socket()
irfan.connect(("localhost",8888))
print("CSP get Connection from SSP")
#acccpe the value from KBD and send
n=input("Enter a number:")
irfan.send(n.encode())
#CSP recevies the result from SSP
sdata=irfan.recv(1024).decode()
print("result from server=",sdata)


#This Server Side Program accept client value and Square it and send
back.
#ServerSquare.py
import socket
s=socket.socket()
s.bind(("localhost",8888))
s.listen(2)
print("\nSSP is ready to accept any CSP request:")
while(True):
      cs,ca=s.accept()
      #receive client side data
      cdata=float(cs.recv(1024).decode())
      print("Client Data at Server=",cdata)
      #process client request
      res=cdata**2
      #send server response from client side program
      cs.send( str(res).encode())
```

===

String Handling in Python(part-2)

===

=>We know that a String is a collection / sequence of Characters enclosed within single / double Quotes (or) triple single / double Quotes.

=>String data is of type <class,'str'>

=>To do various opereations on String data, we have to use the following the functions.

----

1) capitalize():

=>This function is used for capitalizing the given str data

```
=>Syntax:     varname=strobj.capitalize()
-----------------
Examples:
-----------------
>>> s="python is an oop lang"
>>> print(s,type(s))---------python is an oop lang <class 'str'>
>>> cs=s.capitalize()
>>> print(cs,type(cs))----  Python is an oop lang <class 'str'>
>>> print(s,type(s))----     python is an oop lang <class 'str'>
---------------------------------------------------------------------------
--------
```

2) title():

```
=>This Function is used for getting all words First Characters as
capital.
=>Syntax:-       varname=strobj.title()
Examples:
----------------
>>> s="python is an oop lang"
>>> ts=s.title()
>>> print(ts,type(ts))--------Python Is An Oop Lang <class 'str'>
>>> print(s,type(s))-----python is an oop lang <class 'str'>
---------------------------------------------------------------------------
-----
```

3) find():

```
 =>This function is used for finding an index of the first occurance of
specified str data  in the given str data.
 =>If the data found then it returns Its  +ve index value
 =>If the data not found then it returns  -1

Syntax:-    varname=strobj.find(str data)
Examples:
-----------------
>>> s="python is an oop lang"
>>> print(s,type(s))
python is an oop lang <class 'str'>
>>> ind=s.find("python")
>>> print(ind)------0
>>> ind=s.find("n")
>>> print(ind)------5
>>> ind=s.find("k")
>>> print(ind)-------     -1
>>> ind=s.find("o")
>>> print(ind)-------4
Examples:
-----------------
      for let in s:
             ind=s.find(let)
           print(ind)
Examples:
-----------------------
#Indexex.py
line=input("Enter a line of text:")
print("Given Data={}".format(line))
for ch in line:
      print("\tCharacter: {}   Index={}".format(ch,line.find(ch)))
```


```
#indexex.pt
line=input("Enter line of text:")  # Python
i=0
for ch in line:
      print("Character :{}--->Index:{} and orginal Index={}".format( ch,
line.index(ch),i ))
      i=i+1
```

----

4) isalnum():

=>This Function returns True Provided str data contains "Alphabets with digits or only with digits or only with alphabets"

=>This Function returns False Provided str data is a combination of "Alphabets and numbers with any special Symbols"

```
Syntax:-  varname=strobj.isalnum()
                       (or)
           strobj.isalnum()
```

Examples:

```
>>> s="12345"
>>> b=s.isalnum()
>>> print(b)------------True
>>> s="python12345"
>>> s.isalnum()----------True
>>> s="python12345#"
>>> s.isalnum()---------False
>>> s="python  12345"
>>> s.isalnum()----------False
>>> s="Python is an oop lang"
>>> s.isalnum()-----------False
>>> s="python"
>>> s.isalnum()--------True
>>>s="-123"
>>> s.isalnum()-----------False
```

----

5) isalpha():

=>This Function returns True provided str data contains only Alphabets otherwise it returns False.

```
=>Syntax:-          varname=strobj.isalpha()
```

Examples:

```
>>> s="Python"
>>> b=s.isalpha()
>>> print(b)------------True
>>> s="1234"
>>> print(s.isalpha())--------False
>>> s="python1234"
>>> print(s.isalpha())-------False
>>> s="python_1234"
```

```
>>> print(s.isalpha())-------False
------------------------------------------------------------------------
-----
6) isdigit():
-------------------
=>This Function returns True provided str data contains only purly
digits(0-9) otherwise it returns False.
Syntax:-   varname=strobj.isdigit()
                   or
            strobj.isdigit()
Examples:
-----------------
>>> a="1234"
>>> print(a.isdigit())------------True
>>> a="pyth1234"
>>> print(a.isdigit())--------False
>>> a="python"
>>> print(a.isdigit())------False
>>> a="pyth#$123"
>>> print(a.isdigit())---------False
------------------------------------------------------------------------
-----
7) islower() :
--------------------
=>This Function returns True provided the str data is completely
available in lowercase otherwise it returns False.

Syntax:-   varname=strobj.islower()
                   or
            strobj.islower()
Examples:
-----------------
>>> s="python"
>>> print(s.islower())----------True
>>> s="Python"
>>> print(s.islower())---------False
>>> s="python is an oop lang"
>>> print(s.islower())----True
>>> s="python is An oop lang"
>>> print(s.islower())-------False
------------------------------------------------------------------------
-----
7) isupper() :
--------------------
=>This Function returns True provided the str data is completely
available in upper case otherwise it returns False.

Syntax:-   varname=strobj.isupper()
                   or
            strobj.isupper()

Examples:
------------------
>>> s="Python"
>>> print(s.isupper())----------False
>>> s="PYTHON"
>>> print(s.isupper())-------True
>>> s="python is an oop lang"
```

```
>>> print(s.isupper())---------False
>>> s="PYTHON IS AN OOP LANG"
>>> print(s.isupper())-------True
```
----

9) isspace()

=>This Function returns True provided str data contains purely space(s) otherwise it returns False.
=>Syntax:-   varname=strobj.issapce()
                            (or)
                       strobj.isspace()

Examples:

```
>>> s="Python is an oop"
>>> print(s.isspace())--------False
>>> s="  "
>>> print(s.isspace())--------True
>>> s="          "
>>> print(s.isspace())--------True
>>> s="123  345"
>>> print(s.isspace())---False
>>> s=""               # empty string
>>> s.isspace()-----------False
```
----

10) upper():

=>This Function is used for converting lower case data into upper case data.
Syntax:-   varname=strobj.upper()

11) lower():

=>This Function is used for converting upper case data into lower case data.
Syntax:-   varname=strobj.lower()

Examples:

```
>>> s="python is an oop lang"
>>> uc=s.upper()
>>> print(uc)-------PYTHON IS AN OOP LANG
>>> print(s)-------python is an oop lang
>>> print(uc)---- PYTHON IS AN OOP LANG
>>> lc=uc.lower()
>>> print(lc)-------- python is an oop lang
```
----

12) join():

=>This Function is used concatinating all the sequence of values which are available in the form str
Syntax:-   varname=strobj1.join(iterable obj)
=>Here iterable obj contains multiple values in the form of str
Examples-:

```
>>>tpl=('java', 'python', 'Data Science')
>>> print(tpl, type(tpl))--('java', 'python', 'Data Science') <class
'tuple'>
>>> s2=""
>>> s3=s2.join(tpl)
>>> print(s3)---->javapythonData Science
--------------------------
>>> lst=["Apple","Mango","Kiwi","Guava"]
>>> frs=""
>>> frs=frs.join(lst)
>>> print(frs)-------------------AppleMangoKiwiGuava
>>> lst=["Apple","Mango","Kiwi","Guava"]
>>> frs=" "
>>> frs=frs.join(lst)
>>> print(frs)--------------Apple Mango Kiwi Guava
--------------------------------------------------------------------------
-------------------------------
13) split():
--------------------
=>This function is used for splitting the given str data into different
tokens based spitting value. The default splitting value is space
=>This Function returns splitting values in the form of list.

Syntax:-   listobj=strobj.split()

             listobj=strobj.split("spliting value")
Examples:
----------------
>>> s="Python is an oop lang"
>>> s.split()---------  ['Python', 'is', 'an', 'oop', 'lang']
>>> s="9-11-2021"
>>> l=s.split("-")
>>> print(l)----------['9', '11', '2021']
>>> s="apple#kiwi#guava-banana"
>>> l=s.split("#")
>>> print(l)----------['apple', 'kiwi', 'guava-banana']
>>> l[2].split("-")--------['guava', 'banana']
=============================X====================================


                  =================================
                       generator in python
                 =================================
=>generator is one of the function
=>The generator function always contains yield keyword
=>If the function contains return statement then it is called Normal
Function
=>If the function contains yield keyword then it is called generator
=>Syntax:
           def   function_name(start,stop,step):
                   ----------------------------------------
                 ----------------------------------------
                 yield value
                 ----------------
=>The 'yield' key word is used for giving the value back to function call
from function defintion and continue the function execution until
condition becomes false.
```

=>The advantage of generators over functions concept is that it save lot of memory space in the case large sampling of data. In otherwords Functions gives all the result at once and it take more memory space where as generators gives one value at a time when programmer requested and takes minimized memory space.
========================X====================================

```
#genex1.py
def  kvrrange(l,u ):
      while(l<=u):
            yield l
            l=l+1


#main program
kr=kvrrange(10,21)
print("type of kr=",type(kr))
for i in kr:
      print(i)

#genex2.py
def  kvrrange(l,u,s ):
      while(l<=u):
            yield l
            l=l+s

#main program
kr=kvrrange(10,21,2)
print("type of kr=",type(kr))
for i in kr:
      print(i)

#genex3.py
import sys
def  kvrrange(l,u,s ):
      if(l>u):
            print("Invalid Input")
            sys.exit()
      else:
            while(l<=u):
                  yield l
                  l=l+s

#main program
lb=int(input("Enter Lower Bound Value:"))
ub=int(input("Enter upper Bound Value:"))
s=int(input("Enter Step Value:"))
kr=kvrrange(lb,ub,s)
print("="*50)
while(True):
      try:
            print(next(kr))
      except StopIteration:
            print("="*50)
            break


#fungenex.py
```

```
def  fun1():
      return "Hello"

def  fun2():
      s="Hello"
      i=0
      while(i<len(s)):
            yield s[i]
            i=i+1

#main program
print("type of fun1=",type(fun1))  # <class,"function">
obj1=fun1()
print("Content of obj1=",obj1)  # Hello
print("=================================")
obj=fun2()
print(next(obj))
print(next(obj))
```

==

Module Name required for developing Networking applications

==

=>The pre-defined module required for developing Networking Application is "socket".

=>"socket" module contains the following functions.

----

1)socket():

=>This function is used for creating an object socket.

=>An object of socket acts as bi-directional communication entity between Client and Server Side Applications.

      Syntax:-   varname=socket.socket( )

=>Here varname is an object <class,"socket">

Example: s=socket.socket()

----

2) bind()

=>This function is used making Server side program to run at certain machine (DNS) and certain port number .

=>Syntax:-   socketobj.bind((DNS,portno))

OR

=>Syntax:-   socketobj.bind((IP Address,portno))

Examples: s.bind("localhost",8888)

or

s.bind("127.0.0.1",8888)

----

3) listen()

=>This function is used for configuring the server side program in such a way that how many clients can communicate with server side program

=>Syntax:-   socketobj.listen(No. of Client side programs)

=>Examples: socketobj.listen(2)
----

4) accept():

=>This Function is used for accepting client program request and it returns the obejct of Client Side and its address.
=>Syntax: varname1,varname2=socketobj.accept()
=>Here varname1 represents an connection object from client (soket)
=>Var name2 represents address of Client Side program (socket address)
Examples:-   clientobj,clientaddr=socketobj.accept()
----

5) recv() with decode()

=>This function is used for receving the client side program request with decode() at server side progrm and also used at client side for receving Server Side Program response.
=>Syntax1:-   varname =clientsocketobj.recv(1024/2048/4096).decode()-----at client side
=>Syntax2:-   varname=clientobj.recv(1024/2048/4096).decode()---------at Server Side
----

6) send() with encode()

=>This function is used for sending client request data to Server side program and Server Side program send Response to client side program.
Syntax:-   Clientsocketobject.send(1024/2048/4096).encode()----At Client Side Program
Syntax:-   Clientobj.send(1024/2048/4096).encode()----At Server Side Program
----

7) connect():

=>This program is used for obtaining connection from Server Side Program at Client Side
    Program.
=>Syntax:-   socket.connect(("DNS",portno))
                           (OR)
=>Syntax:-   socket.connect(("IP Address",portno))

Examples: s.connect(("localhost",8888))
                           (OR)
                    s.connect(("127.0.0.1",8888))

======================================
Decorators in Python
======================================

=>Decorator is one of the Function which will provides Additional Processing capability to the normal Function value and returns the modified value.

Syntax:-

```
      def     functionname1( functionname ):
                def  innerfunctionname():
                      val=functionname()
                    --------------------------
                    #do the operationon ' val '
                    -------------------------
                    return val
             return   innerfunctionname
```

=>here functionname1 is called Decorator function
=>here Functionname as a formal parameter . Every decorator function must take normal function as parameter.

```
#nondecorator.py
def  getval():
      return (float(input("Enter a number:")))

def   square():
      n=getval()
      return n**2

def cube():
      n=square()
      return n**3
#main program
res=cube()
print("Result=",res)


#decoratorex1.py
def  square(kvr):
      def   operation():
            n=kvr()
            res=n**2
            return res
      return operation


def  getval():
      return 5

#main program
result=square(getval)
print("result=",result())


#decoratorex2.py
def  square(kvr):
      def   operation():
            n=kvr()
            res=n**2
            return res
```

```
	return operation

@square
def  getval():
	return (float(input("Enter a number:")))

#main program
result=getval()
print("result=",result)


#decoratorex3.py
def cube(hyd):
	def operation1():
		x=hyd()
		res=x**3
		return res
	return operation1

def  square(kvr):
	def   operation():
		n=kvr()
		res=n**2
		return res
	return operation

@cube
@square
def  getval():
	return (float(input("Enter a number:")))

#main program
result=getval()
print("result=",result)
```

==

Iterators in Python

==

=>An iterator is an object that contains countable number of values.
=>An Iterator is an object that can be Iterated with all values
Examples: lst=["apple","mango","Kiwi","Guava"]

```
for frt in lst:
	print(frt)
```

=>here lst is by default Iterable object
=>Programatically, to convert an object which contains multiple values(Iterable object) as iterator object, we use iter()

=>Syntax:-   itrobj=iter(object with multiple values)

=>To retrive the values from iterator object , we use next() and it generates an exception called StopIteration when no value present in iterator object.

Examples:

```
--------------------
#iterex1.py
lst=["apple","mango","Kiwi","Guava"]
for val in lst:
	print(val)
print("=========OR==============")
itrlst=iter(lst)
while(True):
	try:
		print(next(itrlst))
	except StopIteration:
		break
=========================X==========


#iteratorex1.py
lst=[10,20,30,40,50,60,70,80]  # Iterable objects
print(lst)
for x in lst:
	print(x)
print("==========================")
iterobj=iter(lst)  # here iterobj is an object of Iterator
while(True):
	try:
		print(next(iterobj))
	except StopIteration:
		break


#iteratorex2.py
s="Python"
for x in s:
	print(x)
print("==========================")
iterobj=iter(s)  # here iterobj is an object of Iterator
print(type(iterobj))
while(True):
	try:
		print(next(iterobj))
	except StopIteration:
		break
```