
History of Python

- =>The Python programming Language foundation stone has laid in the year 1980
 - =>The Python programming Language development started in the year 1989
 - =>The Python programming Language officially released in the year 1991 Feb
 - =>The Python programming Language Developed by "Guido Van Rossum"
 - =>The Python programming Language developed CWI in Nether lands.
 - =>The Python programming Language maintained by non-commercial Organization Python Software Foundation (PSF).
-

Versions of Python

=>Python Programming Contains 3 Types of Versions.

- a) Python 1.x
- b) Python 2.x , here x represents 0 1 2 3 4 5 6 7
- c) Python 3.x here x represents 0 1 2 3 4 5 6 7 8 9 10 11 12

=>Python 3.x does not contain backward Compatibility with Python2.x

Downloading Process of Python

We can download the python software from www.python.org

Python Programming Inspired From

- => Functional Programming From C
- => Object Oriented Programming From CPP
- =>Modular programming from Modulo3
- =>Scripting Programming from PERL

Real Time Applications developed by using PYTHON

=Python Programming can be in 23 different area / applications.

- 1) Web Application Development (Django,flask,pyramid...etc)
- 2) Gaming Application development
- 3) Artificial Intelligence (Machine learning and Deep Learning Applications)
- 4) Desktop GUI Applications.
- 5) Image Processing Based Applications
- 6) Text processing Based Applications.
- 7) Business Based Applications
- 8) Education Sector
- 9) Audio and Video based Applications
- 10)Web Scrapping / Harvesting Application
- 11) Data Visualization.
- 12) Scientific and Numerical Applications
- 13) Software development
- 14) Operating System development
- 15) CAD and CAM based Applications

-
- 16) Embedded System
 - 17) Console Based Applications.
 - 18) Computer Vision.
 - 19) Language development .
 - 20) Automation in testing
 - 21) Data Analysis and Data Analytics
 - 22) Development of IOT
 - 23) Enterprise Applications
-

Features of Python

=>Features of a language are nothing but Services / Facilities Provided by Language Developers and those features used by real time programmers for developing real time applications.

=>Python Programming Provides 11 Features. They are

- 1. Simple
 - 2. Freeware and Open Source
 - 3. Platform Independent
 - 4. Dynamically Typed
 - 5. Interpreted
 - 6. High Level
 - 7. Extensible
 - 8. Embedded
 - 9. Both Procedural (Functional) and object oriented Prog lang.
 - 10. Robust (Strong)
 - 11. Extensive Supports for Third Party APIS (Numpy , Pandas, Scipy,Scikit, matpololib...etc)
-

1. Simple

=>Python is one of the SIMPLE Programming Language, bcoz of 3 important Tech Features.

1) Python Programming Provides "RICH SET OF APIs". So that Python Programmer can re-use the pre-defined code and develops real time applications easily without writing our own code.

Def of APIs(Application Programming Interface):

=>An API is a collection of Modules.

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

2) Python Programming Provides an In-built facility called "Garbage Collector ", which is running behind of every python program for collecting Un-Used Memory space and improves the performance of Python Based Applications.

Def of Garbage Collector:

=>Garbage Collector is one of Back ground python Program, which running behind of every regular python program and whose purpose is that to collect Un-Used Memory space and improves the performance of Python Based Applications.

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

Freeware and Open Source

=>Freeware:

=>Since Python software is Freely Downloadable and hence Python is one of the Freeware.

=>Open Source:

=>The standard python name released by PSF to the real time is called "CPYTHON".

=>Some Companies came forward and customized CPYTHON for their in-house tools development and these customized development of CPYTHON are called "Python Distributions".

=>Some of the Python Distributions are.

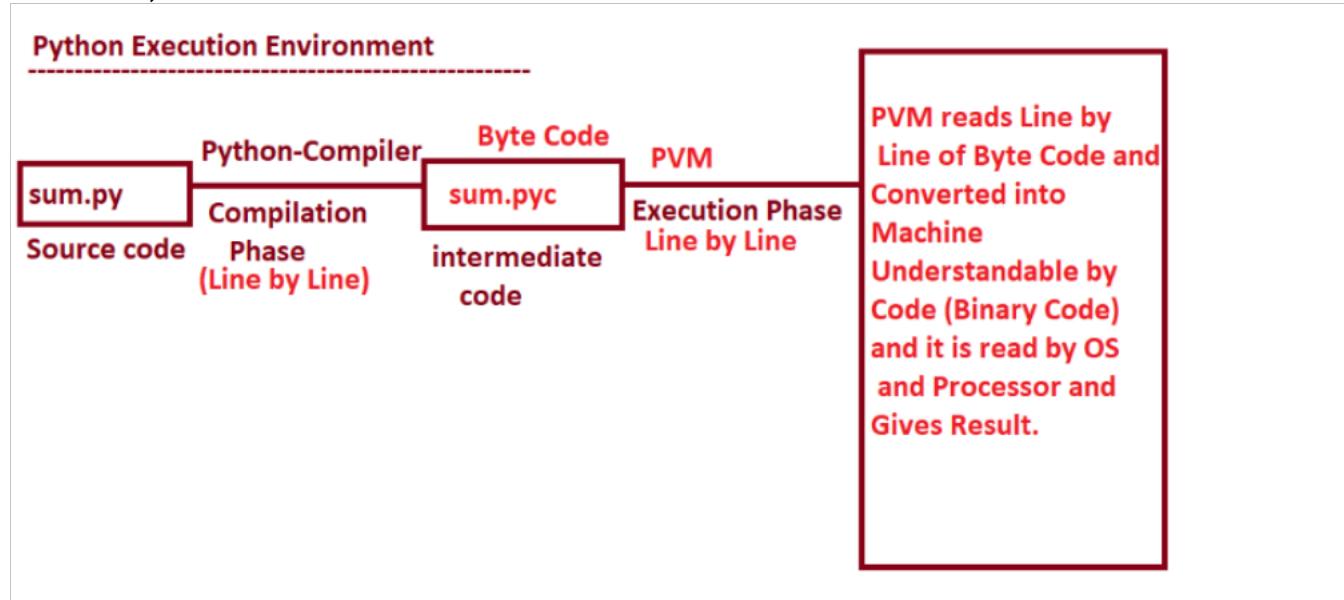
- 1) JPython (or) Jython---->It is used for running Java Based Applications.
- 2) Iron Python (OR) Ipython--->It is used for running C#.net Applications.
- 3) Ruby Python ---->It is used to run RUBY Based Applications
- 4) Micro Python---->It is used to develop Micro Controller Applications.
- 5) Anakonda Python-->It used develop Big Data / Hadoop Based

Applications.

- 6) Stackless Python--->Concurrency
 - 7) PyPy--->It provides JIT(Just -in Time) compiler
-etc
-

Interpreted

=>When we execute any python Program, Internally the phases takes place. They are
a) Compilation Phase
b) Execution Phase



a) Compilation Phase:

=>The python Compiler, reads the Source Code (Example Sum.py) Line by Line and Converted into Intermediate Code of Python called "Byte Code"
=>Since Python Compiler converting the source code into Byte Code Line by Line and hence it is Interpreted.

Examples:--- Sum.py---->Python Compiler(Line by Line)---->Sum.pyc (Byte Code)

b) Execution Phase:

=>The PVM (Python Virtual Machine) reads Line by Line of Byte Code (.pyc) and converted into Machine Understandable Code and It is read by OS and Processor and Gives Final Result of the Program
=>Since this execution performed by PVM Line by Line and hence It is Interpreted Programming.

Platform Independent

=>If any programming language based applications / Programs runs on every OS then it is called Platform Independent.
=>Python is one of the Platform Independent Language because "In Python Programming Execution Environment all Values are stored in the form OBJECTS".
=>All the Objects can store un-limited number of values and They will not depends on any OS. (Independent from OS)

Examples:

=>C,CPP are comes under Platform Dependent because C, CPP lang data types are keywords and their memory space changing from one OS to another OS.

=>Java comes under Platform Independent because Java lang Pre-Data Types are keywords and their memory space remains same on all TTypes OSes but they can store single value and unable to store multiple values. where as Java Programming also contains Classes and Objects and in Objects we can store Multiple Values with Limited only.

=>Where Python Programming Language is one of the Platform Independent because python programming stores all the values in the form OBJECTS with UNLIMITED size.

NOTE:- in python values are stored in the form objects.

Dynamically Typed

=>In context of Programming Languages, we have two types of Programming languages.

- 1) Static Typed Programming Languages
- 2) Dynamically Typed Programming Languages

1) Static Typed Programming Languages:

=>In Static Typed Programming Languages, It is mandatory to specify the data type otherwise we get Compile Time Error

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

Instructions: int a,b,c; // Variable Declaration--Mandatory

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

a=12.34-----Error--incompatible types

2) Dynamically Typed Programming Languages:

=>In Dynamically Typed Programming Languages, It is not necessary to specify the data type for the variables bcoz Python Programming Environment will automatically assign the data type depends on type value programmer assigns.

Example: PYTHON

Instructions:

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

```
>>> b=1.2
>>> c=a+b
>>> print(a,type(a))-----10 <class 'int'>
>>> print(b,type(b))-----1.2 <class 'float'>
>>> print(c,type(c))-----11.2 <class 'float'>
=====X=====
```

High Level

=>In this context , we have two types of Programming Languages. They are

- a) Low Level Programming Lang
- b) High Level Programming Lang

=>In Low Level Programming Lang, we represent data in the form lower level bits like Binary ,Octal and Hexa Decimal number system.

=>Internally Low Level data in the form lower level bits like Binary ,Octal and Hexa Decimal number system can be automatically converted into Understandable format (Decimal Number System) and Hence Python is considered as High Level Programming Language.

Extensible and Embedded

=>An Extensible Programming language is one, which provides/offers the Services / Facilities of its Coding to be re-used in other Languages.

=>Example: Python Programming provides Extensible because Python code can be called part of C, CPP etc languages.

=>An Embedded Programming language is one, which expects Services / Facilities from other languages related their coding to re-use.

=>Example: Python Programming provides Embedded because Python Program can embed the code of C, CPP etc languages.

```
=====X=====
```

Robust

=>A Programming is said to be Robust if and only if It always provides User-friendly Errors Messages when end-user commits mistakes at implementation level.

=>To get User-friendly error messages, we use exception handling feature.

=>Since Python Programming Provides Exception handling Facility and hence python is one of the Robust (Strong) Programming Language.

X-----

Extensive Supports for Third Party APIs

=>The Python Programming Supports extensively for making use of Third party APIS((Modules) to for developing the real time applications with High Performnace with precise code.

NOTE:

=>First Party---Lang Library(API)---Python In-bult Library--developed by ROSSUIM

=>Second party---OS Library-----

=>Third Party-----vendors---- Travis Oliphant----Numpy(C+PYTHON)

MC Kinney-----PANDAS

Scipy
Scikit

matplotlib

seaborn

NLTP

pip
pip3.10
pip3

PEP

Data Representation in Python

=>To Represent the data in Python Programming, we need 3 things.

- 1) Literals
- 2) Data Types
- 3) Identifiers / Variables

1) Literals:

=> Literals are nothing but input values passing to the program.

=>Basically, we have 4 types of Literals. They are

- a) Numeric Literals
 - i) Integers
 - ii) floats
- b) Boolean Literals
 - i) True
 - ii) False
- c) String Literals

Examples: "Python"
"Guido Van Rossum"

Importance of Identifier (or) Variables

=>All the Literals are stored in Main Memory by allocating sufficient amount memory

by the help of Data Types. To Process the Literals / Values / data stored in Main memory, as programmer, we must give distinct names to the created memory space. Since the distinct names makes us to identify the Literals / Values for processing and hence distinct names are called "IDENTIFIERS".

=>Programmatically , Identifier values can be changed during execution of the program and Hence Identifiers are called "VARIABLES".

=>Hence all Types of Inputs / Literals / data Must be stored in main Memory in the form Variables.

=>Def. of Variable:

=>A Variable is an Identifier, whose values can be changed during execution of the program.

=>ALL THE VARIABLES IN PYTHON ARE CALLED OBJECTS.

=>BEHIND OF ALL OBJECTS , THERE EXIST A CLASS.

Rules for using Variables in Python Programming

1) A Variable Name is a combination of Alphabets, Digits and a special symbol Under score (_).

2) The First letter of variable name must starts with Alphabet (or) Under Score (_)
Examples: sal=1.2---valid

 1sal=34---Invalid
 _sal=3.4---valid
 sal1=4.5---valid
 sal=5.6---valid\
 _123=56---valid
 _=45---valid

3) Within the Variable Name , Special Symbols are not allowed except Under Score (_).

Examples: tot sal=56---Invalid

 tot_sal=45---valid
 tot-sal=45---invalid
 tot\$sal=45---invalid
 \$_sal=34---invalid
 tot_sal#India=56----Valid

 Here the symbol hash (#) in python used for commenting

.4) No keywords to be used as Variable Names. bcoz all the key words are having special meaning to the language compilers.

Examples: while=45---inavlid

 if=56---invalid

 while1=67---valid

 _while=67--valid

 if=78---valid

 int=45.67----VALID

 float=45-----VALID

=>ALL Pre-defined Class Names(int, float.....etc) can used as Varaiable Names

5) All the variable Name are Case Sensitive.

Examples:

```
-----  
    >>> AGE=99  
    >>> age=98  
    >>> Age=97  
    >>> aGe=96  
    >>> agE=95  
    >>> print(AGE,age,Age,aGe,agE)--- 99 98 97 96 95
```

6) Choose the Variable Names in such way that They must be user-friendly.

Data Types in Python

=>The main purpose of data types is that To allocate sufficient amount of memory space for input of the program.

=>In Python Programming, we have 14 data types and they are classified into 6 types.

I) Fundamental Category Data Types

- a) int
- b) float
- c) bool
- d) complex

II) Sequence Category Data Types:

- a) str
- b) bytes
- c) bytearray
- d) range

III) List Category Data Types (Collections Data Type)

- a) list
- b) tuple

IV) Set Category Data Types (Collections Data Type)

- a) set
- b) frozenset

V) Dict Category Data Types (Collections Data Type)

- a) dict

VI) None Category Data Type

- a) NoneType
-

I) Fundamental Category Data Types

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

=>Fundamental Category contains 4 data types. They are

- a) int
 - b) float
 - c) bool
 - d) complex
-

int

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

=>The purpose of 'int' data type is that " To Store Integer / Whole / Integral Values."

=>"Integer / Whole / Integral Values are nothing but data without decimal places.

Examples:

>>> a=123

```
>>> print(a)-----123
>>> type(a)-----<class 'int'>
>>> print(a, type(a), id(a))---123 <class 'int'> 2533443702832
>>> a=1567
>>> print(a, type(a), id(a))--1567 <class 'int'> 2533444737104
```

=>With 'int' data type, we can also store Number System Data.

=>In Any Programming Language (Python, C, CPP, JAVA), we have 4 types of Number Systems. They are

- i) Decimal Number System (default).
- ii) Binary Number System.
- iii) Octal Number System.
- iv) Hexa Decimal Number System.

i) Decimal Number System (default):

=>It is one of the default number System

=>The Decimal Number System (default) are

Digits: 0 1 2 3 4 5 6 7 8 9
Total Number of Digits: 10
Base : 10

=>Base 10 Literals are called Decimal Number System Data.

ii) Binary Number System :

=>The Digits in Binary Number System are

Digits: 0 1
Total Number of Digits: 2
Base : 2

=>Base 2 Literals are called Binary Number System Data.

=>To store Binary data in Python Programming Environment, binary data must be preceded with either 0b or 0B.

=>Syntax:- varname=0b Binary Data

(OR)

varname=0B Binary Data

=>Even we store Binary data and when we display The Binary Data converted into Equivalent Decimal Number System data.

Examples:

```
>>> a=0b1110
>>> print(a, type(a))-----14 <class 'int'>
>>> a=0B1110
>>> print(a, type(a))-----14 <class 'int'>
>>>
>>> a=0b1010
>>> print(a, type(a))-----10 <class 'int'>
>>> a=0B1010
>>> print(a, type(a))-----10 <class 'int'>
```

```
>>> a=0B10102----SyntaxError: invalid digit '2' in binary literal
```

III) Octal Number System :

=>The Digits in Octal Number System are

Digits: 0 1 2 3 4 5 6 7

Total Number of Digits: 8

Base : 8

=>Base 8 Literals are called Octal Number System Data.

=>To store Octal data in Python Programming Environment, Octal data must be preceded with either 0o or 0O.

=>Syntax:- varname=0o Octal Data

(OR)

varname=0O Octal Data

=>Even we store Octal data and when we display The Octal Data converted into Equivalent Decimal Number System data.

int

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

=>The purpose of 'int' data type is that " To Store Integer / Whole / Integral Values."

=>"Integer / Whole / Integral Values are nothing but data without decimal places.

Examples:

```
>>> a=123
```

```
>>> print(a)-----123
```

```
>>> type(a)-----<class 'int'>
```

```
>>> print(a, type(a), id(a))---123 <class 'int'> 2533443702832
```

```
>>> a=1567
```

```
>>> print(a, type(a), id(a))--1567 <class 'int'> 2533444737104
```

=>With 'int' data type, we can also store Number System Data.

=>In Any Programming Language (Python, C, CPP, JAVA), we have 4 types of Number Systems. They are

- i) Decimal Number System (default).
 - ii) Binary Number System.
 - iii) Octal Number System.
 - iv) Hexa Decimal Number System.
-

i) Decimal Number System (default):

=>It is one of the default number System

=>The Decimal Number System (default) are

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

Total Number of Digits: 10

Base : 10

=>Base 10 Literals are called Decimal Number System Data.

ii) Binary Number System :

=>The Digits in Binary Number System are

Digits: 0 1

Total Number of Digits: 2

Base : 2

=>Base 2 Literals are called Binary Number System Data.

=>To store Binary data in Python Programming Environment, binary data must be preceded with either 0b or 0B.

=>Syntax:- varname=0b Binary Data

(OR)

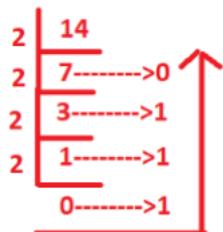
varname=0B Binary Data

=>Even we store Binary data and when we display The Binary Data converted into Equivalent Decimal Number System data.

Conversion from Decimal Number System Data into Binary System Data

Q1) Convert $(14)_{10} \rightarrow (x)_2$ find $x=1110$

Solution:



Hence $(14)_{10} \rightarrow (1110)_2$

Conversion from Binary System Data into Decimal Number System Data

Q2) Convert $(1110)_2 \rightarrow (x)_{10}$ find $x=14$

Solution:

$$\begin{array}{r}
 1 & 1 & 1 & 0 \\
 3 & 2 & 1 & 0 \\
 2 & 2 & 2 & 2 \\
 \hline
 3 & 2 & 1 & 0 \\
 = 1 \times 2 + 1 \times 2 + 1 \times 2 + 0 \times 2 \\
 = 1 \times 8 + 1 \times 4 + 1 \times 2 + 0 \times 1 \\
 = 8 + 4 + 2 + 0 \\
 = 14
 \end{array}$$

Hence $(1110)_2 \rightarrow (14)_{10}$

Examples:

```

>>> a=0b1110
>>> print(a,type(a))----14 <class 'int'>
>>> a=0B1110
>>> print(a,type(a))----14 <class 'int'>
>>>
>>> a=0b1010
>>> print(a,type(a))----10 <class 'int'>
>>> a=0B1010
>>> print(a,type(a))----10 <class 'int'>

```

>>> a=0B10102----SyntaxError: invalid digit '2' in binary literal

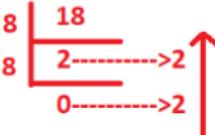
III) Octal Number System :

=>The Digits in Octal Number System are

Digits: 0 1 2 3 4 5 6 7
 Total Number of Digits: 8
 Base : 8

- =>Base 8 Literals are called Octal Number System Data.
- =>To store Octal data in Python Programming Environment, Octal data must be preceded with either 0o or 0O.
- =>Syntax:- varname=0o Octal Data
 (OR)
 varname=0O Octal Data

=>Even we store Octal data and when we display The Octal Data converted into Equivalent Decimal Number System data.

Conversion from Decimal Number System Data into Octal Number System Data	Conversion from Octal Data into Decimal Number System Data						
<p>Q1) $(18)_{10} \longrightarrow (x)_8$ find x =22</p> <p>Sol:-</p>  <p>Hence $(18)_{10} \longrightarrow (22)_8$</p>	<p>Q1) Convert $(22)_8 \longrightarrow (x)_{10}$ find x =18</p> <p>Sol:-</p> <table style="margin-left: auto; margin-right: auto;"> <tr><td>2</td><td>2</td></tr> <tr><td>1</td><td>0</td></tr> <tr><td>8</td><td>8</td></tr> </table> $\Rightarrow 2 \times 8 + 2 \times 8^0$ $\Rightarrow 16 + 2$ $\Rightarrow 18$ <p>Hence $(22)_8 \longrightarrow (18)_{10}$</p>	2	2	1	0	8	8
2	2						
1	0						
8	8						

 Examples:

```
>>> a=0o22
>>> print(a,type(a))----18 <class 'int'>
>>> a=0o345
>>> print(a,type(a))----229 <class 'int'>
>>> a=0o299----SyntaxError: invalid digit '9' in octal literal
>>> a=0o288---SyntaxError: invalid digit '8' in octal literal
```

IV) Hexa Decimal Number System :

=>The Digits in Hexa Decimal Number System are
 Digits: 0 1 2 3 4 5 6 7, 8, 9 , A(10) ,B(11), C(12) , D(13), E(14) ,

F(15)

Total Number of Digits: 16
 Base : 16

- =>Base 16 Literals are called Hexa Decimal Number System Data.
- =>To store Hexa Decimal data in Python Programming Environment, Hexa Decimal data must be preceded with either 0x or 0X.

=>Syntax:- varname=0x Hexa Decimal Data
 (OR)
 varname=0X Hexa Decimal Data

=>Even we store Hexa Decimal data and when we display The Hexa Decimal Data converted into Equivalent Decimal Number System data.

Conversion from Decimal Number System Data into Hexa Decimal Number System	Conversion from Hexa Decimal Number System into Decimal Number System Data
<p>Q1) $(172)_{10} \longrightarrow (x)_{16}$ find x</p> <p>Sol:</p> <p>Hence $(172)_{10} \longrightarrow (AC)_{16}$</p>	<p>Q1) $(AC)_{16} \longrightarrow (x)_{10}$ find $x=172$</p> <p>Sol:</p> $ \begin{array}{r} & A & C \\ & & \\ 16 & 1 & 0 \\ & & \\ & 1 & 0 \\ \hline & 1 & 0 \end{array} $ $ \Rightarrow A \times 16 + C \times 16^0 $ $ \Rightarrow 10 \times 16 + 12 \times 1 $ $ \Rightarrow 160 + 12 $ $ \Rightarrow 172 $ <p>Hence $(AC)_{16} \longrightarrow (172)_{10}$</p>

Examples:

```

>>> a=0xAC
>>> print(a,type(a))-----172 <class 'int'>
>>> a=0x300
>>> print(a,type(a))-----768 <class 'int'>
>>> a=0xBEE
>>> print(a,type(a))-----3054 <class 'int'>
>>> a=0xFACE
>>> print(a,type(a))-----64206 <class 'int'>
>>> a=0xfacer-----SyntaxError: invalid hexadecimal literal
>>> a=0xzxzx-----SyntaxError: invalid hexadecimal literal
    
```

Base Conversion techniques in Python

=>The purpose of Base Conversion techniques in Python is that " Converting One Base Value into another Base Value."

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

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

1) bin():

=>This function is used for converting Other Number Systems data into Binary Number System.

=>Syntax:- varname=bin(Decimal / Octal / Hexa Decimal Data)

Examples:

```
>>> a=15 # Decmal number system data
>>> b=bin(a)
>>> print(b)-----0b1111
>>> b=bin(25)
>>> print(b)-----0b11001
>>> a=0o22 #octal data
>>> b=bin(a)
>>> print(b)-----0b10010
>>> b=bin(0o45)
>>> print(b)-----0b100101
>>> a=0xF # Hexa Decimal
>>> b=bin(a)
>>> print(b)-----0b1111
>>> a=0xFACE
>>> b=bin(a)
>>> print(b)-----0b1111101011001110
>>> b=bin(0xBEE)
>>> print(b)-----0b101111101110
>>> b=bin(0xBEER)-----SyntaxError: invalid hexadecimal literal
>>> x1=bin(0x0133fabe11)
>>> print(x1)-----0b100110011111101011111000010001
```

2) oct():

=>This function is used for converting Other Number Systems data into Octal Number System.

=>Syntax:- varname=oct(Decimal / Binary / Hexa Decimal Data)

Examples:

```
>>> a=18 # Decimal data
>>> c=oct(a)
>>> print(c)-----0o22
>>> c=oct(234)
>>> print(c)-----0o352
>>> a=0b1111
>>> c=oct(a)
>>> print(c)-----0o17
>>> c=oct(0b1010)
>>> print(c)-----0o12
```

```
>>> c=oct(0b101000000111000111000011)
>>> print(c)-----0o240361703
>>> a=0xACE
>>> c=oct(a)
>>> print(c)-----0o5316
>>> c=oct(0xDEE)
>>> print(c)-----0o6756
>>> c=oct(0xDAD)
>>> print(c)-----0o6655
```

3) hex()

=>This function is used for converting Other Number Systems data into Hexa Decimal Number System.

=>Syntax:- varname=hex(Decimal / Binary / octal Data)

Examples:

```
>>> a=15 # decimal data
>>> h=hex(a)
>>> print(h)-----0xf
>>> print(hex(172))-----0xac

>>> a=0o22 #octal data
>>> b=hex(a)
>>> print(b)-----0x12
>>> b=hex(0o34562)
>>> print(b)-----0x3972
>>> a=0b1111
>>> b=hex(a)
>>> print(b)-----0xf
>>> b=hex(0b11111111000000001111111111)
>>> print(b)-----0x3ff007ff
```

Special case:

```
>>> a=15
>>> b=bin(a)
>>> print(b, type(b))-----0b1111 <class 'str'>
>>> b----- '0b1111'
>>> b=oct(a)
>>> print(b, type(b))-----0o17 <class 'str'>
>>> b=hex(a)
>>> print(b, type(b))-----0xf <class 'str'>
```

float

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

=>The purpose of float data type is that " To store Floating Point Values (OR) Real constant Values (Numbers with Decimal Places)"

Examples: 23.45, 56.99 0.99etc

-24.56 -4.5 -0.8..etc

In 23.45, 23 is called Integer part and 0.45 is called decimal part

=>Examples:

```
>>> a=23.45
>>> print(a, type(a))-----23.45 <class 'float'>
>>> a=0.99
>>> print(a, type(a))-----0.99 <class 'float'>
>>> a=22/7
>>> print(a, type(a))-----3.142857142857143 <class 'float'>
```

=>float data type can also represent Scientific Notation(Mantisa e Exponent) and it can be converted into Normal Floating point value as

Mantisa x 10 to the power of Exponent

=>The advantage of Scientific Notation is that It save the memory space.

=>The float data type Never stores Number System data directly.

Examples:

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 and False (Logical values)".

=>Internally, The Value of True is taken as 1 and False ia takes as 0

Examples:

```
>>> a=True  
>>> print(a,type(a))-----True <class 'bool'>  
>>> b=False  
>>> print(b,type(b))-----False <class 'bool'>  
>>>  
>>> print(True+True)-----2  
>>> print(True*True)-----1  
>>> print(True+False)-----1  
>>> print(True*False)-----0  
>>> print(True+False*False)---1  
>>> print(2*True+False)--2  
>>> print(True+0b1111)----16  
>>> print(True+0xF)---16
```

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 Imaginary data / complex numbers".

=>The general format of complex number is shown bellow.

a+bj (OR) a-bj

Here 'a' is called Real Part

'b' is called Imaginary Part
and 'j' is $\sqrt{-1}$

=>To get real and imaginary parts from complex object we use two pre-defined attributes . They are

- 1) real
- 2) imag

=>Syntax:

complexobj.real —>Gives real part of complex object

complexobj.imag—>Gives Imaginary part of complex object

=>By default the values of real and imaginary parts of complex objects are treated internally as floating point values.

=>On complex data , we can perform operations like Addition , subtraction , , multiplication...etc

=>In Complex Data Representation, for Real part , we can use any number System Value but for Imaginary part we must use only Decimal Number System Values only but not other number System values.

Examples:

```
>>> a=2+3j  
>>> print(a,type(a))-----(2+3j) <class 'complex'>  
>>> a=-3-4j
```

```
>>> print(a,type(a))-----(-3-4j) <class 'complex'>
>>> a=2.3+4.5j
>>> print(a,type(a))-----(2.3+4.5j) <class 'complex'>
>>> a=-2.3-4.6j
>>> print(a,type(a))-----(-2.3-4.6j) <class 'complex'>
>>> a=2+3i-----SyntaxError: invalid decimal literal
>>> a=2j
>>> print(a,type(a))-----2j <class 'complex'>
```

Examples:

```
>>> a=10+40j
>>> print(a,type(a))-----(10+40j) <class 'complex'>
>>> print(a.real)-----10.0
>>> print(a.imag)-----40.0
>>> b=2.3-4.5j
>>> print(b.real)-----2.3
>>> print(b.imag)----- -4.5
>>> b=2+4.5j
>>> print(b.real)--- 2.0
>>> print(b.imag)---4.5
```

```
>>> a=2+3j
>>> b=5+6j
>>>
>>> print(a,type(a))-----(2+3j) <class 'complex'>
>>> print(b,type(b))-----(5+6j) <class 'complex'>
>>> c=a+b
>>> print(c, type(c))-----(7+9j) <class 'complex'>
>>>
>>> c=a-b
>>> print(c, type(c))-----(-3-3j) <class 'complex'>
>>> c=a*b
>>> print(c, type(c))-----(-8+27j) <class 'complex'>
```

```
>>> a=True+2j
>>> print(a, type(a))-----(1+2j) <class 'complex'>
>>> a=0b1010-5j
>>> print(a, type(a))-----(10-5j) <class 'complex'>

>>> a=0b1010-0b1111j-----SyntaxError: invalid binary literal
>>> a=0xA+0xFj-----SyntaxError: invalid hexadecimal literal
>>> a=0xA+3j
>>> print(a, type(a))-----(10+3j) <class 'complex'>
```

II) Sequence Category Data Types

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

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

-
- i) str
 - ii) bytes
 - iii) bytearray
 - iv) range
-

str

Index:

- =>Purpose
 - =>Number of ways to organize str data
 - =>Number of Notation to organize str data
 - =>Operations on str data
 - a) Indexing
 - b) Slicing
-

=>'str' is one of the pre-defined class and treated as Sequence Category.
=>The purpose of 'str' data type is that "To store String / Text / Alpha-numeric data".

=>Number of ways to organize/ strore str data

=>In Python Programming, we have two ways to organize str data. They are

- a) Single Line Str data
 - b) Multi Line Str data
-

a) Single Line Str data:

- =>The single line string data is one , which is enclosed within single Quotes or double quotes.
- =>In otherwords, To organize single line string data, we must use single quotes or double quotes.
- =>Syntax1: " Single line String data "
 (OR)
- =>Syntax2: ' Single line String data '

=>We can't use single Quotes or double quotes for organizing Multi Line String data.

Examples:

```
>>> s1="Guido Van Rossum"  
>>> print(s1,type(s1))-----Guido Van Rossum <class 'str'>  
>>> s2='Guido Van Rossum'  
>>> print(s2,type(s2))-----Guido Van Rossum <class 'str'>  
>>> s3="A"  
>>> print(s3,type(s3))-----A <class 'str'>  
>>> s4='A'  
>>> print(s4,type(s4))-----A <class 'str'>  
>>> s5="23456"  
>>> print(s5,type(s5))-----23456 <class 'str'>
```

```
>>> s6="Python3.10"
>>> print(s6,type(s6))-----Python3.10 <class 'str'>
>>> s7="String"
>>> print(s7,type(s7))-----String <class 'str'>
>>> s7='String'
>>> print(s7,type(s7))-----String <class 'str'>
>>> addr1="Guido van Rossum
SyntaxError: unterminated string literal (detected at line 1)
>>> addr1='Guido van Rossum
SyntaxError: unterminated string literal (detected at line 1)
```

Hence To organize Multi line string data, we must use Tripple single quotes or tripple double quotes.

b) Multi Line Str data:

=>The Multi Line string data is one , which is enclosed within tripple single Quotes or tripple double quotes.
=>In otherwords, To organize Multi line string data, we must use tripple single quotes or tripple double quotes.

=>Syntax1: " " "

String Data-1

String Data-2

String Data-n " "

(OR)

=>Syntax2: ''' String Data-1

String Data-2

String Data-n '''

=>Hence with Tripple Single Quotes (or) tripple double quotes , we can organize both single and multi line string data.

Examples:

```
>>> addr1="""Guido van Rossum
... HNO:3-4, Red sea side
... Python Software Foundation
... Centrum Wiscunde Informatica
... Nether Lands """
>>> print(addr1,type(addr1))
Guido van Rossum
HNO:3-4, Red sea side
Python Software Foundation
Centrum Wiscunde Informatica
Nether Lands      <class 'str'>
```

```
>>> addr2="James Gosling
... Sun Micro System
... FNO:5-6 Hill Side
... USA "
>>> print(addr2,type(addr2))
                James Gosling
                Sun Micro System
                FNO:5-6 Hill Side
                USA          <class 'str'>
-----
>>> s1="""Python is an oop lang """
>>> print(s1,type(s1))-----Python is an oop lang <class 'str'>
>>> s2="java is also oop lang"
>>> print(s2,type(s2))-----java is also oop lang <class 'str'>
>>> s3="""A"""
>>> print(s3,type(s3))-----A <class 'str'>
>>> s4="A"
>>> print(s4,type(s4))-----A <class 'str'>
```

=>Hence , we have 4 Notations to organize String data . They are

- a) Single Quotes
 - b) Double Quotes
 - c) Tripple Single Quotes
 - d) Tripple Double Quotes
-

Operations on str data

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

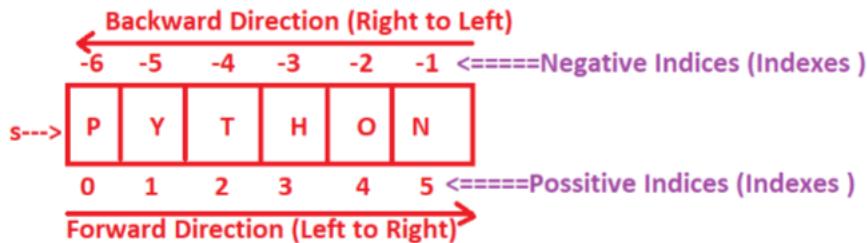
- 1) Indexing
- 2) Slicing

Internal Memory Representation of str data

Consider the following str data

```
>>>s="PYTHON"
```

=>When the above statement is executed, internally memory representation for object 's' is shown below



```
>>>print( s[0])-----P  
>>>print( s[1])-----Y
```

1) Indexing:

=>The process of obtaining an element from given str object by passing valid index value.

=>Syntax: strobj [Index]

=>Index can either Possitive or Negative

=>If we enter valid Index then get Value at that Index

=>If we enter Invalid Index then get IndexError

Examples:

```
>>> s="PYTHON"  
>>> print(s[0])-----P  
>>> print(s[-6])-----P  
>>> print(s[1])-----Y  
>>> print(s[2])-----T  
>>> print(s[4])-----O  
>>> print(s[5])-----N  
>>> print(s[-1])-----N  
>>> print(s[-2])-----O  
>>> print(s[-3])-----H  
>>> print(s[-4])-----T  
>>> print(s[-5])----Y  
>>> print(s[-6])----P  
>>> print(s[10])----IndexError: string index out of range  
>>> print(s[-7])---IndexError: string index out of range
```

=====

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

=>The purpose of bytes data type is that "To store the Numerical Possitive Integer Values ranges from 0 to 256. i.e It stores Numerical Possitive Integer Values ranges from 0 to 255 (256-1) only.

=>bytes data type does not have any Symbolic notation for organizing the data but we can convert the data of any type into bytes data type by using bytes().

Syntax: varname=bytes(object)

=>On the object of bytes data type, we can perform Indexing and Slicing Operations.
=>An object of bytes data type maintains insertion order (OR) In otherwords, In whichever order we insert the data in the same order the data will be displayed.
=>An object of bytes belongs to Immutable bcoz 'bytes' object does not support item assignment.

Examples:

```
>>> t=(10,20,30,256,34,0)
>>> print(t,type(t))----(10, 20, 30, 256, 34, 0) <class 'tuple'>
>>> b=bytes(t)----ValueError: bytes must be in range(0, 256)
-----
>>> t=(10,-20,30,255,34,0)
>>> print(t,type(t))----(10, -20, 30, 255, 34, 0) <class 'tuple'>
>>> b=bytes(t)----ValueError: bytes must be in range(0, 256)
-----
>>> t=(10,20,30,255,34,0)
>>> print(t,type(t))----(10, 20, 30, 255, 34, 0) <class 'tuple'>
>>> b=bytes(t)
>>> print(b, type(b))----b'\n\x14\x1e\xff"\x00' <class 'bytes'>
>>> for v in b:
...     print(v)
...
    10
    20
    30
    255
    34
    0
>>> print(b[0])-----10
>>> print(b[-1])-----0
>>> print(b[-2])-----34
>>> print(b[2:5])-----b'\x1e\xff"'
>>> for v in b[2:5]:
...     print(v)
...
    30
    255
    34
>>> print(b, type(b),id(b))----b'\n\x14\x1e\xff"\x00' <class 'bytes'> 2360235709168
>>> print(b[0])-----10
>>> b[0]=100 ----TypeError: 'bytes' object does not support item assignment
=====X=====
>>> l=[10,20,30,"KVR",34,56,78]
>>> print(l,type(l))----[10, 20, 30, 'KVR', 34, 56, 78] <class 'list'>
>>> b=bytes(l)----TypeError: 'str' object cannot be interpreted as an integer
-----
>>> l=[10,20,30,0,34,56,78,255]
>>> print(l,type(l))----[10, 20, 30, 0, 34, 56, 78, 255] <class 'list'>
```

```
>>> b=bytes(l)
>>> print(b,type(b), id(b))--b'\n\x14\x1e\x00"8N\xff' <class 'bytes'> 2360235711472
>>> print(b[0])---10
>>> print(b[-1])---255
>>> for k in b:
...     print(k)
...
10
20
30
0
34
56
78
255
>>> for k in b[::-1]:
...     print(k)
...
255
78
56
34
0
30
20
10
>>> for k in b[::-2]:
...     print(k)
...
10
30
34
78
>>> for k in b[::-2]:
...     print(k)
...
255
56
0
20
>>> b[0]=200-----TypeError: 'bytes' object does not support item assignment
=====X=====
```

bytarray

=>'bytarray' is one of the pre-defined class and it is treated as Sequence Data Type.
=>The purpose of bytarray data type is that "To store the Numerical Positive Integer Values ranges from 0 to 256. i.e It stores Numerical Positive Integer Values ranges from 0 to 255 (256-1) only.

=>bytarray data type does not have any Symbolic notation for organizing the data but we can convert the data of any type into bytarray data type by using `bytarray()`.

Syntax: `varname=bytarray(object)`

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

=>An object of bytarray data type maintains insertion order (OR) In otherwords, In whichever order we insert the data in the same order the data will be displayed.

=>An object of bytarray belongs to mutable bcoz It allows us to do the changes at same address.

Note:- The Functionality of bytarray is exactly similar to bytes data type but an object of bytarray belongs to mutable and an object of bytes belongs to immutable.

Examples:

```
>>> l=[10,20,30,45,67,89]
>>> print(l,type(l))-----[10, 20, 30, 45, 67, 89] <class 'list'>
>>> ba=bytarray(l)
>>> print(ba,type(ba),id(ba))---bytarray(b'\n\x14\x1e-CY') <class 'bytarray'>
2360235852976
>>> print(ba[0])---10
>>> print(ba[1])---20
>>> print(ba[-1])---89
>>> for x in ba:
...     print(x)
...
10
20
30
45
67
89

>>> for x in ba[::-1]:
...     print(x)
...
89
```

```

67
45
30
20
10
>>> for x in ba[2:5]:
...     print(x)
...
30
45
67
>>> print(ba[0])-----10
>>> ba[0]=153
>>> print(ba,type(ba),id(ba))---bytearray(b'\x99\x14\x1e-CY') <class 'bytearray'>
2360235852976
>>> for k in ba:
...     print(k)
...
153
20
30
45
67
89

```

range

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

=>The purpose of range data type is that " To Store Sequence of Numerical Integer Values with Equal Interval ".

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

=>An object of range data type belongs to Immutable

=>range data type contains 3 types of Syntaxes. They are

Syntax1:

varname=range(value)

=>Here varname is an object of <class,'range'>

=>This syntax generates the range of values from 0 to value-1

Examples:

```

>>> r=range(11)
>>> print(r, type(r))-----range(0, 11) <class 'range'>
>>> for v in r:
...     print(v)

```

```
...
0
1
2
3
4
5
6
7
8
9
10
```

```
>>> for v in range(6):
...     print(v)
```

```
...
0
1
2
3
4
5
```

Syntax2:- varname=range(start,stop)

=>This syntax generates range values from start to stop-1 values
Examples:

```
-----
>>> r=range(10,21)
>>> print(r,type(r))-----range(10, 21) <class 'range'>
>>> for v in r:
...     print(v)
```

```
...
10
11
12
13
14
15
16
17
18
19
20
```

```
>>> for k in range(100,106):
...     print(k)
```

```
...
100
101
102
103
```

```
104  
105  
>>> for k in range(50,56):  
...     print(k)  
...  
50  
51  
52  
53  
54  
55
```

=>NOTE: In the above two syntaxes the default interval is 1 and generates range of values in Forward direction only.

Syntax-3:

varname=range(start,stop,step)

=>This syntax generates range of values from start to stop-1 with specified equal interval of value (step)

Examples:

```
>>> r=range(10,21,2)  
>>> print(r,type(r))-----range(10, 21, 2) <class 'range'>  
>>> for x in r:  
...     print(x)  
...  
10  
12  
14  
16  
18  
20  
>>> for x in range(10,51,10):  
...     print(x)  
...  
10  
20  
30  
40  
50
```

Note: range(Value)
range(start,stop)
range(start,stop,step)

Q1) Generate 0 1 2 3 4 5 6 7 8 9 10---range(11)
>>> for x in range(11):
... print(x)

```
...  
0
```

```
1  
2  
3  
4  
5  
6  
7  
8  
9  
10
```

Q2) Generate 100 101 102 103 104 105----range(100,106)

```
>>> for x in range(100,106):  
...     print(x)  
...  
100  
101  
102  
103  
104  
105
```

Q3) Generate 10 15 20 25 30 35 40---range(10,41,5)

```
>>> for x in range(10,41,5):  
...     print(x)  
...  
10  
15  
20  
25  
30  
35  
40
```

Q4) Generate 10 20 30 40 50 60 70 80 90 100--range(10,101,10)

```
>>> for x in range(10,101,10):  
...     print(x)  
...  
10  
20  
30  
40  
50  
60  
70  
80  
90  
100
```

Q5) Generate -1 -2 -3 -4 -5 -6 -7 -8 -9 -10---- range(-1,-11,-1)

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

Q6) Generate -10 -20 -30 -40 -50---range(-10,-51,-10)

```
>>> for x in range(-10,-51,-10):
...     print(x)
...
-10
-20
-30
-40
-50
```

Q6) Generate 10 9 8 7 6 5 4 3 2 1---range(10,0,-1)

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

Q7) Generate 100 90 80 70 60 50 --range(100,49,-10)

```
>>> for x in range(100,49,-10):
...     print(x)
...
100
90
80
70
60
50
```

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

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

```
>>> r=range(100,151,10)
>>> print(r)-----range(100, 151, 10)
>>> for x in r:
...     print(x)
...
100
110
120
130
140
150
```

```
>>> print(r[2])-----120
>>> print(r[-1])-----150
```

```
>>> for x in r[2:]:
...     print(x)
...
120
130
140
150
```

```
>>> for x in r[::-1]:
...     print(x)
...
150
140
130
120
110
100
```

```
>>> print( range(1000,1050,10)[2])-----1020
>>> print( range(1000,1050,10)[-1])---1040
>>> for x in range(1000,1051,10)[::-1]:
...         print(x)
```

```
...
1050
1040
1030
1020
1010
1000
=====
>>> r=range(10,100,10)
>>> for x in r:
...     print(x)
...
10
20
30
40
50
60
70
80
90
>>> print(r[1])-----20
>>> r[1]=200----TypeError: 'range' object does not support item assignment
>>>
```

Example-----range data type

- 1) 0 1 2 3 4 5 6 7 8 9 10
 - 2) 100 101 102 103 104 105
 - 3) 10 15 20 25 30 35 40
 - 4) 10 20 30 40 50 60 70 80 90 100

 - 5) -1 -2 -3 -4 -5 -6 -7 -8 -9 -10
 - 6) -10 -20 -30 -40 -50

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

III) List Category Data Types (Collections Data Type)

=>The purpose of List Category Data Types is that " To store multiple values either of same type or different type or both types with Unique and Duplicates in single object."

=>We have 2 types of data types in List Category. They are

- 1) list**
 - 2) tuple**
-

1) list

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

=>The purpose of list data type is that "To store multiple values either of same type or

different type or both types with Unique and Duplicates in single object".

=>To store the elements in list , the elements must be written (or) enclosed within Square Brackets [] and the elements must be separated by comma.

Syntax:- **varname=[val1,val2,...,val-n] # Non-empty list**
varname= [] #empty list
varname=list() # empty list

=>An object of list data type maintains insertion order.

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

=>An Object of list belongs to mutable.

Examples:

```
>>> l1=[10,20,-30,40,10,60,20]
>>> print(l1,type(l1))---->[10, 20, -30, 40, 10, 60, 20] <class 'list'>
>>> l2=[100,"Rossum",45.67,True,2+3j,"Python"]
>>> print(l2,type(l2))---->[100, 'Rossum', 45.67, True, (2+3j), 'Python'] <class 'list'>
>>>
>>> len(l1)-----7
>>> len(l2)-----6
>>> l3=[]
>>> print(l3,type(l3))---->[] <class 'list'>
>>> len(l3)-----0
>>> l4=list()
>>> len(l4)-----0
>>> print(l4,type(l4))---->[] <class 'list'>
```

```
>>> l2=[100,"Rossum",45.67,True,2+3j,"Python"]
>>> print(l2[0])-----100
>>> print(l2[1])-----Rossum
>>> print(l2[-1])-----Python
>>> print(l2[2:5])-----[45.67, True, (2+3j)]
>>> l2=[100,"Rossum",45.67,True,2+3j,"Python"]
>>> print(l2,type(l2), id(l2))---[100, 'Rossum', 45.67, True, (2+3j), 'Python']
                                         <class 'list'> 2438540918080
>>> l2[2]=66.99
>>> print(l2,type(l2), id(l2))---[100, 'Rossum', 66.99, True,
                                         (2+3j),
'Python']<class 'list'> 2438540918080
=====X=====
```

Pre-defined Functions in list

=>Along with Indexing and Slicing Operations, on the object of list we can perform various other operations by using pre-defined functions present in list.

=>The pre-defined functions present in list are

1) append():

=>This Function is used for appending (adding at end) new element to existing elements of list

=>Syntax:- listobj.append(Value)

Examples:

```
>>> l1=[10,"Rossum"]
>>> print(l1,type(l1),id(l1))-----[10, 'Rossum'] <class 'list'> 1957574950720
>>> l1.append(23.45)
>>> print(l1,type(l1),id(l1))---[10, 'Rossum', 23.45] <class 'list'> 1957574950720
>>> l1.append("Python")
>>> print(l1,type(l1),id(l1))---[10, 'Rossum', 23.45, 'Python'] <class 'list'>
                                         1957574950720
```

```
>>> l2=[]
>>> print(l2,type(l2),id(l2))----[] <class 'list'> 1957574946752
>>> l2.append(100)
>>> l2.append("KVR")
>>> l2.append("Python")
>>> print(l2,type(l2),id(l2))---[100, 'KVR', 'Python'] <class 'list'> 1957574946752
>>> l3=list()
>>> print(l3,type(l3),id(l3))---[] <class 'list'> 1957575198336
>>> l3.append(True)
>>> l3.append(2+3j)
>>> l3.append("Python")
>>> print(l3,type(l3),id(l3))---[True, (2+3j), 'Python'] <class 'list'> 1957575198336
```

2) insert():

=>This function is used for adding an element at specified position/ Index.

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

Examples:

```
>>> l1=[10,"Rossum",34.56]
>>> print(l1,type(l1))---->[10, 'Rossum', 34.56] <class 'list'>
>>> l1.insert(2,"Python")
>>> print(l1,type(l1))---->[10, 'Rossum', 'Python', 34.56] <class 'list'>
>>> l1.insert(2,True)
>>> print(l1,type(l1))---->[10, 'Rossum', True, 'Python', 34.56] <class 'list'>
```

3) remove() ---content based removal

=>This function is used for removing First Occurrence of specified element.

=>If the specified element not present in list then we get ValueError.

=>**Syntax: listobj.remove(Value)**

Examples:

```
>>> l1=[10,20,30,10,20,"Rossum","Python",20]
>>> print(l1)---->[10, 20, 30, 10, 20, 'Rossum', 'Python', 20]
>>> l1.remove(10)
>>> print(l1)---->[20, 30, 10, 20, 'Rossum', 'Python', 20]
>>> l1.remove(20)
>>> print(l1)---->[30, 10, 20, 'Rossum', 'Python', 20]
>>> l1.remove(100)---->ValueError: list.remove(x): x not in list
```

4) pop(Index):—Index based removal

=>This Function is used for removing the element based Valid Existing Index otherwise we get IndexError

=>**Syntax:- listobj.pop(index)**

Examples:

```
>>> l1=[10,20,20,10,20,30,40,10,20]
>>> print(l1)---->[10, 20, 20, 10, 20, 30, 40, 10, 20]
>>> l1.pop(3)---->10
>>> print(l1)---->[10, 20, 20, 20, 30, 40, 10, 20]
>>> l1.pop(0)---->10
>>> print(l1)---->[20, 20, 20, 30, 40, 10, 20]
>>> l1.pop(-2)---->10
>>> print(l1)---->[20, 20, 20, 30, 40, 20]
>>> l1.pop(-4)---->20
```

```
>>> print(l1)-----[20, 20, 30, 40, 20]
>>> l1.pop(-20)-----IndexError: pop index out of range
```

5) pop():

=>This Function is used for deleting the last element of the list object

=>Syntax:- listobj.pop()

Examples:

```
>>> l1=[10,"Arbaj","Python",45.67,2+3j]
>>> print(l1)-----[10, 'Arbaj', 'Python', 45.67, (2+3j)]
>>> l1.pop()-----(2+3j)
>>> print(l1)-----[10, 'Arbaj', 'Python', 45.67]
>>> l1.insert(2,"Java")
>>> print(l1)-----[10, 'Arbaj', 'Java', 'Python', 45.67]
>>> l1.pop()-----45.67
>>> print(l1)-----[10, 'Arbaj', 'Java', 'Python']
>>> l1.pop()-----'Python'
>>> print(l1)-----[10, 'Arbaj', 'Java']
>>> l1.pop()-----'Java'
>>> print(l1)-----[10, 'Arbaj']
>>> l1.pop()-----'Arbaj'
>>> print(l1)-----[10]
>>> l1.pop()-----10
>>> print(l1)----[]
>>> l1.pop()-----IndexError: pop from empty list

>>> [].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 and makes the list empty.

=>Syntax:- listobj.clear()

Examples:

```
>>> l1=[10,"Arbaj","Python",45.67,2+3j]
>>> print(l1)-----[10, 'Arbaj', 'Python', 45.67, (2+3j)]
>>> len(l1)-----5
>>> l1.clear()
>>> print(l1)----[]
>>> len(l1)-----0
```

NOTE:- del() is a General function used for removing the elements of any mutable

object based on indexing and slicing and we can also remove entire object.

Examples:

```
>>> l1=[10,20,20,10,20,30,40,10,20]
>>> print(l1)-----[10, 20, 20, 10, 20, 30, 40, 10, 20]
>>> del(l1[1:3])
>>> print(l1)-----[10, 10, 20, 30, 40, 10, 20]
>>> del(l1[2])
>>> print(l1)-----[10, 10, 30, 40, 10, 20]
>>> del(l1[-2])
>>> print(l1)-----[10, 10, 30, 40, 20]
>>> del l1
>>> print(l1)-----NameError: name 'l1' is not defined. Did you mean: 'l2'?
```

7) count():

=>This function is used for counting number of occurrences of a specified elements

=>**Syntax: listobj.count(value)**

Examples:

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

8) index() :

=>This function is used for finding Index of First Occurrence of the specified element.

=>If the specified element is not found in list object then we get ValueError.

=>**Syntax: listobj.index(element)**

=>Examples:

```
>>> l1=[10,20,20,10,20,30,40,10,20]
>>> print(l1)-----[10, 20, 20, 10, 20, 30, 40, 10, 20]
>>> l1.index(10)-----0
>>> l1.index(20)-----1
>>> l1.index(40)-----6
>>> l1.index(400)-----ValueError: 400 is not in list
```

9) copy() : (Shallow Copy)

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

=>Syntax: listobj2=listobj1.copy()

Examples:

```
>>> l1=[10,"Rossum"]
>>> print(l1,id(l1))-----[10, 'Rossum'] 1957575199232
>>> l2=l1.copy() # Shallow Copy
>>> print(l2,id(l2))-----[10, 'Rossum'] 1957575198400
>>> l1.append("Python")
>>> l2.append("Data Sci")
>>> print(l1,id(l1))-----[10, 'Rossum', 'Python'] 1957575199232
>>> print(l2,id(l2))-----[10, 'Rossum', 'Data Sci'] 1957575198400
```

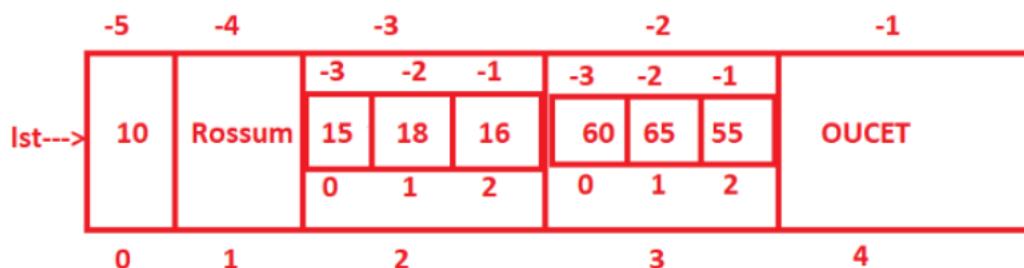
Inner list (or) nested list

=>The process of writing one list into another list is called Inner / Nested list.

=>Syntax: [val1,val2, [val11,val12,val13.....],[val22,val23...],...,val-n]

Memory Management of Inner list

```
>>>lst= [10,"Rossum", [15,18,16], [60,65,55],"OUCET"]
```



Example:-

=>On Inner list , we can perform both Indexing and slicing operations.

=>On Inner list, we can apply all pre-defined functions of list.

```
>>> lst=[10,"Rossum",[15,18,16],[60,65,55],"OUCET"]
>>> print(lst)---[10, 'Rossum', [15, 18, 16], [60, 65, 55], 'OUCET']
>>> print(lst[0])----10
```

```
>>> print(lst[1])-----Rossum
>>> print(lst[2])-----[15, 18, 16]
>>> print(lst[2][0] )-----15
>>> print(lst[2][1] )-----18
>>> print(lst[2][2] )-----16
>>> print(lst[2][-1] )-----16
>>> print(lst[2][-2] )-----18
>>> print(lst[2][-3] )-----15
>>> print(lst[-3])-----[15, 18, 16]
>>> print(lst[-3][-1])-----16
>>> print(lst[-3][-2])-----18
>>> print(lst[-3][-3])-----15
>>> print(lst[-3][:-1])-----[16, 18, 15]
>>> print(lst[3])-----[60, 65, 55]
>>> print(lst[3].sort())----None
>>> print(lst[3])-----[55, 60, 65]
>>> print(lst[4])-----OUCET
>>> lst[2].append(14)-----
>>> print(lst[2])-----[15, 18, 16, 14]
>>> lst[3].insert(2.58)----TypeError: insert expected 2 arguments, got 1
>>> lst[3].insert(2,58)-----
>>> print(lst[3])-----[55, 60, 58, 65]
>>> print(lst)-----[10, 'Rossum', [15, 18, 16, 14], [55, 60, 58, 65], 'OUCET']
>>> lst.pop(2)-----[15, 18, 16, 14]
>>> print(lst)-----[10, 'Rossum', [55, 60, 58, 65], 'OUCET']
>>> lst.pop(-2)-----[55, 60, 58, 65]
>>> print(lst)-----[10, 'Rossum', 'OUCET']
>>> l1=[18,16,14]
>>> l2=[67,34,80]
>>> lst.insert(2,l1)
>>> print(lst)-----[10, 'Rossum', [18, 16, 14], 'OUCET']
>>> lst.append(l2)
>>> print(lst)----[10, 'Rossum', [18, 16, 14], 'OUCET', [67, 34, 80]]
```

2) tuple

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

=>The purpose of list data type is that "To store multiple values either of same type or different type or both types with Unique and Duplicates in single object".

=>To store the elements in tuple , the elements must be written (or) enclosed within Braces () and the elements must be separated by comma.

Syntax:- varname=(val1,val2,...,val-n) # Non-empty tuple

varname= () #empty tuple

varname=tuple() # empty tuple

=>An object of tuple data type maintains insertion order.

=>On the object of tuple , we can perform both Indexing and Slicing Operations.

=>An Object of tuple belongs to immutable.

Note:- The Functionality of tuple is exactly similar to Functionality of list. But an object of list belongs to Mutable and an object of tuple belongs to immutable.

Examples:

```
>>> t1=(10,20,10,34,56,20)
>>> print(t1,type(t1))----(10, 20, 10, 34, 56, 20) <class 'tuple'>
>>> t2=(10,"KVR",33.33,"Python",True)
>>> print(t2,type(t2))----(10, 'KVR', 33.33, 'Python', True) <class 'tuple'>
>>> len(t1)---6
>>> len(t2)---5
>>> t3=() # empty tuple
>>> print(t3,type(t3))----() <class 'tuple'>
>>> t3=tuple() # empty tuple
>>> print(t3,type(t3))----() <class 'tuple'>
>>>
>>> t4=10,"Travis",56.78,"OUCET",2+3j
>>> print(t4,type(t4))----(10, 'Travis', 56.78, 'OUCET', (2+3j)) <class 'tuple'>
>>> print(t4[0])----10
>>> print(t4[1])----Travis
>>> print(t4[-1])----(2+3j)
>>> print(t4[-2])----OUCET
>>> print(t4[2:5])----(56.78, 'OUCET', (2+3j))
>>> print(t4[::-1])----((2+3j), 'OUCET', 56.78, 'Travis', 10)
```

```
>>> lst=[10,20,30,40,"Python"]
>>> print(lst,type(lst))----[10, 20, 30, 40, 'Python'] <class 'list'>
>>> tpl=tuple(lst)
>>> print(tpl,type(tpl))----(10, 20, 30, 40, 'Python') <class 'tuple'>
>>> lst1=list(tpl)
```

```
>>> print(lst1,type(lst1))----[10, 20, 30, 40, 'Python'] <class 'list'>
```

Pre-defined Functions in tuple

=>tuple contains two pre-defined functions. They are

- a) **index()**
 - b) **count()**
-

Examples:

```
>>> t1=(10,20,10,34,56,20)
>>> t1.count(10)----2
>>> t1.count(20)-----2
>>> t1.index(10)-----0
>>> t1.index(56)-----4
```

=>tuple does not contains the following functions.

- a) **append()**
 - b) **insert()**
 - c) **remove()**
 - d) **pop(index)**
 - e) **pop()**
 - f) **copy()**
 - g) **sort()**
 - h) **reverse()**
 - i) **extend()**
-

IV) Set Category Data Types (Collections Data Type)

=>The purpose of Set Category Data Types is that " To store multiple values either of same type or different type or both types with Unique Values in single object(Duplicate values are not allowed). "

=>Set Category Data contains two data types. They are

- 1) set (Mutable & Immutable)
 - 2) frozenset (Immutable)

1) set

=>'set' is one of the pre-defined class and treated as Set Category Data Type.

=>The purpose of set data type is that To store multiple values either of same type or different type or both types with Unique Values in single object (Duplicate values are not allowed). "

=>The elements of set must be written within curly braces { } and elements separated by comma.

=>Syntax:- setobj={Val1,Val2...Val-n} # non-empty set
 setobj=set() # empty set

=>An object of set never maintains insertion order. In otherwords, whatever the elements we are organizing in the object of set , those elements can be displayed in any order out of its many possibilities.

=>On the object of set , we can't perform Indexing and slicing Operations bcoz set object never maintain Insertion Order.

=>An object of set belongs to both Immutable('set' object does not support item assignment) and mutable (in the case add()).

Examples:

```
>>> s1={10,20,30,40,50,60,10,100}
>>> print(s1,type(s1))-----{50, 100, 20, 40, 10, 60, 30} <class 'set'>
>>> s2={10,"KVR",33.33,"Python",True}
>>> print(s2,type(s2))-----{33.33, 'Python', True, 'KVR', 10} <class 'set'>
>>>
>>> s4=set() #empty set
>>> print(s4,type(s4))-----set() <class 'set'>
>>> len(s4)-----0
>>> len(s1)-----7
>>> s5={} # not empty set but it is dict
>>> print(s5,type(s5))-----{} <class 'dict'>
>>> s1={10,20,30,40,50,60,10,100}
>>> print(s1[0])-----TypeError: 'set' object is not subscriptable
>>> print(s1[0:6])-----TypeError: 'set' object is not subscriptable
```

```
>>>  
>>> s1[0]=100---TypeError: 'set' object does not support item assignment  
>>> print(s1,id(s1))---{50, 100, 20, 40, 10, 60, 30} 2190683253088  
>>> s1.add("HYD")  
>>> print(s1,id(s1))---{50, 100, 20, 'HYD', 40, 10, 60, 30} 2190683253088  
  
>>> s1={10,10,10,10,10,10}  
>>> print(s1,type(s1))-----{10} <class 'set'>  
=====X=====
```

pre-defined functions set

=>On the object of set we can perform various operations by using pre-defined functions present in set. They are

1) add():

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

Examples:

```
>>> s1={10,"KVR"}  
>>> print(s1,type(s1),id(s1))-----{10, 'KVR'} <class 'set'> 2190683253088  
>>> s1.add("Python")  
>>> s1.add(45.67)  
>>> print(s1,type(s1),id(s1))---{'Python', 10, 45.67, 'KVR'} <class 'set'>  
2190683253088  
>>> s1.add("satish")  
>>> print(s1,type(s1),id(s1))---{'Python', 10, 45.67, 'KVR', 'satish'} <class 'set'>  
>>> s2=set() #empty set  
>>> print(s2,type(s2),id(s2))----set() <class 'set'> 2190683254432  
>>> s2.add(100)  
>>> s2.add(200)  
>>> s2.add(100)  
>>> s2.add("Django")  
>>> print(s2,type(s2),id(s2))----{200, 'Django', 100} <class 'set'> 2190683254432
```

2) clear():

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

Examples:

```
>>> s1={10,"Rossum","Python",45.67,True,2+3j}  
>>> print(s1,type(s1),id(s1))---{'Python', True, 'Rossum', 10,  
2+3j, 45.67}
```

```
<class 'set'> 2190683252640
>>> len(s1)-----6
>>> s1.clear()
>>> print(s1,type(s1),id(s1))----set() <class 'set'> 2190683252640
>>> len(s1)-----0
```

3) copy():

=>This Function is used for copying the content of one set object object into another set object (Implementing Shallow Copy)

=>Syntax:- setobj2=setobj1.copy()

Examples:

```
>>> s1={10,"Rossum","Python",45.67,True,2+3j}
>>> print(s1,type(s1),id(s1))--{'Python', True, 'Rossum', 10, (2+3j), 45.67}<class 'set'>
2190683253088
>>> s2=s1.copy() # shallow Copy
>>> print(s2,type(s2),id(s2))--{'Python', True, 'Rossum', 10, (2+3j), 45.67} <class 'set'>
2190683253984
>>> s1.add("Java")
>>> s2.add("Django")
>>> print(s1,type(s1),id(s1))--{'Python', True, 'Rossum', 'Java', 10, (2+3j), 45.67}
>>> print(s2,type(s2),id(s2))--{'Python', True, 'Django', 'Rossum', 10, (2+3j),
45.67}<class 'set'> 2190683253984
```

Deep Copy Examples:

```
>>> s1={10,"Rossum","Python",45.67}
>>> print(s1,type(s1),id(s1))--{'Python', 10, 'Rossum', 45.67} <class 'set'>
>>> s2=s1 # Deep Copy
>>> print(s2,type(s2),id(s2))-----{'Python', 10, 'Rossum', 45.67} <class 'set'>
>>> s1.add("Django")
>>> print(s1,type(s1),id(s1))--{'Python', 'Django', 'Rossum', 10, 45.67} <class 'set'>
>>> print(s2,type(s2),id(s2))--{'Python', 'Django', 'Rossum', 10, 45.67} <class 'set'>
```

4) remove() :

=>This function is used for removing the element/key from set object.

=>If the element does not exists in set object then we get KeyError.

=>Syntax:- setobj.remove(element / key)

Examples:

```
>>> s1={10,"Rossum","Python",45.67}
>>> print(s1)-----{'Python', 10, 'Rossum', 45.67}
>>> s1.remove("Python")
>>> print(s1)-----{10, 'Rossum', 45.67}
```

```
>>> s1.remove(45.67)
>>> print(s1)----{10, 'Rossum'}
>>> s1.remove("Rossum")
>>> print(s1)----{10}
>>> s1.remove(10)
>>> print(s1)----set()
>>> s1={10,"Rossum","Python",45.67}
>>> print(s1.remove(1000))----KeyError: 1000
>>> set().remove(10)----KeyError: 10
```

5) discard():

=>This function is used for removing an specified element from set object.
=>if the specified element does not exist in set object, we never get KeyError.
=>Syntax:- setobj.discard(Value)

Examples:

```
>>> s1={10,"Rossum","Python",45.67}
>>> print(s1)----{'Python', 10, 45.67, 'Rossum'}
>>> s1.discard(10)
>>> print(s1)----{'Python', 45.67, 'Rossum'}
>>> s1.discard("Rossum")
>>> print(s1)----{'Python', 45.67}
>>> s1.discard(100) # no error
```

6) pop():

=>This function is used for removing any arbitary element provided element present in setobj otherwise we get KeyError
=>Syntax:- setobj.pop()

Examples:

```
>>> s1={10,"Rossum","Python",45.67}
>>> s1.pop()----'Python'
>>> s1.pop()----10
>>> s1.pop()----45.67
>>> s1.pop()----'Rossum'
>>> s1.pop()----KeyError: 'pop from an empty set'
```

7) isdisjoint():

=>This function returns True provided the two sets are disjoint (there are no common elements) otherwise returns False.
=>Syntax:- setobj1.isdisjoint(setobj2)

Examples:

```
>>> s1={10,20,30,40}
>>> s2={"apple","banana","kiwi"}
>>> s3={10,"apple",20,"kiwi"}
>>> print(s1)----{40, 10, 20, 30}
>>> print(s2)----{'apple', 'kiwi', 'banana'}
>>> print(s3)----{'apple', 10, 20, 'kiwi'}
>>>
>>> s1.isdisjoint(s2)----True
>>> s1.isdisjoint(s3)----False
```

8) issuperset():

Syntax:- setobj1.issuperset(setobj2)

=>This function returns True provided setobj1 contains all the elements of setobj2 otherwise it returns False.

Examples:

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

9) issubset():

Syntax:- setobj1.issubset(setobj2)

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

Examples:

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

10) union():

Syntax:- setobj3=setobj1.union(setobj2)

=>This Function takes all the elements of setobj1 and setobj2 and collects unique elements the place them in setobj3

Examples:

```
>>> s1={10,20,30,40}
>>> s2={10,50,60,70}
```

```
>>> s3=s1.union(s2)
>>> print(s3)---->{70, 40, 10, 50, 20, 60, 30}
```

11) intersection():

Syntax: setobj3=setobj1.intersection(setobj2)

=>This Function takes the common elements from setobj1 and setobj2 and place them in setobj3.

Examples:

```
>>> s1={10,20,30,40}
>>> s2={10,50,60,70}
>>> s3=s1.intersection(s2)
>>> print(s3)---->{10}
>>> s1={10,20,30,40}
>>> s2={60,70,50,80}
>>> s3=s1.intersection(s2)
>>> print(s3)-----set()
>>>
>>> s1={10,20,30,40}
>>> s2={50,60,10,70}
>>> s3={10,70,80,90}
>>> s4=s1.intersection(s2).intersection(s3)
>>> print(s4)---->{10}
```

12) difference():

Syntax:- setobj3=setobj1.difference(setobj2)

=>This Function removes the common elements from setobj1 and setobj2 and takes remaining elements from setobj1 and place them in setobj3

(OR)

Syntax:- setobj3=setobj2.difference(setobj1)

=>This Function removes the common elements from setobj2 and setobj1 and takes remaining elements from setobj2 and place them in setobj3

=>hence setobj1.difference(setobj2) is not equal to setobj2.difference(setobj1)

Examples:

```
>>> s1={10,20,30,40}
>>> s2={30,40,50,60}
```

```
>>> s3=s1.difference(s2)
>>> print(s3)----{10, 20}
>>> s4=s2.difference(s1)
>>> print(s4)----{50, 60}
```

13) symmetric_difference():

Syntax:- setobj3=setobj1. symmetric_difference(setobj2)

=>This function removes the common elements from setobj1 and setobj2 and takes all the remaining elements from both setobj1 and setobj2 and place them in setobj3.

Examples:

```
>>> s1={10,20,30,40}
>>> s2={30,40,50,60}
>>> s3=s1. symmetric_difference(s2)
>>> print(s3)----{10, 50, 20, 60}
>>> s3=s2. symmetric_difference(s1)
>>> print(s3)----{10, 50, 20, 60}
```

Use Case:

Let consider the following problem

cricket players={"Sachin","Kohli","Rohit"}

Tennis players={"Rohit","Ram","Sindu"}

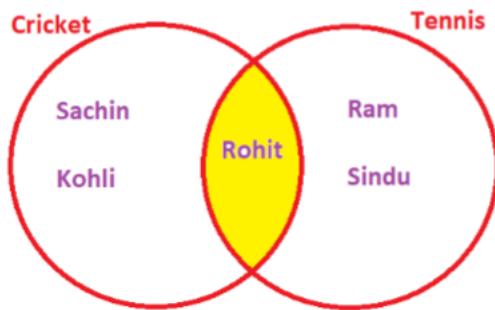
- Q1) Find all the players who are playing either Cricket or tennis or all
- Q2) Find all the players who are playing both the games.
- Q3) Find all the players who are playing Only Cricket but not tennis
- Q4) Find all the players who are playing Only Tennis but not Cricket
- Q5) Find all the players who are exclusively Playing either Cricket or Tennis

Use Case: --Assignment

Let consider the following problem

```
cricket players={"Sachin","Kohli","Rohit"}  
Tennis players={"Rohit","Ram","Sindu"}
```

- Q1) Find all the players who are playing either Cricket or tennis or all -----union()**
- Q2) Find all the players who are playing both the games. -----intersection()**
- Q3) Find all the players who are playing Only Cricket but not tennis-----difference()**
- Q4) Find all the players who are playing Only Tennis but not Cricket -----difference()**
- Q5) Find all the players who are exclusively Playing either Cricket or Tennis ---symmetric_difference()**



Solving:

- Q1) Find all the players who are playing either Cricket or tennis or all**

ANS: >>> cricket={"Sachin","Kohli","Rohit"}
>>> tennis={"Rohit","Ram","Sindu"}
>>> print(cricket)-----{'Rohit', 'Sachin', 'Kohli'}
>>> print(tennis)-----{'Sindu', 'Ram', 'Rohit'}
>>> allplayers=cricket.union(tennis)
>>> print(allplayers)----{'Sachin', 'Sindu', 'Ram', 'Rohit', 'Kohli'}

- Q2) Find all the players who are playing both the games.**

ANS:-

```
>>> cricket={"Sachin","Kohli","Rohit"}  
>>> tennis={"Rohit","Ram","Sindu"}  
>>> print(cricket)-----{'Rohit', 'Sachin', 'Kohli'}  
>>> print(tennis)-----{'Sindu', 'Ram', 'Rohit'}  
>>> bothplayers=cricket.intersection(tennis)  
>>> print( bothplayers)-----{'Rohit'}
```

-
- Q3) Find all the players who are playing Only Cricket but not tennis**

ANS:

```
>>> cricket={"Sachin","Kohli","Rohit"}  
>>> tennis={"Rohit","Ram","Sindu"}  
>>> print(cricket)-----{'Rohit', 'Sachin', 'Kohli'}
```

```
>>> print(tennis)-----{'Sindu', 'Ram', 'Rohit'}
>>> onlycricket=cricket.difference(tennis)
>>> print(onlycricket)-----{'Sachin', 'Kohli'}
```

Q4) Find all the players who are playing Only Tennis but not Cricket

ANS:

```
>>> cricket={"Sachin","Kohli","Rohit"}
>>> tennis={"Rohit","Ram","Sindu"}
>>> print(cricket)-----{'Rohit', 'Sachin', 'Kohli'}
>>> print(tennis)-----{'Sindu', 'Ram', 'Rohit'}
>>> onlytennis=tennis.difference(cricket)
>>> print(onlytennis)-----{'Sindu', 'Ram'}
```

Q5) Find all the players who are exclusively Playing either Cricket or Tennis

ANS:

```
>>> cricket={"Sachin","Kohli","Rohit"}
>>> tennis={"Rohit","Ram","Sindu"}
>>> print(cricket)-----{'Rohit', 'Sachin', 'Kohli'}
>>> print(tennis)-----{'Sindu', 'Ram', 'Rohit'}
>>> criten=cricket.symmetric_difference(tennis)
>>> print(criten)-----{'Sachin', 'Kohli', 'Sindu', 'Ram'}
(OR)
>>> criten=tennis.symmetric_difference(cricket)
>>> print(criten)-----{'Sachin', 'Kohli', 'Sindu', 'Ram'}
```

Use Case: Solving Bitwise Operators

Let consider the following problem

```
cricket players={"Sachin","Kohli","Rohit"}
Tennis players={"Rohit","Ram","Sindu"}
```

Q1) Find all the players who are playing either Cricket or tennis or all

Q2) Find all the players who are playing both the games.

Q3) Find all the players who are playing Only Cricket but not tennis

Q4) Find all the players who are playing Only Tennis but not Cricket

Q5) Find all the players who are exclusively Playing either Cricket or Tennis

```
>>> cricket={"Sachin","Kohli","Rohit"}
>>> tennis={"Rohit", "Ram", "Sindu"}
>>> allplayers=cricket | tennis-----ANS--Q1
>>> print(allplayers)-----{'Sachin', 'Sindu', 'Ram', 'Rohit', 'Kohli'}
```

```
>>> bothplayers=cricket & tennis-----ANS--Q2
```

```
>>> print(bothplayers)-----{'Rohit'}
```

```
>>> onlycricket=cricket-tennis-----ANS--Q3
>>> print(onlycricket)-----{'Sachin', 'Kohli'}
```



```
>>> onlytennis=tennis-cricket-----ANS--Q4
>>> print(onlytennis)-----{'Sindu', 'Ram'}
```



```
>>> criten=tennis^cricket-----ANS--Q5
>>> print(criten)-----{'Sachin', 'Kohli', 'Sindu', 'Ram'}
```

frozenset

=>'frozenset' is one of the pre-defined class and treated as Set Category Data Type.
=>The purpose of frozenset data type is that To store multiple values either of same type or different type or both types with Unique Values in single object (Duplicate values are not allowed). "
=>We don't have any symbolic Notation for storing the elements in frozenset but we can convert any tuple, list , set type elements to frozenset by using frozenset().

=>Syntax:- frozensetobj=frozenset({Val1,Val2...Val-n})
 or
 frozensetobj=frozenset([Val1,Val2...Val-n])
 or
frozensetobj=frozenset((Val1,Val2...Val-n))

=>An object of frozenset never maintains insertion order. In otherwords, whatever the elements we are organizing in the object of frozenset , those elements can be displayed in any order out of its many possibilities.
=>On the object of frozenset , we can't perform Indexing and slicing Operations bcoz set object never maintain Insertion Order.
=>An object of frozenset belongs to Immutable('frozenset' object does not support item assignment).

=>Note:- The Functionality of frozenset is exactly similar to the functionality of set but an object of set belongs both Mutable and immutable where as an object of frozenset belongs to immutable only.

Examples:

```
>>> s1={10,20,"Ram","Python",34.56}
>>> print(s1,type(s1))---{34.56, 20, 'Ram', 10, 'Python'} <class 'set'
>>> fs1=frozenset(s1)
>>> print(fs1,type(fs1))---frozenset({34.56, 20, 'Ram', 10, 'Python'})
                                         <class 'frozenset'>
>>> print(fs1[0])---TypeError: 'frozenset' object is not subscriptable
>>> fs1[0]=100---TypeError: 'frozenset' object does not support item assignment
>>> fs1.add(100)---AttributeError: 'frozenset' object has no attribute 'add'
>>> lst=[10,10,20,30,"KVR","Python",True]
>>> print(lst,type(lst))---[10, 10, 20, 30, 'KVR', 'Python', True] <class 'list'>
>>> fs2=frozenset(lst)
>>> print(fs2,type(fs2))---frozenset({True, 10, 'Python', 20, 'KVR', 30}) <class
                                         'frozenset'>
>>> tpl=(10,10,20,30,"KVR","Python",True)
>>> print(tpl,type(tpl))---(10, 10, 20, 30, 'KVR', 'Python', True) <class 'tuple'>
>>> fs3=frozenset(tpl)
>>> print(fs3,type(fs3))---frozenset({True, 10, 'Python', 20, 'KVR', 30}) <class
                                         'frozenset'>
=====X=====
```

pre-defined functions in frozenset

=>The following are pre-defined functions in frozenset

- a) copy()
- b) isdisjoint()
- c) issuperset()
- d) issubset()
- e) union()
- f) intersection()
- g) difference()
- h) symmetric_difference()

=>The following pre-defined functions not present in frozenset bcoz an object of frozenset is purely Immutable.

- a) add()
 - b) clear()
 - c) remove()
 - d) pop()
 - f) discard()
-

VI) None Category Data Type

=>This contains class name as "NoneType"

=>The value of NoneType is None

=>None is a pre-defined Value for NoneType data type

=>An object of NoneType can't be create bcoz it contains single value called None.

=>The value None is not a False, space , empty . Hence None itself is the value.

=>If we want to make the object and whose memory space to destroy / remove by the Garbage Collector then whose object value must be made as None.

=>If a Function is not returning any value then its value is None.

Examples:

```
>>> d1={10:3.4,20:4.5,30:6.7,40:3.4}
>>> val=d1.get(10)
>>> print(val)----3.4
>>> print(type(val))-----<class 'float'>
>>> val=d1.get(20)
>>> print(val,type(val))-----4.5 <class 'float'>
>>> val=d1.get(200)
>>> print(val,type(val))---- None <class 'NoneType'>
>>>
>>>
>>> None=23.45-----SyntaxError: cannot assign to None
>>> True=23-----SyntaxError: cannot assign to True
>>> False=34-----SyntaxError: cannot assign to False

>>> n1=NoneType()-----NameError: name 'NoneType' is not defined
>>> None==False-----False
```

```
>>> None=="-----False
```

My requirement

I want to store stno , name , internal marks(sub1,sub2,sub3), Ext Marks(sub1,sub2,sub3) collegne name

[10,"Rossum", 15,20,18,60 65 55,'OUCET"]---no proper arragement

[10,"Rossum", [15,20,30], [60,65,55],"OUCET"]

reverse() sort() extend()

Type Casting Techniques in Python

=>The purpose of Type Casting Techniques in Python is that " To convert One Possible Type of Value into Another Possible Type of Value".

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

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

1) int()

=> int() is used converting "Any Possible Type Value into int type value".

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

Examples:-) (float Value---> int value---> **Possible**)

```
>>> a=10.24
>>> print(a, type(a))-----10.24 <class 'float'>
>>> b=int(a)
>>> print(b,type(b))-----10 <class 'int'>
>>> a=0.99
>>> print(a, type(a))-----0.99 <class 'float'>
>>> b=int(a)
>>> print(b,type(b))-----0 <class 'int'>
```

Examples:- (bool Value---> int value--->**Possible**)

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

Examples:- (Complex Value---> int value--->**Not Possible**)

```
>>> a=2.3+4.5j
>>> print(a, type(a))-----(2.3+4.5j) <class 'complex'>
>>> b=int(a)----TypeError: int() argument must be a string, a bytes-like object or a
real number, not 'complex'
```

Examples:- (Str Value---> int value) **POSSIBLE**

```
>>> a="100" # it is a int str
>>> print(a,type(a))-----100 <class 'str'>
>>> a=a+1-----TypeError: can only concatenate str (not "int") to str
>>> b=int(a) #POSSIBLE
>>> print(b, type(b))---100 <class 'int'>
>>> b=b+1
>>> print(b, type(b))---101 <class 'int'>
```

```
>>>a="KVR" # it is pure str
>>> print(a,type(a))----KVR <class 'str'>
>>> b=int(a)----ValueError: invalid literal for int() with base 10: 'KVR'
```

```
>>> a="12.34" # it is float str
>>> print(a,type(a))----12.34 <class 'str'>
>>> b=int(a)---ValueError: invalid literal for int() with base 10: '12.34'
```

```
>>> a="True" # it is bool str
>>> print(a,type(a))-----True <class 'str'>
>>> b=int(a)----ValueError: invalid literal for int() with base 10: 'True'
```

```
>>> a="2+3j" # it is complex str
>>> print(a,type(a))-----2+3j <class 'str'>
>>> b=int(a)----ValueError: invalid literal for int() with base 10: '2+3j'
```

2) float()

=> float() is used converting "Any Possible Type Value into float type value".
=> Syntax:- varname=float(int / bool / complex /str)

Example (int value--->float--->Possible)

```
>>> a=12
>>> print(a,type(a))-----12 <class 'int'>
>>> b=float(a)
>>> print(b,type(b))-----12.0 <class 'float'>
```

Example (bool value--->float--->Possible)

```
>>> a=True
>>> print(a,type(a))-----True <class 'bool'>
>>> b=float(a)
>>> print(b,type(b))-----1.0 <class 'float'>
>>> a=False
>>> print(a,type(a))-----False <class 'bool'>
>>> b=float(a)
>>> print(b,type(b))-----0.0 <class 'float'>
```

Example (complex value--->float--->Not Possible)

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

Example (Str value--->float Value)

```
>>> a="123" # it is int str
>>> print(a,type(a))-----123 <class 'str'>
>>> b=float(a) # POSSIBLE
>>> print(b,type(b))-----123.0 <class 'float'>
```

```
>>> a="12.34" # it is float str
>>> print(a,type(a))----12.34 <class 'str'>
>>> b=float(a) # POSSIBLE
```

```
>>> print(b, type(b))----12.34 <class 'float'>
-----
>>> a="KVR" # It is pure str
>>> print(a,type(a))----KVR <class 'str'>
>>> b=float(a)----ValueError: could not convert string to float: 'KVR'
```

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

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

3) bool()

=> bool() is used converting "Any Possible Type Value into bool type value".
=>Syntax:- varname=bool(int / float / complex /str)
=>ALL NON-ZERO Values are Treated as TRUE
=>ALL ZERO Values are Treated as FALSE

Example (int value----->bool--->Possible)

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

Example (float value---->bool--->Possible)

```
>>> print(a,type(a))---- 1e-41 <class 'float'>
>>> b=bool(a)
>>> print(b, type(b))----True <class 'bool'>
```

Example:- (complex value---->bool-->Possible)

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

Example:- (str value---->bool)

```
>>> a="123"
>>> print(a,type(a))-----123 <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"
>>> print(a,type(a))-----0 <class 'str'>
>>> b=bool(a)
>>> print(b, type(b))-----True <class 'bool'>
>>> len(a)-----1
>>> a="False"
>>> print(a,type(a))-----False <class 'str'>
>>> b=bool(a)
>>> print(b, type(b))-----True <class 'bool'>
>>> a=""
>>> print(a,type(a))----- <class 'str'>
>>> b=bool(a)
>>> print(b, type(b))-----False <class 'bool'>
>>> len(a)----- 0

>>> a=" "
>>> print(a,type(a))----- <class 'str'>
>>> b=bool(a)
>>> print(b, type(b))----- True <class 'bool'>
>>> len(a)----- 4
```

4) complex()

=> complex() is used converting "Any Possible Type 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'>
>>> a=-4
>>> print(a,type(a))----- -4 <class 'int'>
>>> b=complex(a)
>>> print(b,type(b))----- (-4+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'>
>>> a=-4.5
>>> print(a,type(a))----- -4.5 <class 'float'>
>>> b=complex(a)
>>> print(b,type(b))----- (-4.5+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:(str---->complex)

```
>>> a="12" # it is int str
>>> print(a,type(a))----- 12 <class 'str'>
>>> b=complex(a)
>>> print(b,type(b))----- (12+0j) <class 'complex'>
>>> a="12.3" # it is float str
>>> print(a,type(a))----- 12.3 <class 'str'>
>>> b=complex(a)
>>> print(b,type(b))----- (12.3+0j) <class 'complex'>
>>> a="True" # it is bool str
>>> print(a,type(a))----- True <class 'str'>
>>> b=complex(a)-----ValueError: complex() arg is a malformed string
```

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

5) str()

=> str() is used for converting "All Types of Values into str type value".
=> Syntax:- varname=str(int / float /bool /complex)

Examples:

```
>>> a=12  
>>> print(a,type(a))----12 <class 'int'>  
>>> b=str(a)  
>>> b----- '12'  
>>> print(b,type(b))---- 12 <class 'str'>  
>>> a=2.3  
>>> print(a,type(a))----2.3 <class 'float'>  
>>> b=str(a)  
>>> b----- '2.3'  
>>> print(b,type(b))---- 2.3 <class 'str'>  
>>> a=True  
>>> print(a,type(a))----True <class 'bool'>  
>>> b=str(a)  
>>> b----- 'True'  
>>> print(b,type(b))---- True <class 'str'>  
>>> a=2+3j  
>>> print(a,type(a))----(2+3j) <class 'complex'>  
>>> b=str(a)  
>>> b----- '(2+3j)'  
>>> print(b,type(b))----(2+3j) <class 'str'>
```

Number of approaches to develop python programs

→ In real time, we can develop python programs in 2 approaches. They are

- a) Interactive Mode
- b) Batch Mode

a) Interactive Mode:

=>In this mode, Python Programmer issued a statement and gets an output at a time.

=>This is useful to test one Instruction / statement at a time.

=>This mode of development is not recommended to solve Big Problems bcoz the instructions related big problem solving are not able to save and we can't retrieve those instructions.

=> Examples Software:-

Python Command Prompt, Python IDLE. (These things are comming along with Python Software Installation)

=>Hence to solve Big Problems / Big programs, we must go for Batch Mode Programming.

b) Batch Mode:

=>In this mode of development, Python Programmer develops group / batch of Instructions in a single unit and it must be saved on some file name with an extension .py (FileName.py)

=>Once we save the the instructions on some file name then we can open and access that file name any time in our projects.

Example Softwares:

- 1) Python IDLE --- (Python Software Installation)

The following IDEs need to install Separately --

- 2) Jupiter Note Book (Anakonda)
- 3) Spider (Anakonda)
- 4) VS Code
- 5) Sublime Text
- 6) PyCharm
- 7) Edit Plus

Note:- These are called IDEs (Integrated Development Environment) used for developing Python Programs

```
#This Program computes sum of two numbers
#sumex1.py
a=10
b=20
c=a+b
print("-----")
print("\tR e s u l t")
print("-----")
print("\tVal of a=",a)
print("\tVal of b=",b)
print("\tSum=",c)
print("-----")
```

#This Program computes sum of two numbers

```
#sumex2.py
a=float(input("Enter First Value:"))
b=float(input("Enter second Value:"))
c=a+b
print("-----")
print("\tR e s u l t")
print("-----")
print("\tVal of a=",a)
print("\tVal of b=",b)
print("\tSum=",c)
print("-----")
```

Display the Data / result of Python Program

- =>To display the result of the python program, we have a pre-defined Function called **print()**.
- =>In Other words, **print()** is used for displaying the result of python program on the console (Monitor)
- =>**print()** contains the 5 syntaxes . They are

Syntax-1: **print(value)**
 (or)
 print(value1,value,...value-n)

=>This syntax display only the values on the console.

Examples:

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

Syntax-2 : print(Message)

=>This Syntax displays the Messages(strs) on the console

Examnpls:

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

Syntax-3 : print(Message cum Value)

(OR)

print(Value cum Message)

=>This Syntax displays the Messages(strs) along with values on the console

Examnpls:

```
>>> a=10
>>> b=20
>>> c=a+b
>>> print("sum=",c)-----sum= 30
>>> print(c," is the sum")-----30 is the sum
>>> print("sum of ",a," and ",b,"=",c)--sum of 10 and 20 = 30
>>> a=1
>>> b=2
>>> c=3
>>> d=a+b+c
>>> print("sum of ",a," ,",b," and ",c,"=",d)----sum of 1 , 2 and 3 = 6
```

Syntax-4 : print(Message cum Value format())

(OR)

print(Value cum Message with format())

=>This Syntax displays the values and messages by using format()

=>The purpose of format() is that it supplies the specified variables to the empty curly braces {}.

Examples:

```
>>> a=10
>>> b=20
>>> c=a+b
>>> print("sum={}".format(c) )-----sum=30
>>> print("{} is the sum".format(c))----30 is the sum
>>> print("sum of {} and {}={}{}".format(a,b,c))---sum of 10 and 20=30
>>> a=1
```

```

>>> b=2
>>> c=3
>>> d=a+b+c
>>> print("sum of {},{} and {}={}".format(a,b,c,d))--sum of 1,2 and 3=6
>>> stno=10
>>> sname="Rossum"
>>> print("My Number is {} and name is {}".format(stno,sname))--My Number is 10
                                         and name is Rossum
>>> a=10
>>> b=20
>>> c=a*b
>>> #mul(10,20)=200
>>> print("mul({},{})={}".format(a,b,c))---mul(10,20)=200
>>> print("sum({},{})={}".format(a,b,a+b))---sum(10,20)=30

```

Syntax-5 : **print(Message cum Value with format specifiers) (OR)**

print(Value cum Message with format specifiers)

- =>This Syntax display the values cum messages with format specifiers.
- =>The format specifier %d is for displaying Integer data, %f for float value and %s for str value.
- =>If don't have any format specifier then those values converted into str and use %s

Examples:

```

>>> a=10
>>> b=5
>>> c=a+b
>>> print("sum=%d" %c)-----sum=15
>>> print("sum=%f" %c)-----sum=15.000000
>>> print("sum=%0.2f" %c)-----sum=15.00
>>> print("%d is the sum" %c)-----15 is the sum
>>> print("sum of %d and %d=%d" %(a,b,c))---sum of 10 and 5=15
>>> print("sum of %f and %f=%f" %(a,b,c))---sum of 10.000000 and
                                         5.000000=15.000000
>>> print("sum of %0.2f and %0.2f=%0.3f" %(a,b,c))---sum of 10.00 and 5.00=15.000
>>> print("sum of %0.2f and %0.2f=%d" %(a,b,c))---sum of 10.00 and 5.00=15
>>> print("sum(%d,%d)=%0.2f" %(a,b,c))---sum(10,5)=15.00
>>> print("sub(%d,%d)=%d mul(%d,%d)=%d" %(a,b,a-b,a,b,a*b))
                                         sub(10,5)=5
mul(10,5)=50
>>>
>>> stno=10
>>> name="Rossum"
>>> print("My Number is %d and name is %s" %(stno,name))
                                         My Number is 10 and
name is Rossum
>>> lst=[10,"KVR",22.22,"Hyd","Python"]
>>> print("content of lst=",lst)--content of lst= [10, 'KVR', 22.22, 'Hyd', 'Python']
>>> print("content of lst={}".format(lst))--content of lst=[10, 'KVR', 22.22, 'Hyd',

```

```
'Python']
>>> print("content of lst=%s" %lst)--content of lst=[10, 'KVR', 22.22, 'Hyd',
'Python']
```

Reading this Data Dynamically from Key Board

=>To read the data from Key Board, we have 2 pre-defined Functions. They are

- a) input()
 - b) 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 type values into other type of values by using Type casting Techniques.

=>Syntax: varname=input()

=>Here 'varname' is an object of type str.

=>input() is pre-defined function and reads any type of data from Key board.

Examples:

```
#mulex5.py
print("Enter two values:")
```

```
a=input()
b=input()
x=float(a)
y=float(b)
z=x*y
print("mul({},{})={}".format(x,y,z))
```

OUTPUT:

Enter two values:

12

30

mul(12.0,30.0)=360.0

b) input(Message):

=>This function is used for reading any type of data from Key Board in the form of str by giving User-Prompting Messages .

=>Syntax: varname=input(Message)

=>Here 'varname' is an object of type str.

=>input() is pre-defined function and reads any type of data from Key board.

=>Message represents User-Prompting Messages.

Examples:

#This program reads two values and multiply them

#mulex9.py

#convert str type into float type

x=float(input("Enter First Value:"))

y=float(input("Enter Second Value:"))

z=x*y

print("mul of {} and {}={}".format(x,y,z))

OUTPUT:

Enter First Value:30

Enter Second Value:20

mul of 30.0 and 20.0=600.0

#this program cal area of circle by accepting radius

r=float(input("Enter Radius:"))

ac=3.14*r**2

print("Area of Circle={}".format(ac))

OUTPUT:

Enter Radius:27

Area of Circle=2289.06

#This program reads two values and multiply them

#mulex3.py

print("Enter First Value:")

a=input() # a=12

print("Enter Second Value:")

b=input() # 4

```
print("Val of a={} and its type={}".format(a, type(a)))
print("Val of b={} and its type={}".format(b, type(b)))
#c=a*b # 12 * 4----error
#convert a and b into float
x1=float(a)
x2=float(b)
print("Val of x1={} and its type={}".format(x1, type(x1)))
print("Val of x2={} and its type={}".format(x2, type(x2)))
x3=x1*x2 #--valid
print("Mul ={} and its type={}".format(x3, type(x3)))
```

OUTPUT:

Enter First Value:

30

Enter Second Value:

20

Val of a=30 and its type=<class 'str'>
Val of b=20 and its type=<class 'str'>
Val of x1=30.0 and its type=<class 'float'>
Val of x2=20.0 and its type=<class 'float'>
Mul =600.0 and its type=<class 'float'>

```
#This program reads two values and multiply them
#mulex4.py
print("Enter First Value:")
a=input() # a=12
print("Enter Second Value:")
b=input() # 4
print("Val of a={} and its type={}".format(a, type(a)))
print("Val of b={} and its type={}".format(b, type(b)))
#c=a*b # 12 * 4----error
#convert a and b into float
x1=float(a)
x2=float(b)
print("Val of x1={} and its type={}".format(x1, type(x1)))
print("Val of x2={} and its type={}".format(x2, type(x2)))
x3=x1*x2 #--valid
print("Mul ={} and its type={}".format(x3, type(x3)))
```

OUTPUT:

Enter First Value:

30

Enter Second Value:

20

Val of a=30 and its type=<class 'str'>
Val of b=20 and its type=<class 'str'>
Val of x1=30.0 and its type=<class 'float'>
Val of x2=20.0 and its type=<class 'float'>
Mul =600.0 and its type=<class 'float'>

```
#This program reads two values and multiply them
#mulex5.py
print("Enter two values:")
a=input()
b=input()
x=float(a)
y=float(b)
z=x*y
print("mul({},{}]={}".format(x,y,z))
```

OUTPUT:

Enter two values:

12

30

mul(12.0,30.0)=360.0

```
#This program reads two values and multiply them
#mulex6.py
print("Enter two values:")
x=float(input())
y=float(input())
z=x*y
print("mul({},{}]={}".format(x,y,z))
```

OUTPUT:

Enter two values:

30

20

mul(30.0,20.0)=600.0

```
#This program reads two values and multiply them
#mulex7.py
print("Enter two values:")
print("mul={}".format(float(input())*float(input())))

```

OUTPUT:

Enter two values:

12

15

mul=180.0

```
#This program reads two values and multiply them
#mulex8.py
a=input("Enter First Value:")
b=input("Enter Second Value:")
#convert str type into float type
x=float(a)
y=float(b)
z=x*y
print("mul of {} and {}={}".format(x,y,z))
```

OUTPUT:

Enter First Value:30
Enter Second Value:15
mul of 30.0 and 15.0=450.0

```
#This program reads two values and multiply them
```

```
#mulex11.py
print("Mul={}".format(float(input("Enter First Value:"))*float(input("Enter Second Value:")) ))
```

OUTPUT:

Enter First Value:20
Enter Second Value:30
Mul=600.0

===== Operators in Python =====

=====
ex:
=====

- =>Purpose of Operators
 - =>Expression
 - =>Types of Operators
 - =>Programming Examples on Every Operator type.
-

===== Operators in Python =====

- =>An operator is a symbol.
- =>The purpose of Operators is that to perform certain operation on the given data.
- =>If two or more Variables / Objects are connected with an operator then it is called Expression.
- =>In Python Programming, we have 7 types of Operators. They are

1. Arithmetic Operators

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

Note:- Unary Increment/Decrement Operator ---C,CPP,JAVA, .NET ++ --

Not There in Python

? : Ternary Operator --not there in Python

1. Arithmetic Operators

=>The purpose of Arithmetic Operators is that "To Perform Arithmetic Operations such as Addition, subtraction, 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

SINO	Symbol	Meaning	Example a=10 b=3
1.	+	Addition	print(a+b)----->13
2.	-	Subtraction	print(a-b)----->7
3.	*	Multiplication	print(a*b)-----> 30
4.	/	Division	print(a/b)-----> 3.33333333333335
5.	//	Floor Division	print(a//b)-----> 3
6.	%	Modulo Division	print(a%b)-----> 1
7.	**	Exponentiation	print(a**b)----->1000

#This Program demonstrates the arithmetic Operators

```
#aop.py
a=int(input("Enter Value of a:"))
b=int(input("Enter Value of b:"))
print("-"*50)
print("Arithmetic Operation Results")
print("-"*50)
print("{}+{}={}".format(a,b,a+b))
print("{}-{}={}".format(a,b,a-b))
```

```
print("{}*{}={}".format(a,b,a*b))
print("{}//{}={}".format(a,b,a//b))
print("{}%{}={}".format(a,b,a%b))
print("{}**{}={}".format(a,b,a**b))
print("-"*50)
```

OUTPUT:

Enter Value of a:20

Enter Value of b:7

Arithmetic Operation Results

20+7=27

20-7=13

20*7=140

20/7=2.857142857142857

20//7=2

20%7=6

20**7=1280000000

EXAMPLE:2

```
#This program calculates 'a' to the power of 'm'
#powerex1.py
a=float(input("Enter Base:"))
m=float(input("Enter Power"))
res=a**m
print("pow({},{})={}".format(a,m,res))
```

OUTPUT:

Enter Base:9

Enter Power3

pow(9.0,3.0)=729.0

EXAMPLE:

```
#This program calculates square root of given number
#sqrtex1.py
n=float(input("Enter a number:"))
res=n**0.5 # OR res= n** (1/2)
print("sqrt({})={}".format(n,res))
print("-----OR-----")
```

```
print("sqrt({})={}".format(n,round(res,2)))
print("-----OR-----")
print("sqrt({})={:.2f} {}".format(n,res))
```

OUTPUT:

Enter a number:30
sqrt(30.0)=5.477225575051661
-----OR-----
sqrt(30.0)=5.48
-----OR-----
sqrt(30.000000)=5.48

2. Assignment Operator

=>The symbol for assignment operator is =
=>The purpose of Assignment Operator is that "To transfer RHS value(s) to LHS Variable(s)"
=>We can use the Assignment Operator in two ways. They are
 1) Single Line Assignment Operator
 2) Multi Line Assignment Operator

1) Single Line Assignment Operator

=>This operator transfer Single value from RHS to Single Variable of LHS.

=>Syntax: varname=value

Examples:

```
>>> a=123
>>> b=100
>>> c=a+b
>>> print(a,b,c)---- 123 100 223
```

2) Multi Line Assignment Operator

=>This operator transfer Multiple values from RHS to Multiple Variables of LHS.

=>Syntax: var1,var2.....var-n=val1,val2,...val-n

=>here the values of val1,val2.....val-n are assigned to var1,var2...var-n respectively.

Examples:

```
>>> a,b=10,20
>>> sum,sub,mul=a+b,a-b,a*b
>>> print(a,b)-----10 20
>>> print(sum,sub,mul)----30 -10 200
>>> sno,sname,marks=100,"Rossum",44.44
>>> print(sno,sname,marks)---100 Rossum 44.44
```

```
#This program swaps two values
#swap.py
a=input("Enter Value of a:")
b=input("Enter Value of b:")
print("-"*50)
print("Original Value of a:{}".format(a))
print("Original Value of b:{}".format(b))
print("-"*50)
#swapping logic
a,b=b,a
print("Swapped Value of a:{}".format(a))
print("Swapped Value of b:{}".format(b))
print("-"*50)
```

PROGRAM:

```
#To perform all Arithmetic Operation using Multiline Assignment Operators
#multiAsgn.py
a=float(input("Enter first number:"))
b=float(input("Enter second value:"))

print("_"*50)
add,sub,mul,div,floor_div,modulo,pow=a+b,a-b,a*b,a/b,a//b,a%b,a**b

print("{}+{}={}\n{}={}*{}={}\n{}={}//{}={}\n{}={}%{}={}\n{}={}**{}={}".
format(a,b,add,a,b,sub,a,b,mul,a,
b,div,a,b,floor_div,a,b,modulo,a,b,pow))
```

OUTPUT:

Enter first number:10

Enter second value:3

10.0+3.0=13.0

10.0-3.0=7.0

10.0*3.0=30.0

10.0/3.0=3.333333333333335

10.0//3.0=3.0

10.0%3.0=1.0

10.03.0=1000.0**

3. Relational Operators

=>The purpose of Relational Operators is that "To compare two values "

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

=>The result of Relational Expression is either True or False and they helps us for taking decisions.

=>Relational Operators are given in the following Table:

SLNO	Symbol	Meaning	Example a=10 b=20,c=10
1.	>	greater than	print(a>b)-----False print(b>c)-----True
2.	<	Less than	print(a<b)-----True print(a<c)-----False
3.	==	Equality	print(a==b)-----False print(a==c)-----True
4.	!=	not equal to	print(a!=b)-----True print(a!=c)-----False
5.	>=	greater than or equal to	print(a>=b)-----False print(a>=c)-----True
6.	<=	Less than or equal to	print(a<=-2)-----False print(a<=c)-----True

#This program swaps two values

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

OUTPUT:

Enter Value of a:20
Enter Value of b:30

Original Value of a:20

Original Value of b:30

**Swapoped Value of a:30
Swapoped Value of b:20**

```
#This program demonstares Relational Operators
#rop.py
a=int(input("Enter val of a:"))
b=int(input("Enter val of b:"))
print("*"*60)
print("Results of Relational Operators")
print("*"*60)
print("{}>{}={}".format(a,b,a>b))
print("{}<{}={}".format(a,b,a<b))
print("{}=={}={}".format(a,b,a==b))
print("{}!={}={}".format(a,b,a!=b))
print("{}>={{}={}}".format(a,b,a>=b))
print("{}<={{}={}}".format(a,b,a<=b))
print("*"*60)
```

OUTPUT:

Enter val of a:30

Enter val of b:20

Results of Relational Operators

30>20=True

30<20=False

30==20=False

30!=20=True

30>=20=True

30<=20=False

=====

4. Logical Operators

=>The purpose of Logical Operators is that "To combine two or more relational expressions"
=>If two or more relational expressions are connected with logical Operators then it is called Logical Expression.

=>The result of Logical expressions is either True or False.

=>Logical Expression also called Compund Condition.

=>We have 3 types of Logical Operators. They are given in the following table.

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

1) and operator:

=>The functionality of 'and' operator is shown in the following Truth Table.

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

Examples:

```
>>> print(100>20 and 20>10)-----True
>>> print(100<20 and 20>10)-----False
>>> print(100!=20 and 20!=20)-----False
>>> print(20!=20 and 20!=20)-----False
```

2) or operator:

=>The functionality of 'or' operator is shown in the following Truth Table.

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

Examples:

```
>>> print(100!=20 or 20!=20)-----True
>>> print(100<=20 or 2!=20)-----True
>>> print(100>=20 or -12!=-12)-----True
>>> print(100==20 or -12!=-12)-----False
```

Note:- The Short Circuit Evaluation in the case of 'or' operator is that " If or operator is connected with n-relational expressions and if First relational Expression is True then remaining relational expressions will not be evaluated and result is considered as True".

Note:- The Short Circuit Evaluation in the case of 'and' operator is that " If 'and' operator is connected with n-relational expressions and if First relational Expression is False then remaining relational expressions will not be evaluated and result is considered as False".

3) not operator:-

=>This operator gives opposite result of Boolean Value.

=>The functionality of 'not' operator is shown in the following Truth Table.

RelExpr1	not RelExpr1
True	False
False	True

Examples:

```
>>> print( 10<=2 and 1000>=200 )-----False
>>> print( not(10<=2 and 1000>=200 ) )-----True
>>> print(not( not(10<=2 and 1000>=200 ) ))---False
>>> print(10>2)---True
>>> print(not 10>2)---False
>>> print(10==2)---False
>>> print(not 10==2)---True
```

5. Bitwise Operators (Most Imp)

=>Bitwise Operators are Applicable on Integer data only but not applicable on floating point values because they are varying precision.

=>The purpose of Bitwise Operators is that "Converting Integer data internally into Binary Format and Performs Binary Operations in the form Bit by Bit and final result always displayed in the form decimal number system".

=>Since the operations are carrying out in the form of Bit by Bit and Hence these operators are named as Bitwise Operators.

=>In Python Programming, we have

1. Bitwise Left Shift Operator (<<)
 2. Bitwise Right Shift Operator (>>)
 3. Bitwise 'and' Operator (&)
 4. Bitwise 'or' Operator (|)
 5. Bitwise complement Operator (~)
 6. Bitwise 'xor' Operator (^)
-

1. Bitwise Left Shift Operator (<<):

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

=>The concept of bitwise left shift operator says that " Shift specified number of bits towards left by adding number of zeros at right side (no. of zeros==no. of specified number of bits)

Bitwise Left Shift Operator

Examples:

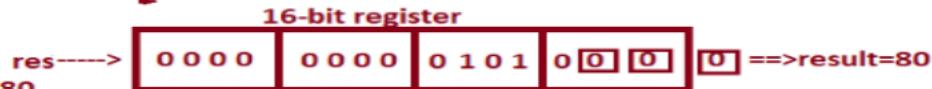
```
>>>a=10
```



```
>>>res=a<<3
```



```
>>>print(res)-->80
```



FORMULA:

$$\text{Res} = \text{GivenData} \ll \text{No.of bits} \Rightarrow \text{GivenData} \times 2^{\text{No.of Bits}}$$

Ex:- res=10<<3----- 10×2^3
= 10 x 8---->80

res=10<<4----- 10×2^4
= 10x16--->160

res=23<<5

Examples:

```
>>> a=10
```

```
>>> res=a<<3
```

```
>>> print(res)-----80
```

```
>>> print(24<<2)-----96
```

```
>>> print(2<<2)-----8
```

```
>>> print(12<<3)-----96
```

```
>>> print(5<<4)-----80
```

2. Bitwise Right Shift Operator (>>):

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

=>The concept of bitwise right shift operator says that " Shift specified number of bits towards right by adding number of zeros at left side (no. of zeros==no. of specified number of bits)

Bitwise Right Shift Operator

Example:

```
>>a=10      a----> 0 0 0 0 | 0 0 0 0 | 0 0 0 0 | 1 0 1 0
                                         16-bit register
>>>res=a>>3   res----> 0 0 0 0 | 0 0 0 0 | 0 0 0 0 | 1 0 1 0
                                         16-bit register
                                         16-bit register
>>print(res)-->1    a--> 0 | 0 0 | 0 0 0 | 0 0 0 | 0 0 0 1  --->result= 1
```

Formula:

$$\text{result} = \frac{\text{Given Data}}{\text{No.of Bits}}$$

$$\text{Example: } \text{res}=10>>3 \Rightarrow \frac{10}{3} = 10//8 \Rightarrow 1$$

$$\text{Example:- print}(23>>2) \Rightarrow 23 // 4 \Rightarrow 5$$

Examples:

```
>>> c=a>>3
>>> print(c)---->1
>>> print(12>>2)---->3
>>> print(12>>3)---->1
>>> print(12>>4)---->0
>>> print(36>>4)---->2
```

3. Bitwise 'and' Operator (&):

=>Syntax:- varname = Value1 & Value2

=>The Functionality of Bitwise 'and' Operator is expressed in the following truth table.

Input1	Input2	Input1 & Input2
0	1	0
1	0	0
0	0	0
1	1	1

Example1:

```
>>>a=10-----1010
>>>b=15-----1111
>>>c=a&b-----1010---->result is 10
>>>print(c)--->10
```

Examples:

```
>>> print(3&4)-----0
>>> print(10&3)-----2
```

Special Case:

```
>>>s1={10,20,30}
>>>s2={40,30,50}
>>>s3=s1.intersection(s2)
>>>print(s3)-----{30}
>>>s4=s1&s2
>>>print(s4)-----{30}
```

4. Bitwise 'or' Operator (|):

=>Syntax:- varname = Value1 | Value2

=>The Functionality of Bitwise 'or' Operator is expressed in the following truth table.

Input1	Input2	Input1 Input2
0	1	1
1	0	1
0	0	0
1	1	1

Examples:

```
>>> a=10-----> 1010
>>> b=7-----> 0111
>>> c=a|b-----> 1111-----> result is 15
>>> print(c)-----15
>>> print(3|4)-----7
```

5. Bitwise complement Operator (~):

Syntax:- varname= ~Value

=>This operator obtains complement of the given number

Examples:

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

```
>>>a=15
>>>b=~a-----> ~(1111+1)
                  -(1111
                  + 0001)= - 10000----> result is -16
```

```
>>> a=13
>>> b=~a
>>> print(b)----- -14
>>> a=345
>>> print(~a)----- -346
```

6. Bitwise 'xor' Operator (^):

=>Syntax:- varname = Value1 ^ Value2

=>The Functionality of Bitwise 'xor' Operator is expressed in the following truth table.

Input1	Input2	Input1 ^ Input2
0	1	1
1	0	1
0	0	0
1	1	0

Examples:

```
>>>a=3-----0 0 1 1  
>>>b=4-----0 1 0 0  
>>>c=a^b-----0 1 1 1-----Result is 7  
>>>print(c)---7  
>>> print(15^10)---5
```

Special Case:

```
>>>s1={10,20,30,40}  
>>>s2={10,40,50,60}  
>>>s3=s1.symmetric_difference(s2)  
>>>print(s3)-----{50, 20, 60, 30}  
>>>  
>>>s4=s1^s2  
>>>print(s4)-----{50, 20, 60, 30}
```

#Swapping of values using xor (^) operator

```
a=int(input("Enter a value of a:"))  
b=int(input("Enter a value of b:"))  
print("-"*50)  
print("Values of a and b after swapping")  
print("-"*50)  
a=a^b  
b=a^b  
a=a^b  
print("a:{}\nb:{}".format(a,b))  
print("-"*50)
```

OUTPUT:

Enter a value of a:30
Enter a value of b:40

Values of a and b after swapping

a:40

b:30

=====x=====

6. Membership Operators

=>The purpose of Membership Operators is that " To check the specified value in any Iterable object".

=>An Iterable object is one, which contains multiple elements (Sequence type, list type, set type and dict type).

=>In Python, We have two type of Membership Operators. They are

- a) in
 - b) not in
-

a) in:

Syntax:- varname= Value in Iterable_object

=>If the "Value" present in Iterable_object then It returns True otherwise it returns False.

=>Here "varname" contains True or False and whose type is bool.

b) not in:

Syntax:- varname= Value not in Iterable_object

=>If the "Value" not present in Iterable_object then It returns True otherwise it returns False.

=>Here "varname" contains True or False and whose type is bool.

Examples:

```
>>> s="PYTHON"
>>> "P" in s-----True
>>> "P" not in s----False
>>> "T" not in s----False
>>> "T" in s-----True
>>> "K" in s-----False
>>> "K" not in s----True
>>> "TH" in s-----True
>>> "TH" not in s----False
>>> "THON" not in s----False
>>> "THON" in s-----True
>>> s="PYTHON"-----
>>> "PTO" in s-----False
>>> "PYT" in s-----True
>>> "PYON" in s-----False
>>> "PYON" not in s----True
>>> "HON" not in s----False
>>> "HON" in s-----True
>>> "NOH" in s-----False
```

```
>>> "NOH" not in s-----True
>>> "NOH" in s[::-1]-----True

>>> lst=[10,20,30,40,"python"]
>>> 10 not in lst-----False
>>> 10 in lst-----True
>>> "python" in lst-----True
>>> "python" not in lst-----False
>>> "pyt" in lst-----False
>>> print(lst[4])-----python
>>> "pyt" in lst[4]-----True
>>> "pyt" not in lst[4]-----False
>>> "pt" not in lst[4]-----True
>>> "noh" not in lst[4]-----True
>>> "noh" in lst[4]-----False
>>> "noh" in lst[4][::-1]-----True
>>> "nh" not in lst[4][::-1]-----True
-----X-----
```

7. Identity Operators

=>The purpose of Identity Operators is that "To compare the Memory Address of objects".

=>To get Memory Address of an object / variable, we use id().

=>In Python, we have two Identity Operators. They are

- 1) is
 - 2) is not
-

a) is:

Syntax:- **varname= obj1 is obj2**

=>This operator returns True provided obj1 and obj2 contains Same Address otherwise it returns False.

b) is not:

Syntax:- **varname= obj1 is not obj2**

=>This operator returns True provided obj1 and obj2 contains Different Address otherwise it returns False.

Examples:

```
>>> a=None
>>> b=None
>>> print(id(a))-----140716433311736
>>> print(id(b))-----140716433311736-
>>> a is b-----True
>>> a is not b-----False
```

```
>>> d1={10:"Apple",20:"Kiwis"}
>>> d2={10:"Apple",20:"Kiwis"}
```

```
>>> print(id(d1))-----2896803430144
>>> print(id(d2))-----2896803363776
>>> d1 is d2-----False
>>> d1 is not d2-----True
```

```
>>> s1={10,20,30}
>>> s2={10,20,30}
>>> print(id(s1))-----2896803638944
>>> print(id(s2))-----2896803637152
>>> s1 is s2-----False
>>> s1 is not s2-----True
>>> fs1=frozenset(s1)
>>> fs2=frozenset(s2)
>>> print(id(fs1))-----2896803636928
>>> print(id(fs2))-----2896803638496
>>> fs1 is fs2-----False
>>> fs1 is not fs2-----True
```

```
>>> l1=[10,"Rossum","Python"]
>>> l2=[10,"Rossum","Python"]
>>> print(id(l1))-----2896803430016
>>> print(id(l2))-----2896803473856
>>> l1 is l2-----False
>>> l1 is not l2-----True
>>> t1=tuple(l1)
>>> t2=tuple(l2)
>>> print(id(t1))-----2896803468800
>>> print(id(t2))-----2896803470528
>>> t1 is t2-----False
>>> t1 is not t2-----True
```

```
>>> r1=range(10,20)
>>> r2=range(10,20)
>>> print(id(r1))-----2896803585632
>>> print(id(r2))-----2896803585824
>>> r1 is r2-----False
>>> r1 is not r2-----True
>>> b1=bytes([10,20,30,40])
>>> b2=bytes([10,20,30,40])
>>> print(id(b1))-----2896803584624
>>> print(id(b2))-----2896803585776
>>> b1 is b2-----False
>>> b1 is not b2-----True
>>> ba1=bytearray([10,20,30,40])
>>> ba2=bytearray([10,20,30,40])
>>> print(id(ba1))-----2896803726768
>>> print(id(ba2))-----2896803726960
>>> ba1 is ba2-----False
>>> ba1 is not ba2-----True
```

```
>>> s1="INDIA"
>>> s2="INDIA"
>>> print(id(s1))-----2896803726640
>>> print(id(s2))-----2896803726640
>>> s1 is s2-----True
>>> s1 is not s2-----False
>>> s3="python"
>>> print(id(s3))-----2896803725360
>>> s1 is s3-----False
>>> s1 is not s3-----True
```

```
>>> a=2+3j
>>> b=2+3j
>>> print(id(a))-----2896799652016
>>> print(id(b))-----2896799651888
>>> a is b-----False
>>> a is not b-----True
```

```
>>> a=12.3
>>> b=12.3
>>> print(id(a))-----2896799647728
>>> print(id(b))-----2896799648336
>>> a is b-----False
>>> a is not b-----True
```

```
>>> a=True
>>> b=True
>>> print(id(a))
140716433259368
>>> print(id(b))
140716433259368
>>> a is b
True
>>> a is not b
False
```

```
>>> a=100
>>> b=100
>>> print(id(a))
2896798616912
>>> print(id(b))
2896798616912
>>> a is b-----→ True
>>> a is not b-----→ False
>>> a=257
>>> b=257
>>> print(id(a))-----→ 2896799652016
>>> print(id(b))-----→ 2896799652176
>>> a is b-----→ False
```

```
>>> a is not b-----→True
>>> a=300
>>> b=300
>>> print(id(a))-----→2896799652208
>>> print(id(b))-----→ 2896799652016
>>> a is b-----→ False
>>> a is not b-----→True
>>> a=256
>>> b=256
>>> print(id(a))-----→2896798621904
>>> print(id(b))-----→2896798621904
>>> a is b-----→True
>>> a is not b-----→False
>>> a=-4
>>> b=-4
>>> print(id(a))-----→2896798613584
>>> print(id(b)) -----→2896798613584
>>> a is b-----→True
>>> a is not b-----→False
>>> a=-5
>>> b=-5
>>> print(id(a)) -----→2896798613552
>>> print(id(b)) -----→2896798613552
>>> a is b-----→True
>>> a is not b-----→False
>>> a=-6
>>> b=-6
>>> print(id(a)) -----→2896799652144
>>> print(id(b)) -----→2896799652080
>>> a is b-----→False
>>> a is not b-----→True
=====
>>> a,b=300,300
>>> print(id(a))-----2896799652080
>>> print(id(b))-----2896799652080
>>> a is b-----True
>>> a is not b-----False
>>> a,b=-10,-10-----
>>> a is not b-----False
>>> a is b-----True
=====X=====
```

Flow Control Statements in Python

(OR)

Control Structures in Python--8 days

=>The purpose of Flow Control Statements in Python is that "To perform Certain Operation either Once or Perform the operation Repeatedly for finite number of times until Condition is False".

=>In Python Programming, Flow Control Statements are classified into 3 types. They are

1. Conditional / Selection / Branching Statements
 2. Looping / Iterative / Repetitive Statements
 3. MiSc Flow Control Statements (break,continue,pass)
-

1. Conditional / Selection / Branching Statements

=>The purpose of this operation is that to perform certain operation only once depends on condition evaluation.

=>In other words perform certain X-operation when condition is TRUE or perform Y-operation when condition is FALSE.

=>in Python we have 4 types of Conditional / Selection / Branching Statements.

They are:

- 1) simple if statement
 - 2) if.....else statement
 - 3) if...elif...else statement
 - 4) match...case statement
-

1) simple if statement:

Syntax:

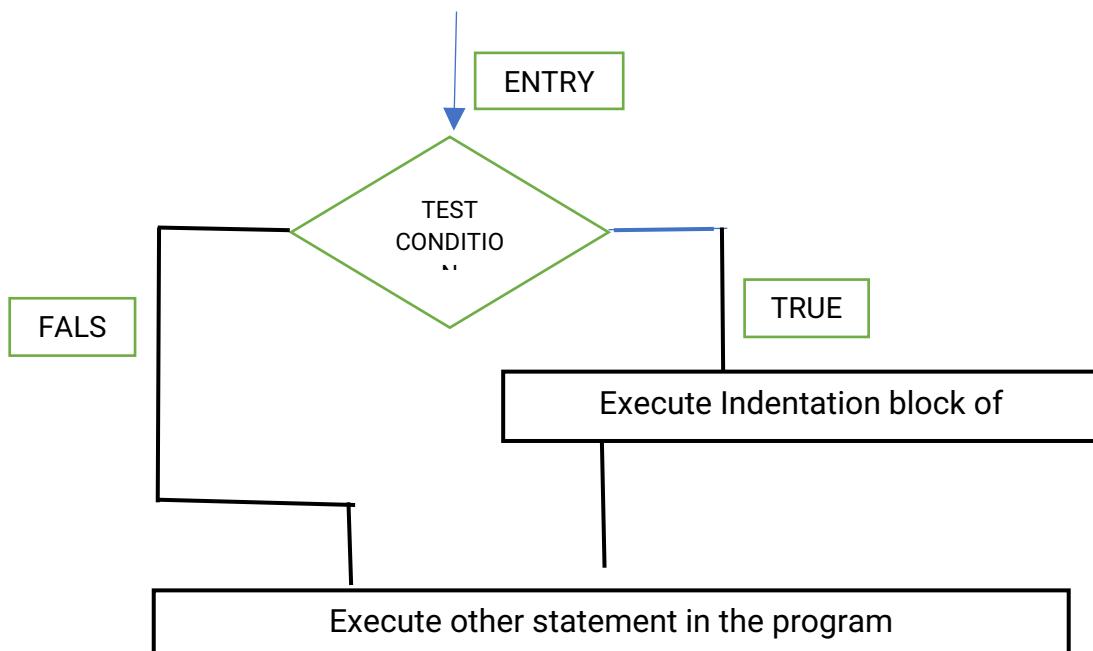
If(TEST Condition): ➔(that : is called Indentation Symbol)

Statement1
Statement2
.....
Statement-n

➔(that block is called Indentation Block)

Other statement in python program

Flow chart for simple if statement:



Explanation:

=>Here if is a keyword.

=>If the TEST CONDITION is TRUE then PVM executes indentation block of statements and later executes other statements in the program.

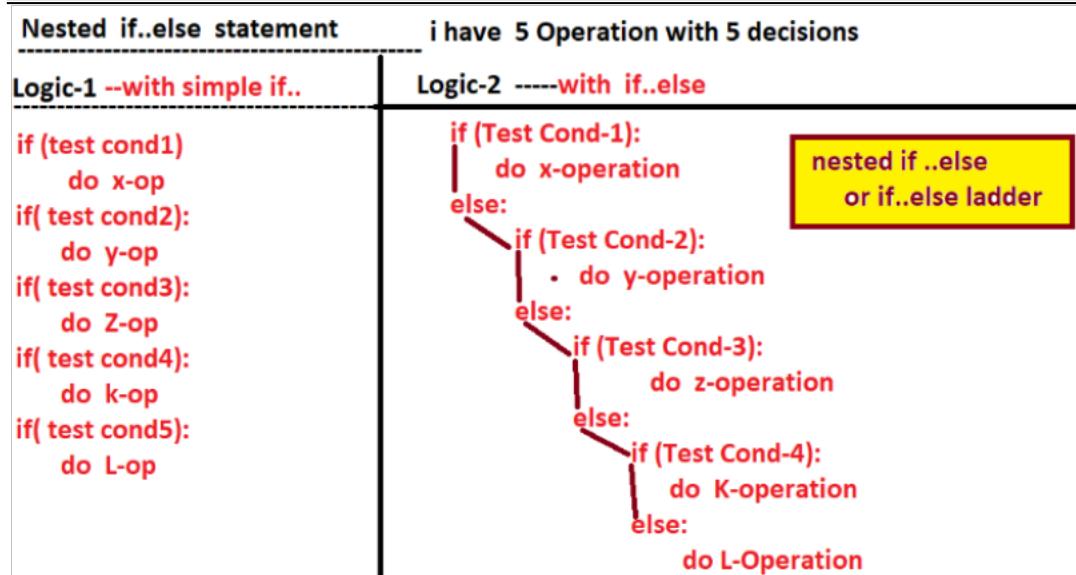
=>If the TEST CONDITION is FALSE then PVM executes other statements in the program without executing indentation block of statements .

PROGRAMS:

```
#moviee.py
tkt=input("D u have a ticket(yes/no):")
if(tkt.lower()=="yes"):
    print("Enter into theater")
    print("watch movee")
    print("Enjoy..")
print("\nGoto Home and Read PYTHON NOTES")
```

```
#posnegzero.py
n=int(input("Enter a number:")) # n-- 10
if(n>0):
    print("{} is Possitive".format(n))
if(n<0):
    print("{} is Negative".format(n))
if(n==0):
    print("{} is Zero".format(n))
print("Program completed")
```

```
#evenodddno
n=int(input("Enter a number:"))
if(n%2==0):
    print("{} is even.".format(n))
if(n%2!=0):
    print("{} is odd.".format(n))
```



2. ifelse statement

Syntax:

If(TEST CONDITION):

 Statement-1

 Statement-2

 Statement-ns

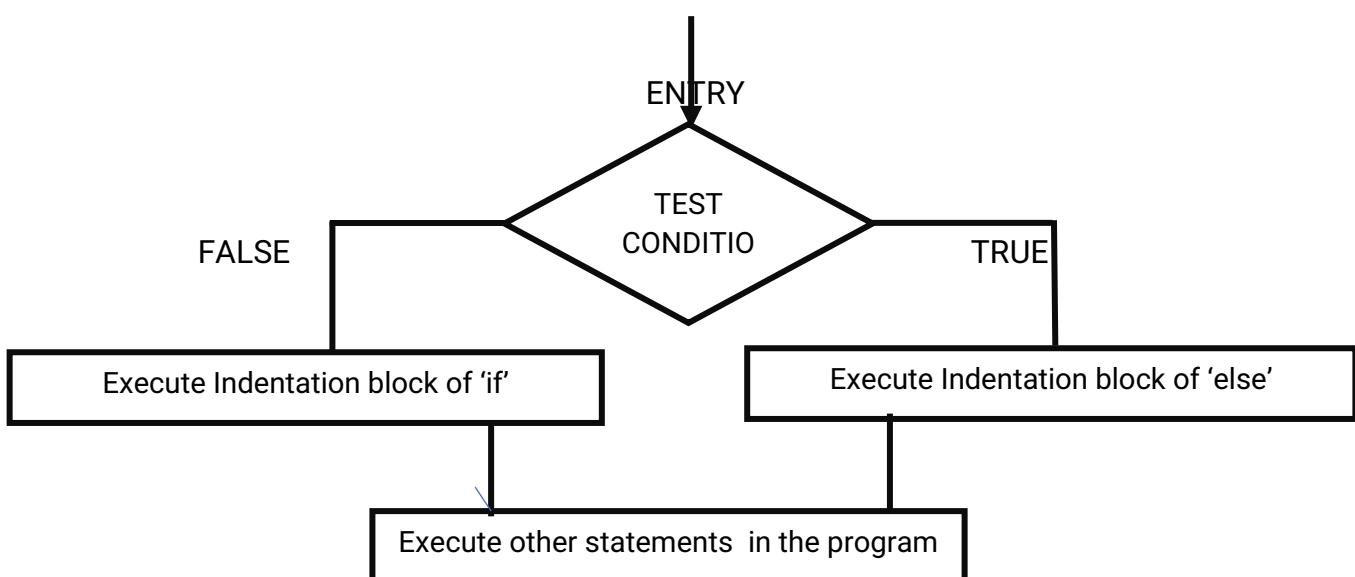
else:

 Statement-11

 Statement-22

 Statement-nn

Other statements in the program



Explanation of if....else Statement:

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

=>If the TEST CONDITION is TRUE then PVM executes Indentation block of 'if' Statement and later execute other statements in the program(Without executing indentation block of 'else' statement).

=>If the TEST CONDITION is FALSE then PVM executes Indentation block of 'else' Statement and later execute other statements in the program(Without executing indentation block of 'if' statement).

Hence any point of time PVM execute either Indentation block of 'if' statement or Indentation block of 'else' statement and later executes other statements in the program.

EXAMPLE1:

```
#WPP which will accept a numerical number and decide whether it is even or odd.  
#evenoddex1.py  
n=int(input("Enter a number:"))  
if(n%2==0):  
    print("{} is Even.".format(n))  
else:  
    print("{} is Odd.".format(n))
```

OUTPUT:

Enter a number:975

975 is Odd.

EXAMPLE-2:

```
#write a python program which will accept any numerical integer value and decide  
whether it is positive, negative or zero.  
#posnegzeroex1.py  
n=int(input("Enter a number:"))  
if(n==0):  
    print("{} is Zero.".format(n))  
else:  
    if(n>0):  
        print("{} is Positive.".format(n))  
    else:  
        print("{} is negative.".format(n))
```

OUTPUT:

Enter a number:2122

2122 is Positive.

EXAMPLE-3

```
#WPP which will accept an alphabet from keyboard and decide whether it is vowel or  
consonant.  
#vowelcon1.py  
a=input("Enter an alphabet :")  
if(a.lower() in ["a","e","i","o","u"]):  
    print("{} is Vowel.".format(a))  
else:  
    print("{} is Consonant.".format(a))
```

OUTPUT:

Enter an alphabet:R

R is Consonant.

EXAMPLE-3

#WPP which will accept the employee no. ,employee name and employee salary and calculate the net salary.

```
#emppayslip.py
eno=int(input("Enter employee number:"))
ename=input("Enter employee name:")
basicssal=float(input("enter basic salary of employee:"))
if(basicssal>=10000):
    da=(20/100)*basicssal
    ta=(15/100)*basicssal
    hra=(12/100)*basicssal
    cca=(2/100)*basicssal
    ma=(2/100)*basicssal
    gpf=(2/100)*basicssal#deduction
    lic=(1/100)*basicssal#deduction
else:
    da=(25/100)*basicssal
    ta=(20/100)*basicssal
    hra=(16/100)*basicssal
    cca=(3/100)*basicssal
    ma=(2/100)*basicssal
    gpf=(2/100)*basicssal#deduction
    lic=(1/100)*basicssal#deduction
netsal=(basicssal+da+ta+hra+cca+ma)-(gpf+lic)
# display the pay slip
print("-"*60)
print("\tEmployee pay slip for the month of feb-2022")
print("-"*60)
print("\tEmployee number:{}".format(eno))
print("\nEmployee name:{}".format(ename))
print("\tEmployee basic salary:{}".format(basicssal))
print("\tEmployee da:{}".format(da))
print("\tEmployee ta:{}".format(ta))
print("\tEmployee hra:{}".format(hra))
print("\tEmployee cca:{}".format(cca))
print("\tEmployee ma:{}".format(ma))
print("\tEmployee gpf:{}".format(gpf))
print("\tEmployee lic:{}".format(lic))
print("-"*60)
print("\tEmployee net salary:{}".format(netsal))
print("-"*60)
```

OUTPUT:

Enter employee number:123

Enter employee name:Ritesh

enter basic salary of employee:35000

Employee pay slip for the month of feb-2022

Employee number:123

Employee name:Ritesh

Employee basic salary:35000.0

Employee da:7000.0

Employee ta:5250.0

Employee hra:4200.0

Employee cca:700.0

Employee ma:700.0

Employee gpf:700.0

Employee lic:350.0

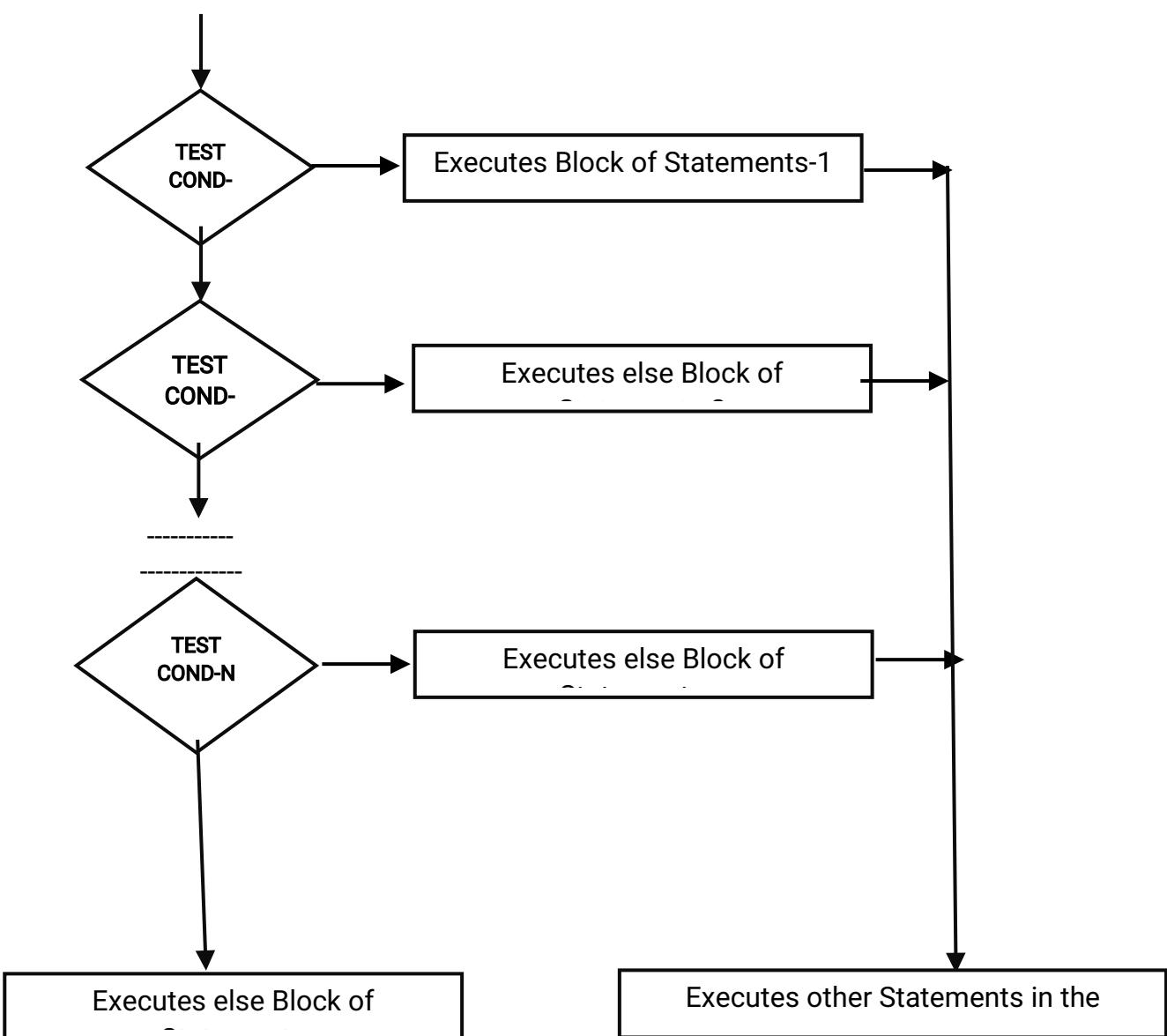
Employee net salary:51800.0

if....elif....else Statement:

Syntax:

```
If(Test Condition-1):
    Execute Block of Statement-1
elif(Test Condition-2):
    Execute Block of Statement-2
elif(Test Condition-3):
    Execute Block of Statement-3
-----
-----
elif(Test Condition-n):
    Execute Block of Statement-n
else:
    Execute 'else' Block of Statement.
```

FLOW CHART:





Explanation for if ...elif...else statement :

=>Here if the test condition -1 is true PVM execute block of statements -1 and also executes other statements in the program.

=>If the test condition -1 is false then evaluate test condition -2 and if it is true the PVM executes block of statement -2 and also execute other statements in program. This process will performed until all test condition are evaluated.

=>If all test condition are false then execute else block of statements which are writing under the else block.

=>Writing else block is optional .

EXAMPLE:

#WPPwhich will accepts three integer value and find biggest among them.

```
#bigex1.py
a=int(input("Enter first number:"))
b=int(input("Enter second value:"))
c=int(input("enter third value:"))
if(a>b)and(a>c):
    print("max({},{},{}):{}".format(a,b,c,a))
elif(b>a)and(b>c):
    print("max({},{},{}):{}".format(a,b,c,b))
elif(c>a)and(c>b):
    print("max({},{},{}):{}".format(a,b,c,a))
elif(a==b)and(b==c)and(a==c):
    print("All values are equal.")
```

OUTPUT:

Enter first number:67

Enter second value:89

enter third value:45

max(67,89,45):89

4. match case statement

=>match....case is one of the new feature in Python 3.10 onwards
=>match case statement is always recommended to deal with pre-defined / designed decisions / Operations.

=>Syntax:-

```
match ( Choice Expression ):
    case Label1:
        Execute Block of statements-1
    case Label-2:
        Execute Block of statements-2
    case Label-3:
        Execute Block of statements-3
    -----
    -----
    case Label-n:
        Execute Block of statements-n
    case _ :
        execute default case block
```

Explanation:

=>Here 'match' 'case ' are keywords.
=>"Choice Expression" can be either Integer, Char , String, bool.
=>Label1,Label2....label-n can be Integer, Char , String, bool.
=>If the value of Choice Expression is matching with Label-1 then PVM executes corresponding Block of Statement-1.
=>If the value of Choice Expression is not matching with Label-1 and it is matching with Label-2 then PVM executes corresponding Block of Statement-2 and Hence This comparison process will be continued with all Case Labels and they matches PVM executes Corresponding Block of statements.
=>If the value of Choice expression is not matching with any case labels then PVM executes default case block statements which are written under case_:
=>Writing the default case block with case_:_ is optional

EXAMPLE-1:

#Write a python program which calculate area of different figures.

```
#matcharea.py
print("-"*50)
print("\tArea of different figure")
print("-"*50)
print("\t1.Circle")
```

```
print("\t2.Rectangle")
print("\t3.Square")
print("\t4.Triangle")
print("\t5.Exit")
print("-"*50)
ch=int(input("Enter Your choice:"))
print("-"*50)
match(ch):
    case 1:
        r=float(input("Enter radius of circle:"))
        print("Area of circle:{}".format((22/7)*r*r))
    case 2:
        l,b=float(input("Enter Length of rectangle:")),float(input("Enter breadth of rectangle:"))
        print("Area of Rectangle:{}".format(l*b))
    case 3:
        s=float(input("Enter side of a square:"))
        print("Area of Square:{}".format(s*s))
    case 4:
        b,h=float(input("Enter base of a Triangle:")),float(input("Enter height of a Triangle:"))
        print("Area of Triangle:{}".format(0.5*b*h))
    case 5:
        print("\n Thanks for using program!")
        exit()
    case _:
        print("Your choice is wrong !")
print("-"*50)
```

OUTPUT:

Area of different figure

- 1.Circle**
 - 2.Rectangle**
 - 3.Square**
 - 4.Triangle**
 - 5.Exit**
-

Enter Your choice:3

Enter side of a square:9
Area of Square:81.0

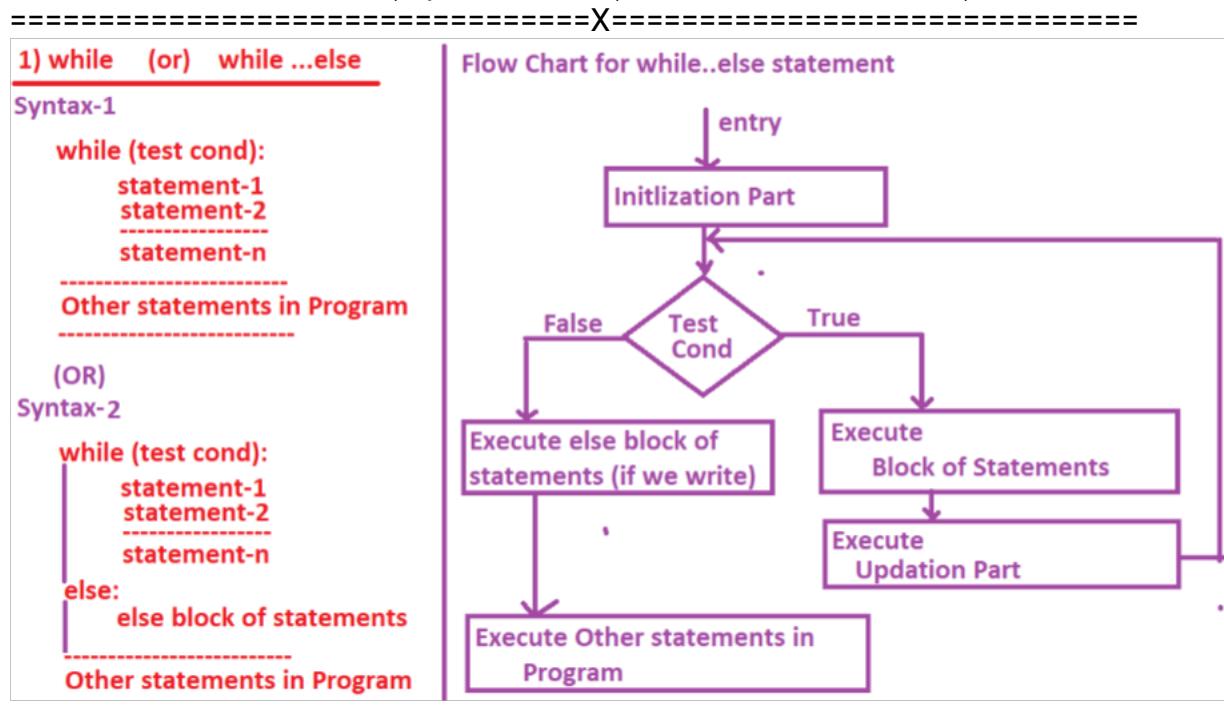
===== 2. Looping (or) Iterative (or) Repetitive Statements =====

=>The purpose of Looping statements is that "To Perform certain Operation repeatedly for finite number of times until Test condition becomes False".
=>In Python Programming, we have 2 types of Looping statements. They are

- a) while loop (or) while....else Loop
- 2) for loop (or) for....else Loop

=>At time of dealing with loop of Python, we must ensure that there must exists 3 points.

- 1)Initialization Part
- 2)Conditional Part
- 3)Updation Part (Increment or Decrement)



Explanation for while loop:

=>Here 'while' and 'else' are key words.

=>if the Test condition is true then execute block of statements and once again evaluate Test condition and if it is TRUE then again execute block of statements and this process will be repeated for finite number of times until Test condition becomes FALSE.

=>If Test condition becomes FALSE then executes else block of statements (If we write) and also execute other statements in the program.

=>Writing else block is optional.

EXAMPLE-1

```
#Write a python program which will display 1 to n numbers where n is positive.  
#displayno1.py  
n=int(input("Enter how many number you want to display:"))  
if(n<=0):  
    print(" is invalid input.".format(n))  
else:  
    i=1  
    print("Range of value within :{}".format(n))  
    while(i<=n):  
        print("value of i:{}.".format(i))  
        i=i+1  
    else:  
        print("I am from else part of while loop.")  
print("I am other part of program.")
```

EXAMPLE-2

```
#Write a python program which will accept any numerical integer value and display  
even number with in it  
#whileeven.py  
n=int(input("Enter any number to print even no:"))  
if(n<0):  
    print("Invalid Input.")  
else:  
    i=2  
    print("*"*50)  
    print("Even numbers are")  
    print("*"*50)  
    while(i<=n):  
        print("\t{}".format(i))  
        i=i+2  
    else:  
        print("*"*50)
```

EXAMPLE-3

```
#Write a python program which will accept any numerical integer value and display  
odd number with in it  
#whileodd.py
```

```
n=int(input("Enter any number to print odd no:"))  
if(n<0):  
    print("Invalid Input.")  
else:  
    i=1
```

```
print("*"*50)
print("Even numbers are")
print("*"*50)
while(i<=n):
    print("\t{}".format(i))
    i=i+2
else:
    print("*"*50)
```

EXAMPLE:-4

#Write a python program which will generate multiplication table for given positive integer value.

```
#whilemul.py
n=int(input("Enter a number to generate table:"))
if(n<0):
    print("Invalid Input")
else:
    print("-"*50)
    print("Table of:{}".format(n))
    print("-"*50)
    i=1
    while(i<=10):
        print("\t{}x{}={}".format(n,i,n*i))
        i=i+1
    else:
        print("-"*50)
```

EXAMPLE:-5

#WPP which will accept a number and find sum of the digits of the given number.

```
#sumdigit.py
n=int(input("Enter a number:"))
if(n<0):
    print("{} is invalid input".format(n))
else:
    print("Given number is:{}".format(n))
    s=0
    while(n>0):
        d=n%10
        s=s+d
        n=n//10
    else:
        print("sum of digits :{}".format(s))
```

EXAMPLE:-6

#WPP which will accept a string and print its revers order.

```
#revers.py
```

```
str=input("enter any word:")
s=str[::-1]
print(s)
```

EXAMPLE-7

#WPP which will accept a string and print it is palindrome or not.

```
#palindrom.py
str=input("enter any word:")
s=str[::-1]
print("Revers of input string is:",s)
if(s==str):
    print("{} is palindrome.".format(str))
else:
    print("{} is not palindrome.".format(str))
```

EXAMPLE-8:

```
#wp to print reverse of input string
#reverseex1.py
str=input("Enter any string:")
l=len(str)
print("Length of string :",l)
i=l-1
while(i>=0):
    print(str[i])
    i=i-1
```

EXAMPLE-9:

```
#whilereverse.py
str=input("Enter any word:")
l=len(str)
print("Revers of input string:")
i=l-1
while(i>=0):
    print(str[i],end="")
    i=i-1
else:
    print("\n")
    print("*50")
    if(str==str[::-1]):
        print("{} is palindrome ".format(str))
    else:
        print("{} is not palindrome ".format(str))
```

for loop (or) for....else Loop

=>The purpose of for loop is that to retrieve the data or extract the data from any Iterable object.

=>Syntax1:

```
for varname in Iterable_Object:
```

Block of Statements

Other Statements in Program

(OR)

=>Syntax2:

```
for varname in Iterable_Object:
```

Block of Statements

else:

else block of statements

Other Statements in Program

Explanation:

=>Here 'for' , 'in' , 'else' are the keywords

=>Varname represents Programmer choice

=>Iterable object represents any object contains multiple values like sequence, list, set and dict types.

=>The execution process of for loop is that "Each Element of Iterable Object selected, placed in Varname and executes Block of statements". This Process will be repeated for finite number of times

(len(iterable object)) until all elements are completed in Iterable object

=>After all elements of Iterable_Object of for loop completed and it indicates condition of for loop is false and later executes else block of statements which are written in else block. After executing block PVM executes Other statements in Program

=>Writing else block is optional.

Note1:- For Iterbale objects repetition, It is recommended for using "for Loop"

Note2:- For Non-Iterable_objects repetition, It is recommended for using " while Loop"

Example:1

```
# This program extracts or retrieve the from given string
#forloopex1.py
s="PYTHON"
print("By using while loop")
i=0
while(i<len(s)):
    print(s[i])
    i=i+1
print("*50")
print("By using for loop")
for val in s:
    print(val)
print("*50")
print("By using while loop")
lst=[10,"Rossum",34.56,"Python"]
i=0
while(i<len(lst)):
    print(lst[i])
    i=i+1
print("*50")
print("By using for loop")
for val in lst:
    print(val)
print("*50")
print("By using while loop")
r=range(100,106)
j=0
while(j<len(r)):
    print(r[j])
    j=j+1
print("*50")
print("By using for loop")
for val in r:
    print(val,end=" ")
```

EXAMPLE:2

```
#WPP which will generate 1 to n number.where n is positive(use for loop)
#fornumgenex1.py
n=int(input("Enter a number:"))
if(n<=0):
    print("invalid Input")
else:
    print("-"*50)
```

```
print("Numbers within :",n)
print("-"*50)
for i in range(1,n+1):
    print("{}".format(i),end=" ")
else:
    print("\n")
print("-"*50)
```

EXAMPLE:3

#WPP which will accept any integer value and generate even no. and odd no. separately.

```
#forevenodd.py
n=int(input("Enter a number:"))
if(n<=0):
    print("invalid Input")
else:
    print("-"*50)
    print("Even numbers within:",n)
    print("-"*50)
    for even in range(2,n+1,2):
        print("{}".format(even),end=" ")
    else:
        print("\n")
        print("-"*50)
        print("Odd numbers within:",n)
        print("-"*50)
        for odd in range(1,n+1,2):
            print("{}".format(odd),end=" ")
        else:
            print("\n")
            print("-"*50)
=====
=====
```

EXAMPLE:4

#WPP which will accept any numerical integer value and display multiplication table,where n is positive.

```
#formultable.py
n=int(input("Enter a number:"))
if(n<=0):
    print("Invalid input.")
else:
    print("-"*50)
    print("Multiplicatin table of:",n)
    print("-"*50)
    for i in range(1,11):
        print("\t{}x{}={}".format(n,i,n*i))
```

```
else:  
    print("_"*50)
```

EXAMPLE:5

```
#WPP which will find sum of first n natural number,sum of its squares and sum of its  
cubes.  
#forsumsqcube.py  
n=int(input("Enter a number:"))  
if(n<=0):  
    print("invalid input.")  
else:  
    s,ss,sc=0,0,0  
    for i in range(1,n+1):  
        s=s+i  
        ss=ss+i**2  
        sc=sc+i**3  
        i=i+1  
    else:  
        print("Sum of number from 1 to {}:{}".format(n,s))  
        print("Sum of square of number from 1 to {}:{}".format(n,ss))  
        print("Sum of cube of number from 1 to {}:{}".format(n,sc))
```

Example of for loop

EXAMPLE:1

```
#sqcube.py  
n=int(input("Enter 1st n natural no. to calculate its sum of no.,sq,cube."))  
if(n<=0):  
    print("Invalid Input.")  
else:  
    s,ss,sc=0,0,0  
    print("-"*50)  
    print("\tno.\tsquare\tcube")  
    print("-"*50)  
    for i in range(1,n+1):  
        s=s+i  
        ss=ss+i**2  
        sc=sc+i**3  
        print("\t{}\t{}\t{}".format(i,i**2,i**3))  
    else:  
        print("-"*50)  
        print("\t{}\t{}\t{}".format(s,ss,sc))  
        print("-"*50)
```

EXAMPLE:2

```
#calculate sum of a number using list.
```

```
#listsum.py
n=int(input("Enter how many element you want to add:"))
if(n<=0):
    print("invalid input.")
else:
    lst=list()
    for i in range(1,n+1):
        print("Enter {} element:".format(i))#m=float(input("Enter {} element:".format(i)))
        m=float(input())
        lst.append(m)
    else:
        print("You entered:",lst)
        s=0
        for val in lst:
            print("\t{}".format(val))
            s=s+val
        else:
            print("sum=",s)
            print("Avg=",s/len(lst))
```

EXAMPLE:3

```
#sumavg1.py
n=int(input("How many element you want to add:"))
if(n<=0):
    print("Invalid Input.")
else:
    i=1
    s=0
    while(i<=n):
        val=float(input("Enter {} value: ".format(i)))
        s=s+val
        i=i+1
    else:
        print("Sum=",s)
        print("Avg=",s/n)
```

EXAMPLE-4:

```
#wpp which will accept list of numerical values and sort them in ascending and
descending order.
#sortno.py
n=int(input("Enter how many element you want to sort:"))
if(n<=0):
    print("Invalid Input.")
else:
    lst=list()
    for i in range(1,n+1):
```

```
val=int(input())
lst.append(val)
else:
    print("You entered:",lst)
    print("Values in ascending order:")
    lst.sort()
    for val in lst:
        print(val,end=" ")
else:
    print()
    print("values in descending order:")
    lst.sort(reverse=True)
    for val in lst:
        print(val,end=" ")
```

EXAMPLE:-5

#wpp which will accept list of values and sort them in ascendind and desending order.

```
#sortword.py
n=int(input("Enter how many element you want to sort:"))
if(n<=0):
    print("Invalid Input.")
else:
    lst=list()
    for i in range(1,n+1):
        val=input()
        lst.append(val)
    else:
        print("You entered:",lst)
        print("Values in ascending order:")
        lst.sort()
        for val in lst:
            print(val,end=" ")
    else:
        print()
        print("values in descending order:")
        lst.sort(reverse=True)
        for val in lst:
            print(val,end=" ")
```

EXAMPLE-6

#write a python program which will obtain reverse of given number.

```
#reverseno.py
n=int(input("Enter any integer number:"))
if(n<=0):
    print("Invalid input")
```

```
else:  
    i=1  
    rvs=0  
    d=0  
    while(n>0):  
        d=n%10  
        rvs=rvs*10+d  
        n=n//10  
    else:  
        print("Reversed of a number:",rvs)
```

OUTPUT:-

Enter any integer number:12345

Reversed of a number: 54321

#This program accepting age of Citizen and decide eligible to vote or not.

```
#VoterEx1.py  
age=int(input("Enter the age of Citizen:"))  
if(age>=18):  
    print("Citizen is eligible to Vote:")  
else:  
    print("Citizen is not eligible to Vote:")
```

#This program accepting age of Citizen and decide eligible to vote or not.

```
#VoterEx2.py  
while(True):  
    age=int(input("Enter the Correct age of Citizen:"))  
    if(age>=18) and (age<=100) :  
        break
```

```
print("{} years old Citizen is eligible to Vote:".format(age))
```

#This program accepting age of Citizen and decide eligible to vote or not.

```
#VoterEx3.py  
ctr=0  
while(ctr<3):  
    age=int(input("Enter the Correct age of Citizen:"))  
    if(age>=18) and (age<=100) :  
        break  
    ctr=ctr+1  
  
if(ctr==3):  
    print("U lost ur chance registering as voter and u are blocked")  
else:  
    print("{} years old Citizen is eligible to Vote:".format(age))
```

```

#StudentMarksReport.py
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):
    pym=int(input("Enter Marks in PYTHON:"))
    if(pym>=0) and (pym<=100):
        break

#calculate total and percentage of marks
totmarks=cm+cppm+pym
permarks=(totmarks/300)*100
#decides the grades
if(cm<40) or (cppm<40) or (pym<40):
    grade="FAIL"
else:
    if(totmarks<=300) and (totmarks>=250):
        grade="DISTINCTION"
    if(totmarks<=249) and (totmarks>=200):
        grade="FIRST"
    if(totmarks<=199) and (totmarks>=150):
        grade="SECOND"
    if(totmarks<=149) and (totmarks>=120):
        grade="THIRD"

```

```

grade="THIRD"
#Display the Marks Memo
print("*70)
print("\tStudent Marks Report")
print("*70)
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(pym))
print("-*70)
print("\tStudent Total Marks:{}".format(totmarks))
print("\tStudent Percentage of Marks:{}".format(permarks))
print("-*70)
print("\tStudent Grade:{}".format(grade))
print("*70)

```

#WPP which will accept employee no. ,employee name and basic salary and calculate netsalary and show payslip.

#empsalex2.py

```

empno=int(input("Enter a employee number:"))
empname=(input("Enter name of employee:"))
while(True):
    basicsal=int(input("Enter basic salary of employee:"))
    if(basicsal>=1000)and(basicsal<=10000):
        break
    print("basic salary:",basicsal)
    if(basicsal>=1000)and(basicsal<=5000):
        hra=(15*basicsal)/100
        da=(20*basicsal)/100
        ta=(14*basicsal)/100
        ma=(4*basicsal)/100
        gpf=(2*basicsal)/100
        lic=(2*basicsal)/100
    else:
        hra=(15*basicsal)/100
        da=(20*basicsal)/100
        ta=(14*basicsal)/100
        ma=(2*basicsal)/100
        gpf=(2*basicsal)/100
        lic=(2*basicsal)/100
    netsal=basicsal+(hra+da+ta+ma)-(gpf+lic)
    print("-*60)
    print("\t\tEmployee pay slip ")
    print("-*60)
    print("\tEmployee number:{}".format(empno))
    print("\tEmployee name:{}".format(empname))
    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("-"*60)
print("\tEmployee net salary:{}".format(netsal))
print("-"*60)
```

OUTPUT:

Enter a employee number:123
Enter name of employee:Pooja
Enter basic salary of employee:7000
basic salary: 7000

employee pay slip

Employee number:123
Employee name:Pooja
Employee basic salary:7000
Employee da:1400.0
Employee ta:980.0
Employee hra:1050.0
Employee ma:140.0
Employee gpf:140.0
Employee lic:140.0

Employee net salary:10290.0

break

=>break statement is used for terminating the execution process of Looping provided certain condition satisfied.
=>break statement always used inside of loop statements.
=>when we use break statement inside of the Loop, PVM control comes out of loop

and executes other statements in the program but not corresponding else block of loop.

=>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  
-----
```

```
=====X=====  
==  
#breakex1.py  
s="PYTHON"  
for v in s:  
    if(v=="O"):  
        break  
    else:  
        print("\t{}".format(v))  
else:  
    print("i am from else of for loop")  
  
print("Other statements in program")
```

```
#breakex2.py  
lst=[10,"Rossum",34.56,True,2+3j]  
for val in lst:  
    if(val==True):  
        break  
    else:  
        print("\t{}".format(val))
```

```
#breakex3.py
d={10:"Python",20:"Data Scienece",30:"Django",40:"Java"}
for k,v in d.items():
    if (k==30):
        break
    else:
        print("\t{}-->{}".format(k,v))
```

#This program accept a number and decide wethere it is prime or not

```
#primeex1.py
n=int(input("Enter a Number:"))
if(n<=1):
    print("{} is invalid input:".format(n))
else:
    result=True
    for i in range(2,n):
        if(n%i==0):
            result=False
            break

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

#This program accept a number and decide wethere it is prime or not

```
#primeex2.py
n=int(input("Enter a Number:"))
if(n<=1):
    print("{} is invalid input:".format(n))
else:
    result="Prime"
    for i in range(2,n):
        if(n%i==0):
            result="Not Prime"
            break

    if(result=="Prime"):
        print("{} is Prime:".format(n))
    else:
        print("{} is not Prime:".format(n))
```

#This program accept a number and decide wethere it is prime or not

```
#primeex3.py
n=int(input("Enter a Number:"))
if(n<=1):
    print("{} is invalid input:".format(n))
else:
    result=0
```

```
for i in range(2,n):
    if(n%i==0):
        result=1
        break
    #other statements
    if(result==0):
        print("{} is Prime:".format(n))
    else:
        print("{} is not Prime:".format(n))
```

Continue statement

=>continue is one of the keyword

=>The purpose of continue statement is that to make control to top of the loop for that current iteration without executing the following statement written after continue statement

Syntax1:-

```
-----
while(Test cond1):
-----
-----
if(Test cond2):
-----
    continue
-----
statements written - after continue statement
-----
-----
```

Syntax2:-

```
-----
for varname in iterable-object
-----
-----
if(Test cond):
-----
    continue
-----
statements written - after continue statement
-----
-----
```

```
#continueex1.py
s="PYTHON"
# want to display  PYTON
for v in s:
    if(v=="H"):
```

```
        continue
else:
    print("\t{}".format(v))
else:
    print("else part of for loop")
print("-"*60)
# want to display PYTN
for v in s:
    if(v=="H") or(v=="O"):
        continue
    else:
        print("\t{}".format(v))
```

```
#continueex1.py
s="PYTHON"
# want to display PYTON
i=0
while(i<len(s)):
    if(s[i]=="H"):
        i=i+1
        continue
    else:
        print(s[i])
        i=i+1
print("-"*60)
# want to display PYTN
i=0
while(i<len(s)):
    if(s[i]=="H") or (s[i]=="O"):
        i=i+1
        continue
    else:
        print(s[i])
        i=i+1
```

#This Program find sum of +ve numbers and -ve numbers separately.

```
#continueex3.py
lst=[10,-20,30,40,-34,-45,89,-3,23,6]
ps=0
for val in lst:
    if(val<0):
        continue
    else:
        print("\t\t{}".format(val))
        ps=ps+val
else:
    print("Positive Sum={}".format(ps))
    print("-"*60)
    ns=0
    for val in lst:
        if(val>0):
            continue
```

```

        else:
            print("\t\t{}".format(val))
            ns=ns+val
    else:
        print("Negative Sum={}{}".format(ns))
        print("*60)

```

```

#This Program accept a line of text and display the Vowels
#line=input("Enter a line of text:") # Python is an oop lang
for v in line:
    if v not in ['a','e','i','o','u','A','E','I','O','U']:
        continue
    else:
        print("\t{} and whose index={}".format(v,line.index(v)))

```

Nested (or) Inner Loops

- =>The Process writing / Defining one loop inside of another loop is called Nested / Inner Loop.
 - =>The execution Process of nested / inner loop is that "For Every Value of Outer Loop inner loop repeated for finite number of times.
-

=>Syntax1: (for loop in for loop)

```

for varnam1 in Iterable_object: # outer for Loop
-----
    for varnam2 in Iterable_object: # Inner for Loop
-----
    else:
-----
else:
-----
```

Examples:

```

#innerloopsex1.py
print("-"*40)
for i in range(1,6):
    print("Val of i-Outer loop={}".format(i))
    print("-"*40)
    for j in range(1,4):
        print("\tVal of j-Inner loop={}".format(j))
    else:
        print("-"*40)
        print("i am out of inner loop")

```

```
    print("-"*40)

else:
    print("I am out of outer for loop")
    print("-"*40)
```

=>Syntax2: (while loop in while loop)

```
-----  
        while (test cond1): # Outer while loop  
-----  
            while(Test cond2): # inner while loop  
-----  
            else:  
-----  
            else:  
-----
```

Examples:

```
#innerloopsex2.py
print("-"*40)
i=1
while(i<6):
    print("Val of i--outer while loop=",i)
    print("-"*40)
    j=1
    while(j<4):
        print("\tVal of j-inner while loop=",j)
        j=j+1
    else:
        print("-"*40)
        print("\ti am out-of inner while loop")
        i=i+1
        print("-"*40)
else:
    print("i am out-of outer while loop")
```

=>Syntax3: (while loop in for loop)

```
-----  
        for varnam1 in Iterable_object: # outer for Loop  
-----  
            while(Test cond2): # inner while loop  
-----  
            else:  
-----
```

else:

Example:

```
#innerloopsex3.py
print("-"*40)
for i in range(1,6): # outer loop
    print("Val of i-Outer for loop={}".format(i))
    print("-"*40)
    j=3
    while(j>0): # Inner loop
        print("\tVal of j-inner while loop=",j)
        j=j-1
    else:
        print("-"*40)
        print("\ti am out-of inner while loop")
        print("-"*40)
else:
    print("I am out of outer for loop")
    print("-"*40)
```

=>Syntax4: for loop in while loop

while (test cond1): # Outer while loop

for varnam2 in Iterable_object: # Inner for Loop

else:

else:

Examples:

```
#innerloopsex4.py
print("-"*40)
i=1
while(i<6):
    print("Val of i--outer while loop=",i)
    print("-"*40)
    for j in range(1,4):
        print("\tVal of j-Inner for loop={}".format(j))
    else:
        i=i+1
        print("-"*40)
        print("i am out of inner for loop")
        print("-"*40)
```

```
else:  
    print("i am out-of outer while loop")  
=====X=====
```

Nested (or) Inner Loops

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

=>The execution Process of nested / inner loop is that "For Every Value of Outer Loop inner loop repeated for finite number of times.

=>Syntax1: (for loop in for loop)

```
for varnam1 in Iterable_object: # outer for Loop  
-----  
    for varnam2 in Iterable_object: # Inner for Loop  
-----  
        else:  
-----  
        else:  
-----
```

Examples:

```
#innerloopsex1.py  
print("-"*40)  
for i in range(1,6):  
    print("Val of i-Outer loop={}".format(i))  
    print("-"*40)  
    for j in range(1,4):  
        print("\tVal of j-Inner loop={}".format(j))  
    else:  
        print("-"*40)  
        print("i am out of inner loop")  
        print("-"*40)  
  
else:  
    print("I am out of outer for loop")  
    print("-"*40)
```

=>Syntax2: (while loop in while loop)

```
while (test cond1): # Outer while loop  
-----  
    while(Test cond2): # inner while loop  
-----
```

```
-----  
    else:  
-----  
    else:  
-----
```

Examples:

```
-----  
#innerloopsex2.py  
print("-"*40)  
i=1  
while(i<6):  
    print("Val of i--outer while loop=",i)  
    print("-"*40)  
    j=1  
    while(j<4):  
        print("\tVal of j-inner while loop=",j)  
        j=j+1  
    else:  
        print("-"*40)  
        print("\ti am out-of inner while loop")  
        i=i+1  
        print("-"*40)  
else:  
    print("i am out-of outer while loop")
```

=>Syntax3: (while loop in for loop)

```
-----  
for varnam1 in Iterable_object: # outer for Loop  
-----  
    while(Test cond2): # inner while loop  
-----  
    else:  
-----  
else:  
-----
```

Example:

```
-----  
#innerloopsex3.py  
print("-"*40)  
for i in range(1,6): # outer loop  
    print("Val of i-Outer for loop={} ".format(i))  
    print("-"*40)  
    j=3  
    while(j>0): # Inner loop  
        print("\tVal of j-inner while loop=",j)  
        j=j-1
```

```
else:  
    print("-"*40)  
    print("\ti am out-of inner while loop")  
    print("-"*40)  
else:  
    print("I am out of outer for loop")  
    print("-"*40)
```

=>Syntax4: for loop in while loop

```
-----  
while (test cond1): # Outer while loop  
-----  
    for varnam2 in Iterable_object: # Inner for Loop  
-----  
    else:  
-----  
else:  
-----
```

Examples:

```
#innerloopsex4.py  
print("-"*40)  
i=1  
while(i<6):  
    print("Val of i--outer while loop=",i)  
    print("-"*40)  
    for j in range(1,4):  
        print("\tVal of j-Inner for loop={}".format(j))  
    else:  
        i=i+1  
        print("-"*40)  
        print("i am out of inner for loop")  
        print("-"*40)  
else:  
    print("i am out-of outer while loop")
```

=====X=====

```
#innerloopsex1.py
print("-"*40)
for i in range(1,6):
    print("Val of i-Outer loop={}".format(i))
    print("-"*40)
    for j in range(1,4):
        print("\tVal of j-Inner loop={}".format(j))
    else:
        print("-"*40)
        print("i am out of inner loop")
        print("-"*40)

else:
    print("I am out of outer for loop")
    print("-"*40)
```

E:\KVR-PYTHON-9AM\INNER LOOPS>py innerloopsex1.py

Val of i-Outer loop=1

Val of j-Inner loop=1
Val of j-Inner loop=2
Val of j-Inner loop=3

i am out of inner loop

Val of i-Outer loop=2

Val of j-Inner loop=1
Val of j-Inner loop=2
Val of j-Inner loop=3

i am out of inner loop

Val of i-Outer loop=3

Val of j-Inner loop=1
Val of j-Inner loop=2
Val of j-Inner loop=3

i am out of inner loop

Val of i-Outer loop=4

```
Val of j-Inner loop=1  
Val of j-Inner loop=2  
Val of j-Inner loop=3
```

```
i am out of inner loop
```

```
Val of i-Outer loop=5
```

```
Val of j-Inner loop=1  
Val of j-Inner loop=2  
Val of j-Inner loop=3
```

```
i am out of inner loop
```

```
I am out of outer for loop
```

```
"""
```

```
#innerloopsex2.py  
print("-"*40)  
i=1  
while(i<6):  
    print("Val of i--outer while loop=",i)  
    print("-"*40)  
    j=1  
    while(j<4):  
        print("\tVal of j-inner while loop=",j)  
        j=j+1  
    else:  
        print("-"*40)  
        print("\ti am out-of inner while loop")  
        i=i+1  
        print("-"*40)  
else:  
    print("i am out-of outer while loop")
```

```
"""
```

```
E:\KVR-PYTHON-9AM\INNER LOOPS>py innerloopsex2.py
```

```
Val of i-outer while loop= 1
```

```
Val of j-inner while loop= 1  
Val of j-inner while loop= 2  
Val of j-inner while loop= 3
```

```
i am out-of inner while loop
```

```
Val of i-outer while loop= 2
```

```
Val of j-inner while loop= 1
```

```
Val of j-inner while loop= 2
Val of j-inner while loop= 3
-----
i am out-of inner while loop
-----
Val of i-outer while loop= 3
-----
Val of j-inner while loop= 1
Val of j-inner while loop= 2
Val of j-inner while loop= 3
-----
i am out-of inner while loop
-----
Val of i-outer while loop= 4
-----
Val of j-inner while loop= 1
Val of j-inner while loop= 2
Val of j-inner while loop= 3
-----
i am out-of inner while loop
-----
Val of i-outer while loop= 5
-----
Val of j-inner while loop= 1
Val of j-inner while loop= 2
Val of j-inner while loop= 3
-----
i am out-of inner while loop
-----
i am out-of outer while loop"""
```

```
#innerloopsex3.py
print("-"*40)
for i in range(1,6): # outer loop
    print("Val of i-Outer for loop={}".format(i))
    print("-"*40)
    j=3
    while(j>0): # Inner loop
        print("\tVal of j-inner while loop=",j)
        j=j-1
    else:
        print("-"*40)
        print("\ti am out-of inner while loop")
        print("-"*40)
else:
    print("I am out of outer for loop")
    print("-"*40)
```

"""

E:\KVR-PYTHON-9AM\INNER LOOPS>py innerloopsex3.py

Val of i-Outer for loop=1

Val of j-inner while loop= 3
Val of j-inner while loop= 2
Val of j-inner while loop= 1

i am out-of inner while loop

Val of i-Outer for loop=2

Val of j-inner while loop= 3
Val of j-inner while loop= 2
Val of j-inner while loop= 1

i am out-of inner while loop

Val of i-Outer for loop=3

Val of j-inner while loop= 3
Val of j-inner while loop= 2
Val of j-inner while loop= 1

i am out-of inner while loop

Val of i-Outer for loop=4

Val of j-inner while loop= 3
Val of j-inner while loop= 2
Val of j-inner while loop= 1

i am out-of inner while loop

Val of i-Outer for loop=5

Val of j-inner while loop= 3
Val of j-inner while loop= 2
Val of j-inner while loop= 1

i am out-of inner while loop

I am out of outer for loop

.....

```
#innerloopsex4.py
print("-"*40)
i=1
while(i<6):
    print("Val of i--outer while loop=",i)
    print("-"*40)
    for j in range(1,4):
        print("\tVal of j-Inner for loop={}".format(j))
    else:
        i=i+1
        print("-"*40)
        print("i am out of inner for loop")
        print("-"*40)
else:
    print("i am out-of outer while loop")

"""
E:\KVR-PYTHON-9AM\INNER LOOPS>py innerloopsex4.py
```

```
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
```

```
i am 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
```

```
i am 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
```

```
i am 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

i am 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

i am out of inner for loop

i am out-of outer while loop"""

#innerloopsex5.py

write a python program which will accept list of numerical values and display the multiplication table for every value of list.

lst=[-3,6,-12,-19,-4,0]

for n in lst:

 if(n<=0):

 print("{} is invalid input and No Mul Table:".format(n))

 else:

 print("-"*40)

 print("Mul Table for {}".format(n))

 print("-"*40)

 for i in range(1,11):

 print("\t{} x {}={}".format(n,i,n*i))

 else:

 print("-"*40)

#innerloopsex6.py

write a python program which will accept list of numerical values and display the multiplication table for every value of list.

lst=list()

nt=int(input("How Many mul tables u want:"))

if(nt<=0):

 print("{} is Invalid Input".format(nt))

else:

 for i in range(1,nt+1):

 val=int(input("Enter {} number:".format(i)))

 lst.append(val)

 else:

 print("-"*40)

 print("Given List of elements={}".format(lst))

 print("-"*40)

 for n in lst: # Outer for loop

 if(n<=0):

```

        print("{} is invalid input and No Mul
Table:".format(n))
    else:
        print("-"*40)
        print("Mul Table for {}".format(n))
        print("-"*40)
        for i in range(1,11): # inner for loop
            print("\t{} x {}={}".format(n,i,n*i))
        else:
            print("-"*40)

```

```

#innerloopsex7.py
# write a python program which will accept list of numerical values and display the
prime numbers
lst=list()
nt=int(input("How Many Numbers u Have:"))
if(nt<=0):
    print("{} is Invalid Input".format(nt))
else:
    for i in range(1,nt+1):
        val=int(input("Enter {} number:".format(i)))
        lst.append(val)
    else:
        print("-"*40)
        print("Given List of elements={}".format(lst)) # [12, 21, 3, 7, 14, 46]
        print("-"*40)
        primelist=list()
        nonprimelist=[]
        for n in lst: #outer for loop--which will supply the values of list one
by one
        if(n<=0):pass
        else:
            result=True
            for i in range(2,n):
                if(n%i==0):
                    result=False
                    break
            if(result):
                primelist.append(n)
            else:
                nonprimelist.append(n)
        else:
            print("Given List={}".format(lst))
            print("Prime Numbers List={}".format(primelist))
            print("Non-Prime Numbers List={}".format(nonprimelist))

```

```

#innerloopsex8.py
# write a python program which will accept list of numerical values and display the
prime numbers
lst=list()
nt=int(input("How Many Numbers u Have:"))
if(nt<=0):
    print("{} is Invalid Input".format(nt))
else:
    for i in range(1,nt+1):
        val=int(input("Enter {} number:".format(i)))
        lst.append(val)
    else:
        print("-"*40)
        print("Given List of elements={}".format(lst)) # [12, 21, 3, 7, 14, 46]
        print("-"*40)
        primelist=list()
        nonprimelist=[]
        for n in lst: #outer for loop--which will supply the values of list one
by one
        if(n<=0):pass
        else:
            result=True
            for i in range(2,n):
                if(n%i==0):
                    result=False
                    break
            if(result):
                primelist.append(n)
            else:
                nonprimelist.append(n)
        else:
            print("Given List={}".format(lst))
            print("Prime Numbers List={}".format(primelist))
            print("Non-Prime Numbers List={}".format(nonprimelist))

```

```
#innerloopsex7.py
# write a python program which will accept list of numerical values and display the
prime numbers
lst=list()
nt=int(input("How Many Numbers u Have:"))
if(nt<=0):
    print("{} is Invalid Input".format(nt))
else:
    for i in range(1,nt+1):
        val=int(input("Enter {} number:".format(i)))
        lst.append(val)
    else:
        print("-"*40)
        print("Given List of elements={}".format(lst)) # [12, 21, 3, 7, 14, 46]
        print("-"*40)
        primelist=list()
        nonprimelist=[]
        for n in lst: #outer for loop--which will supply the values of list one
by one
        if(n<=0):pass
        else:
            result=True
            for i in range(2,n):
                if(n%i==0):
                    result=False
                    break
            if(result):
                primelist.append(n)
            else:
                nonprimelist.append(n)
        else:
            print("Given List={}".format(lst))
            print("Prime Numbers List={}".format(primelist))
            print("Non-Prime Numbers List={}".format(nonprimelist))
```

Functions in Python

-
- =>The purpose of Functions is that "To perform certain Operation and provides Re-usability".
 - =>Def. of Function:
 - =>A Part of main program is called Function
 - (or)
 - =>Sub Program of main program is called Function.
-

=>Advantages of Functions:

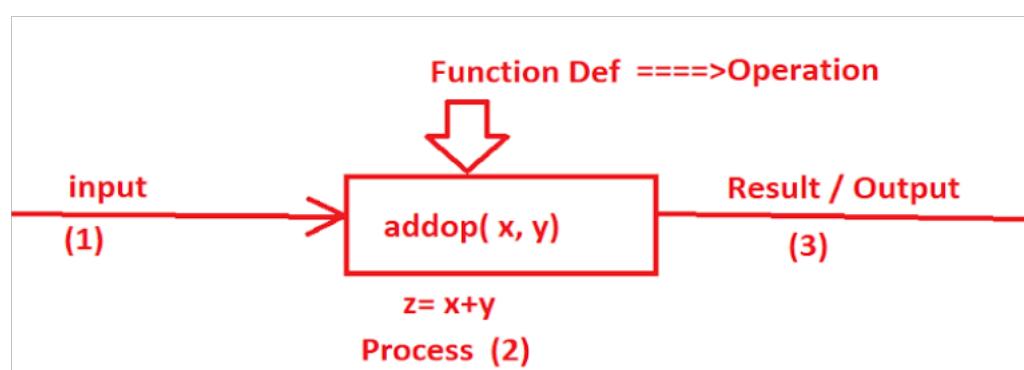
-
- =>If we develop any problem or program by using Function concept , we get following advantages.
 - =>Application Development time is Less
 - =>Application Memory Space is Less
 - =>Application Execution time is Less
 - =>Application Performance is Enhanced (Improved)
 - =>Redundency (Duplication) of Code is Minimized
-

=>Parts of Functions:

- =>When we deal with Functional Programming, we must ensure that there must exist 2 parts. They are
 - 1) Function Definition
 - 2) Function Calls
 - =>A particular Function Definition exist only once and we can have one or more Function Calls.
 - =>For Every Function call , there must exists a Function Definition otherwise we get NameError.
-

Phases in Functions:

- =>In Functions, there must exists 3 phases.
 - a) Every Function Must Take INPUT
 - b) Every Function Must PROCESS the input
 - c) Every Function must give OUTPUT / RESULT
-



Syntax for defining the Function:

```
def functionname(list of formal params if any): <-- Function Heading  
    """ doc string """  
    statement-1  
    statement-2  
    -----  
    statement-n
```

Function Body

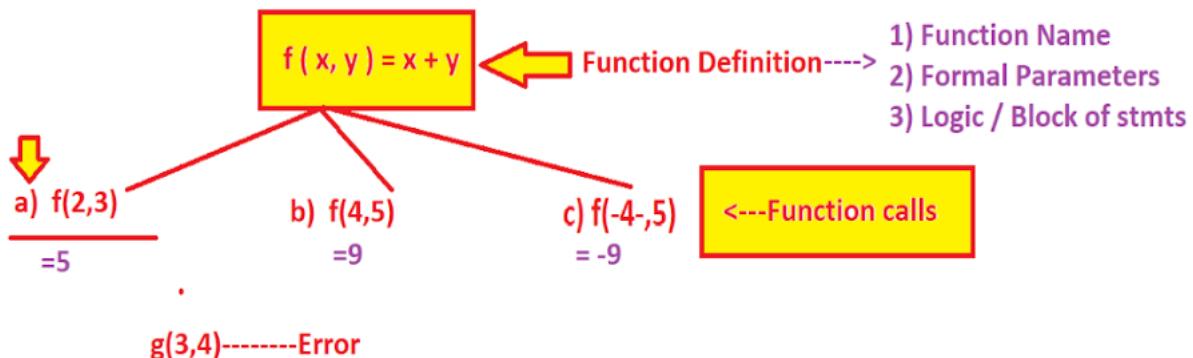
Function Definition

Explanation:

1. "def" is keyword used for defining Function definition
2. "functionname" is valid variable name and it is an object of class 'function'
3. "list of formal params " represents list of variables used function heading and they are used for storing or holding the inputs coming from function calls.
4. 'doc string' represents document string and it describes the functionality of function.
5. 'statement-1, statement-2...statement-n' represents Indentation block of statements represents Business Logic for Problem Solving.
6. In Function body, we use some variables for storing temporary result during function execution and such type variables are called "Local Variables".
7. The Values of Local Variables and Formal Params can be accessed within corresponding Function Definition only but not possible to access in some function definitions.

School Days---Maths---- Functions

Q1) Consider $f(x, y) = x + y$ find a) $f(2,3)$ b) $f(4,5)$ c) $f(-4,-5)$



Types of Programming Languages

=>In industry we have two types of programming languages. They are

1. Un-Structured Programming Languages.
 2. Structured Programming Languages.
-

1. Un-Structured Programming Languages.

=>Un-Structured Programming Languages does not contain the concept of Functions.

=>Since Functions concept not present in Un-Structured Programming Languages, it has the following Limitations.

- =>Application Development time is More
- =>Application Memory Space is More
- =>Application Execution time is More
- =>Application Performance is Degraded
- =>Redundancy (Duplication) of Code is More

=>Example: GW-BASIC

2. Structured Programming Languages.

=>Structured Programming Languages contain the concept of Functions.

=>Since Functions concept present in Structured Programming Languages, it has the following Advantages.

- =>Application Development time is less.
- =>Application Memory Space is less.
- =>Application Execution time is less.
- =>Application Performance is Enhanced or Improved.
- =>Redundancy (Duplication) of Code is minimized

=>Example: C,CPP,JAVA,PYTHON

We are all developing a TASK --GW-BASIC

"ADDITION of TWO Numbers---350---Students--team members and KVR is a teacher

In this Programming
--350 Students has to
defined 350 times 6 lines
of code

- =>Application Development time is More
- =>Application Memory Space is More
- =>Application Execution time is More
- =>Application Performnace is Degraded
- =>Redundancy (Duplication) of Code is More

s1
a=float(input("Enter First value"))
b=float(input("Enter Second value"))
c=a+b
print("Val of a={}".format(a))
print("Val of b={}".format(b))
print("Sum={}".format(c))

s2
a=float(input("Enter First value"))
b=float(input("Enter Second value"))
c=a+b
print("Val of a={}".format(a))
print("Val of b={}".format(b))
print("Sum={}".format(c))

Numbers of approaches in functions-MOST IMP

Program-1

#This program display the values of list,tuple,set, dict

#iterabledisp.py

```
def disp(k):
    print("type of k=",type(k))
    print("-"*40)
    for val in k:
        print("\t{}".format(val))
    else:
        print("-"*40)
def dispdict(k):
    print("type of k=",type(k))
    print("-"*40)
    for cno,cname in k.items():
        print("\t{}-->{}".format(cno,cname))
    else:
        print("-"*40)
#main program
lst=[10,20,30,40,50]
disp(lst) #function call
tpl=("Python","Django","Data Science","ML","DL")
disp(tpl) #function call
s={10,"Rossum",34.56,"Python",True}
disp(s) #function call
d={1:"Python",2:"Data Science",3:"Django",4:"Java",5:".Net"}
dispdict(d) # function call
```

Program-2

#This program accept list of values and sort them

```
#listsort.py
def disp(k):
    print("-"*40)
    for val in k:
        print("\t{}".format(val))
    else:
        print("-"*40)
def sortelements(lst):
    print("Original Elements")
    disp(lst)
    lst.sort()
    print("Sorted Elements in Ascending Order")
    disp(lst)
    lst[::-1] # or lst.reverse() or lst.sort(reverse=True)
    print("Sorted Elements in Descending Order")
    disp(lst)

#main program
lst=[10,2,222,50,10,4,55,-3,0,22]
sortelements(lst)
```

```
#This program display the values of list,tuple,set, dict
#iterabledisp.py
def disp(k):
    print("type of k=",type(k))
    print("-"*40)
    for val in k:
        print("\t{}".format(val))
    else:
        print("-"*40)
def dispdict(k):
    print("type of k=",type(k))
    print("-"*40)
    for cno,cname in k.items():
        print("\t{}-->{}".format(cno,cname))
    else:
        print("-"*40)
#main program
lst=[10,20,30,40,50]
disp(lst) #function call
tpl=("Python","Django","Data Science","ML","DL")
disp(tpl) #function call
s={10,"Rossum",34.56,"Python",True}
disp(s) #function call
d={1:"Python",2:"Data Science",3:"Django",4:"Java",5:".Net"}
dispdict(d) # function call
```

```
#This program accept list of values and sort them
#listsort.py
def disp(k):
```

```

print("-"*40)
for val in k:
    print("\t{}".format(val))
else:
    print("-"*40)
def sortelements(lst):
    print("Original Elements")
    disp(lst)
    lst.sort()
    print("Sorted Elements in Ascending Order")
    disp(lst)
    lst[::-1] # or lst.reverse() or lst.sort(reverse=True)
    print("Sorted Elements in Descending Order")
    disp(lst)
#main program
lst=[10,2,222,50,10,4,55,-3,0,22]
sortelements(lst)

```

List comprehension

=>The purpose of List comprehension is that to read the values dynamically from key board separated by a delimiter (space, comma, colon..etc)
=>List comprehension is the most effective way for reading the data for list instead traditional reading the data.
=>Syntax:- listobj=[expression for varname in Iterable_object]
=>here expression represents either type casting or mathematical expression

Examples:

```

print("Enter List of values separated by space:") # [10,2,222,50,10,4,55,-3,0,22]
lst= [float(val) for val in input().split() ]
print("content of lst",lst)

```

Examples:

```

lst=[4,3,7,-2,6,3]
newlst=[ val*2 for val in lst]
print("new list=",newlst) # [ 8, 6, 14,-4,12,6 ]

```

```

#listcompre.py
print("Enter List of values separated by space:") # [10,2,222,50,10,4,55,-3,0,22]
lst= [float(val) for val in input().split() ]
print("content of lst",lst)
print("=====OR=====")
ls=[]
n=int(input("Enter how many values u want to be list:"))
if(n<=0):

```

```
        print("{} is invalid input:".format(n))
else:
    print("Enter {} values:".format(n))
    for i in range(1,n+1):
        val=int(input())
        ls.append(val)
else:
    print("content of lst",ls)
```

```
#listcompre1.py
print("Enter List of values separated by space:") # [10,2,222,50,10,4,55,-3,0,22]
lst= [float(val) for val in input().split() ]
print("content of lst",lst) # [3.0, 5.0, 2.0, 7.0, 3.0, 9.0, 6.0]
print("-"*40)
newlst=[]
for val in lst:
    newlst.append(val*2)
else:
    print("new list=",newlst)
print("-"*40)
print("=====OR=====")
newlst=[ val*2 for val in lst]
print("new list=",newlst)
```

```
#This program accept list of values dynamically by using list comprehension and
sort them
#listsortcompre.py
def disp(k):
    print("-"*40)
    for val in k:
        print("\t{}".format(val))
    else:
        print("-"*40)

def sortelements(lst):
    print("Original Elements")
    disp(lst)
    lst.sort()
```

```
print("Sorted Elements in Ascending Order")
disp(lst)
lst.reverse() # or lst.sort(reverse=True)
print("Sorted Elements in Descending Order")
disp(lst)

#main program
print("Enter List of values separated by space:") # [10,2,222,50,10,4,55,-3,0,22]
lst= [ int(val) for val in input().split() ]
sortelements(lst)
```

```
#This program accept list of names dynamically by using list comprehension and
sort them
#listsortcompre1.py
def disp(k):
    print("-"*40)
    for val in k:
        print("\t{}".format(val))
    else:
        print("-"*40)

def sortelements(lst):
    print("Original Elements")
    disp(lst)
    lst.sort()
    print("Sorted Elements in Ascending Order")
    disp(lst)
    lst.reverse() # or lst.sort(reverse=True)
    print("Sorted Elements in Descending Order")
    disp(lst)

#main program
print("Enter List of Names separated by comma:")
lst= [ str(val) for val in input().split(",") ]
sortelements(lst)
```

Parameters and Arguments

=>Parameters are classified into two types. They are

- a) Formal Parameters
 - b) Local Parameters (or) Variables
-

=>Formal parameters are those, which are used in Function Heading and they are used for storing the inputs which are coming from Function calls.

=>Local Parameters (or) Variables are those which are used in Function Body and they are used for storing Temporary results.

=>Hence both Formal Parameters and Local Parameters (or) Variables are available in the context Function Definition.

Examples:

```
def compute(a,b,c): # here 'a', 'b', 'c' are called formal Parameters  
    d=a+b*c # Here 'd' is called Local Parameters / Variable
```

=>Arguments are those which are used in Function Call and they may be in the form variables or values.

Example:

```
#main program  
x=10  
y=20  
z=30
```

```
compute(x,y,z) # here 'x','y','z' are called Arguments  
compute(100,200,300) # here 100,200 and 300 are called  
Argument values.
```

=>The relation between arguments and formal parameters is that "all values of Arguments are passing to Formal parameters".

Types of Parameters and Arguments

=>Based on passing arguments values to Parameters , the arguments are classified into 5 types.

- 1) Positional Arguments / Parameters
 - 2) Default Arguments / Parameters
 - 3) Key word Arguments / Parameters
 - 4) Variable Length Arguments / Parameters
 - 5) Key Word Variable Length Arguments / Parameters
-

1) Positional Arguments (or) Parameters

=>The Concept of Positional Parameters (or) arguments says that "The Number of Arguments(Actual arguments) 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 Positional Arguments (or) 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.

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 Positional 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 "Positional parameters"

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

```
#posparamex1.py
```

```
def dispstudinfo(stno,sname,marks):
    print("\t{}\t{}\t{}".format(stno,sname,marks))
#main program
print("*50")
print("\tStno\tName\tMarks")
print("*50")
dispstudinfo(10,"RS",33.33)
dispstudinfo(20,"DR",33.33)
dispstudinfo(30,"RR",43.33)
dispstudinfo(40,"SR",23.33)
print("*50")
```

```
#posparamex2.py
```

```
def dispstudinfo(stno,sname,marks,crs):
    print("\t{}\t{}\t{}\t{}".format(stno,sname,marks,crs))

#main program
print("*50")
print("\tStno\tName\tMarks\tCourse")
print("*50")
dispstudinfo(10,"RS",33.33,"PYTHON")
dispstudinfo(20,"DR",33.33,"PYTHON")
dispstudinfo(30,"RR",43.33,"PYTHON")
dispstudinfo(40,"KV",63.33,"PYTHON")
dispstudinfo(50,"WR",93.93,"PYTHON")
dispstudinfo(60,"PR",83.33,"PYTHON")
print("*50")
```

```
#defaultparamex1.py
```

```
def dispstudinfo(stno,sname,marks,crs="PYTHON"):
    print("\t{}\t{}\t{}\t{}".format(stno,sname,marks,crs))
```

```
#main program
print("*50)
print("\tStno\tName\tMarks\tCourse")
print("*50)
dispstudinfo(10,"RS",33.33)
dispstudinfo(20,"DR",33.33)
dispstudinfo(30,"RR",43.33)
dispstudinfo(40,"KV",63.33)
dispstudinfo(50,"WR",93.93)
dispstudinfo(60,"PR",83.33)
dispstudinfo(70,"SN",93.33,"JAVA")
dispstudinfo(60,"AJ",73.33)
print("*50)
```

```
#defaultparamex2.py
```

```
def dispstudinfo(stno,sname,marks,crs="PYTHON",city="HYD"):
    print("\t{}\t{}\t{}\t{}\t{}".format(stno,sname,marks,crs,city))
```

```
#main program
print("*50)
print("\tStno\tName\tMarks\tCourse\City")
print("*50)
dispstudinfo(10,"RS",33.33)
dispstudinfo(20,"DR",33.33)
dispstudinfo(30,"RR",43.33)
dispstudinfo(40,"KV",63.33)
dispstudinfo(50,"WR",93.93)
dispstudinfo(60,"PR",83.33)
dispstudinfo(70,"SN",93.33,"JAVA")
dispstudinfo(80,"AJ",73.33)
dispstudinfo(90,"RR",33.33,"JAVA")
dispstudinfo(35,"KB",33.33,"DJ","Bang")
print("*50)
```

```
#defaultparamex3.py
def disp(a=10,b=20,c=30):
    print("{}+{}+{}={}".format(a,b,c,a+b+c))
```

```
#main program
disp()
disp(100)
disp(1,2)
disp(1,2,3)
```

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

```
#kwdparamex1.py  
def disp(a,b,c):  
    print("\t{}\t{}\t{}".format(a,b,c))  
  
#main program  
print("*50)  
print("\ta\tb\tc")  
print("*50)  
disp(10,20,30)  
disp(c=30,b=20,a=10)  
disp(b=20,a=10,c=30)  
disp(10,c=30,b=20)  
disp(10,20,c=30)  
#disp(c=30,10,20)----error---SyntaxError: positional argument follows keyword  
argument  
print("*50)
```

```
#kwdparamex2.py  
def disp(a,b,c,crs="PYTHON"):  
    print("\t{}\t{}\t{}\t{}".format(a,b,c,crs))  
  
#main program  
print("*50)  
print("\ta\tb\tc\tcrs")  
print("*50)  
disp(10,20,30)  
disp(c=30,b=20,a=10)
```

```
disp(b=20,a=10,c=30)
disp(10,c=30,b=20)
disp(10,20,c=30)
#disp(c=30,10,20)----error---SyntaxError: positional argument follows keyword
argument
disp(c=30,a=10,b=20)
disp(crs="Java",c=30,a=10,b=20)
#disp(10,20,crs="DSC",30) ----error---SyntaxError: positional argument follows
keyword argument
disp(10,20,30,crs="JAVA")
disp(10,20,30,"Django")
print("=*50)
```

```
#kwparamex3.py
def studinfo(sno,sname,marks,crs="Python"):
    print("\t{}\t{}\t{}\t{}".format(sno,sname,marks,crs))

#main program
print("=*50)
print("\tstno\tName\tMarks\tCrs")
print("=*50)
studinfo(10,"RS",77.77)
studinfo(20,marks=22.22,sname="DR")
studinfo(marks=42.22,sno=30,sname="DR")
studinfo(crs="Django",marks=11.11,sname="TR",sno=40)
#studinfo(crs="Django",50,"MC",44.44) SyntaxError: positional argument follows
keyword argument
print("=*50)
```

4) Variables Length Parameters (or) arguments

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

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

Syntax for function definition with Variables Length Parameters:

```
def functionname(list of 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 Positional Argument Value(s)

#nonvarlenargex1.py---

#This program will not execute as it is shown below , because PVM remembers latest function definition only.

```
def dispvalues(a):
    print("{}".format(a))
```

```
def dispvalues(a,b):
    print("{}\t{}".format(a,b))
```

```
def dispvalues(a,b,c):
    print("{}\t{}\t{}".format(a,b,c))
```

```
def dispvalues(a,b,c,d):
    print("{}\t{}\t{}\t{}".format(a,b,c,d))
```

#main program

```
dispvalues(10) # Function call-1
dispvalues(10,20) # Function call-2
dispvalues(10,20,30) # Function call-3
dispvalues(10,20,30,40) # Function call-4
```

#nonvarlenargex2.py

```
def dispvalues(a): # Function Def-1
    print("{}".format(a))
```

```
dispvalues(10) # Function call-1
```

```
def dispvalues(a,b): # Function Def-2
    print("{}\t{}".format(a,b))
```

```
dispvalues(10,20) # Function call-2
```

```
def dispvalues(a,b,c): # Function Def-3
    print("{}\t{}\t{}".format(a,b,c))
```

```
dispvalues(10,20,30) # Function call-3
```

```
def dispvalues(a,b,c,d): # Function Def-4
    print("{}\t{}\t{}\t{}".format(a,b,c,d))
```

```
dispvalues(10,20,30,40) # Function call-4
```

```
def dispvalues(a,b,c,d,e): # Function Def-5
    print("{}\t{}\t{}\t{}\t{}".format(a,b,c,d,e))
```

```
dispvalues("RS","DR","MC","TR","SS") # Function call-5
```

```
#varlenargex1.py
def dispvalues(*a): # here *a is called variable argument and whose type is tuple
    print("*50")
    print("\nType of a:{} and length={}".format(type(a), len(a)))
    for val in a:
        print("\t{}".format(val))
    else:
        print("*50")
```

```
#main program
dispvalues(10) # Function call-1
dispvalues(10,20) # Function call-2
dispvalues(10,20,30) # Function call-3
dispvalues(10,20,30,40) # Function call-4
dispvalues("RS","DR","MC","TR","SS") # Function call-5
```

```
#varlenargex2.py
def dispvalues(*a):
    for val in a:
        print("{}".format(val),end=" ")
    else:
        print()
```

```
#main program
print("*50")
dispvalues(10) # Function call-1
dispvalues(10,20) # Function call-2
dispvalues(10,20,30) # Function call-3
dispvalues(10,20,30,40) # Function call-4
dispvalues("RS","DR","MC","TR","SS") # Function call-5
print("*50")
```

```
#Write a python program which will compute sum of variable number of numerical values
```

```
#varlenargex3.py
def findsum(sname, *nums):
    print("*50")
    print("My Name is {}".format(sname))
    s=0
    for val in nums:
        print("\t{}".format(val))
        s=s+val
    else:
```

```

        print("sum={}".format(s))
        print("*50)

#main program
findsum("Mahdev",10,20,30,40,50,60)
findsum("pooja",10,20)
findsum("supriya",10,20,30)
findsum("priyanka",12.3,34.5,5.6,-5.7)
findsum("venkatesh",1.2,13)
findsum("pavan",23,45,67,-56,78,12,-34,56)
findsum("nani")

#Write a python program which will compute sum of variable number of numerical
values
#varlenargex4.py
def findsum(sname, *nums, crs="PYTHON", city="hyd"):
    print("*50")
    print("My Name is '{}' doing '{}' course AND LIVING
IN:{}".format(sname, crs, city))
    s=0
    for val in nums:
        print("\t{}".format(val))
        s=s+val
    else:
        print("sum={}".format(s))
        print("*50")

#main program
findsum("Mahdev",10,20,30,40,50,60)
findsum("pooja",10,20)
findsum("supriya",10,20,30)
findsum("priyanka",12.3,34.5,5.6,-5.7)
findsum("venkatesh",1.2,13)
findsum("pavan",23,45,67,-56,78,12,-34,56)
findsum("nani")
#findsum(crs="Django",sname="KVR",5,6,7)--SyntaxError: positional argument
follows keyword argument
#findsum("KVR",5,6,7,"Django") TypeError: unsupported operand type(s) for +: 'int'
and 'str'
findsum("KVR",5,6,7,crs="Django")
findsum("Pranav",15,16,17,crs="Data Science",city="AP")
findsum("Ajay-Ankit",15,16,city="Delhi-BANG")

```

5) Key Word Variables Length Parameters (or) arguments

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

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

Syntax for function definition with Keyword Variables Length Parameters:

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

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

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

```
#kwdvarlenparamex1.py
def dispinfo( **a): # here **a is called keyword variable length argrument /
Parameter--dict
    print("-"*50)
    print("type of a=",type(a))
    print("No. of (Key,Value) ={}".format(len(a)))
    for k,v in a.items():
        print("\t{}--->{}".format(k,v))
    else:
        print("-"*50)
#main program
dispinfo(sno=10,sname="RS",marks=33.33,cname="PSF")
dispinfo(eno=111,ename="DR",sal=4.5)
dispinfo(crs1="Python1",crs2="Django")
dispinfo(author="Ritche")
dispinfo()
```

#This program computes total marks different student studying in different classes who secured in different subjects.

```
#kwdvarlenparamex2.py
def findtotalmarks(sname,cls,**submarks):
    print("*"*50)
    print("\tS t u d e n t \t M a r k s \t R e p o r t:")
    print("*"*50)
    print("\tStudent Name:{}\tClass Name:{}".format(sname,cls))
    print("-"*50)
    tm=0
```

```

print("\tSubject Name\tMarks")
print("-"*50)
for sub,marks in submarks.items():
    print("\t{}\t{}".format(sub,marks))
    tm=tm+marks
else:
    print("-"*50)
    print("Total Marks={}".format(tm))
    print("-"*50)
print("*"*50)
#main program
findtotalmarks("Santosh","X",Sci=70,Soc=77,Maths=88,Hindi=55)
findtotalmarks("Umesh","XII",Maths=75,Phy=56,Che=55)
findtotalmarks("Venkatesh","B.Tech(Mech)",sub1=50,sub2=88,sub3=77,sub4=55)
findtotalmarks("Mahima","P.hD(DAA)", Core=90,Rm=56)
findtotalmarks("Rossum","Research")

```

Global Variables and Local Variables

=>The purpose of Global variables is that "To store Common Values for Different Function Calls"

=>Global Variables are those, which 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, which are defined within the Function Body

=>Syntax:-

FileName.py

```

var1=val1
var2=val2 # here var1,var2...are called global variables
def functionname1(list of formal params if any):
    -----
        var5=val # here var5 is called Local Variable
    -----
def functionname2(list of formal params if any):
    -----
        var6=val # here var6 is called Local Variable
    -----
def functionname-n(list of formal params if any):
    -----
        var7=val # here var7 is called Local Variable

```

```

#globlalvarex1.py
lang="PYTHON" # here 'lang' is called Global Variable
def learnDataSci():
    crs1="Data Science" # crs1 is called local variable
    print(" To do coding in '{}', we need '{}' language.".format(crs1,lang))
    #print(crs2,crs3) can't access crs2 and crs3 bcoz they are local in other
functions

```

```
def learnAI():
    crs2="AI" # crs2 is called local variable
    print(" To do coding in '{}', we need '{}' language:".format(crs2,lang))
    #print(crs1,crs3) can't access crs1 and crs3 bcoz they are local in other
functions

def learnML():
    crs3="ML" # crs3 is called local variable
    print(" To do coding in '{}', we need '{}' language:".format(crs3,lang))
    #print(crs1,crs2) can't access crs1 and crs2 bcoz they are local in other
functions
#main program
learnDataSci()
learnAI()
learnML()
```

#globalvarex2.py

```
def learnDataSci():
    crs1="Data Science" # crs1 is called local variable
    print(" To do coding in '{}', we need '{}' language:".format(crs1,lang))
    #print(crs2,crs3) can't access crs2 and crs3 bcoz they are local in other
functions
def learnAI():
    crs2="AI" # crs2 is called local variable
    print(" To do coding in '{}', we need '{}' language:".format(crs2,lang))
    #print(crs1,crs3) can't access crs1 and crs3 bcoz they are local in other
functions

def learnML():
    crs3="ML" # crs3 is called local variable
    print(" To do coding in '{}', we need '{}' language:".format(crs3,lang))
    #print(crs1,crs2) can't access crs1 and crs2 bcoz they are local in other
functions
#main program
lang="PYTHON" # here 'lang' is called Global Variable
learnDataSci()
learnAI()
```

#globalvarex3.py

```
def learnDataSci():
    crs1="Data Science" # crs1 is called local variable
    print(" To do coding in '{}', we need '{}' language:".format(crs1,lang))
    #print(crs2,crs3) can't access crs2 and crs3 bcoz they are local in other
functions
lang="PYTHON" # here 'lang' is called Global Variable
def learnAI():
    crs2="AI" # crs2 is called local variable
    print(" To do coding in '{}', we need '{}' language:".format(crs2,lang))
    #print(crs1,crs3) can't access crs1 and crs3 bcoz they are local in other
functions

def learnML():
```

```
crs3="ML" # crs3 is called local variable
print(" To do coding in '{}', we need '{}' language:".format(crs3,lang))
#print(crs1,crs2) can't access crs1 and crs2 bcoz they are local in other
functions
#main program
learnDataSci()
learnAI()
learnML()
```

```
#globalvarex4.py
def learnDataSci():
    crs1="Data Science" # crs1 is called local variable
    print(" To do coding in '{}', we need '{}' language:".format(crs1,lang))
    #print(crs2,crs3) can't access crs2 and crs3 bcoz they are local in other
functions
def learnAI():
    crs2="AI" # crs2 is called local variable
    print(" To do coding in '{}', we need '{}' language:".format(crs2,lang))
    #print(crs1,crs3) can't access crs1 and crs3 bcoz they are local in other
functions
def learnML():
    crs3="ML" # crs3 is called local variable
    print(" To do coding in '{}', we need '{}' language:".format(crs3,lang))
    #print(crs1,crs2) can't access crs1 and crs2 bcoz they are local in other
functions
#main program
#learnDataSci()
#learnAI()
learnML()
lang="PYTHON" # here 'lang' is called Global Variable
```

global key word

=>When we want MODIFY the GLOBAL VARIABLE values in side of function defination 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  
-----
```

```
#globalkwdx1.py  
a=10 # Global Variable  
def fun1():  
    print("Val of a in fun1()=",a) # Accessing Global Variable Value  
def fun2():  
    print("Val of a in fun2()=",a) # Accessing Global Variable Value  
def fun3():  
    print("Val of a in fun3()=",a) # Accessing Global Variable Value  
  
#main program  
fun1()  
fun2()  
fun3()
```

```
#globalkwdx2.py  
a=10  
def increment():  
    global a  
    a=a+1  
    print("val of a in fun1()=",a) # 11  
  
def multiplyval():  
    global a  
    a=a*2  
    print("val of a in multiplyval()=",a) #22  
#main program  
print("Val of a in main program before increment()=",a) # 10  
increment()  
print("Val of a in main program after increment()=",a) # 11  
multiplyval()  
print("Val of a in main program after multiplyval()=",a) # 22
```

```
#globalkwdx3.py  
a=10  
b=20 # here 'a' and 'b' are called Global Variables  
def operation():  
    global a,b  
    a=a+1  
    b=b+1
```

```
def updatevalues():
    global a,b
    c=a*b
    print("Val of c in updatevalues()=",c)
    a=a+2
    b=b+2

#main program
print("Val of a before operation()",a) # 10
print("Val of b before operation()",b) # 20
operation()
print("Val of a after operation()",a) # 11
print("Val of b after operation()",b) # 21
updatevalues()
print("Val of a after updatevalues()",a) # 13
print("Val of b after updatevalues()",b) # 23
```

Anonymous or Lambda Functions

- =>Anonymous Functions are those which does not contain any name explicitly.
 - =>The purpose of Anonymous Functions is that "To Perform Instant Operations".
Instant Operations are those which are no longer interested to carry in further programs.
 - =>Anonymous Functions contains single executable statement
 - =>Anonymous Functions automatically or implicitly returns the value(No Need to use return statement to return the value).
 - =>To define Anonymous Functions, we use a key word "lambda" and hence Anonymous Functions are called Lambda Functions.
-

=>Syntax:

varname=lambda params-list : Expression

Explanation:

- =>varname is an object of type of <class 'function'> and varname is itself indirectly acts function name.
 - =>lambda is a keyword used for defining Anonymous Functions
 - =>Params-list represents list of variable names used for storing the input values coming function calls.
 - =>Expression represents single executable statement and provides solution for Instant Operation.
-

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

```

sumop=lambda a,b: a+b # anonymous Function definition

#main program
res=addop(10,20)
print("type of addop=",type(addop))# type of addop= <class 'function'>
print("sum=",res)
print("====")
print("type of sumop=",type(sumop)) # type of sumop= <class 'function'>
x1=float(input("Enter First value:"))
x2=float(input("Enter Second value:"))
res1=sumop(x1,x2)
print("sum=",res1)



---


#Write a python program which will calculate all types of academic operations (Use menu driven application along with anonymous function).

#aopmenuop1.py
# anonymous function definions
import sys
addop=lambda a,b:a+b
subop=lambda a,b:a-b
mulop=lambda a,b:a*b
divop=lambda a,b:a/b
floordivop=lambda k,v:k//v
modop=lambda k,v:k%v
expop=lambda k,v:k**v

def aopmenu(): # Normal Function Definition
    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.Floor Division")
    print("\t6.Modulo Division")
    print("\t7.Exponentiation")
    print("\t8.Exit")
    print("*50")

#main program
aopmenu()
ch=int(input("Enter Ur Choice:"))
match (ch):
    case 1:
        x1=float(input("Enter First Value for Addition:"))
        x2=float(input("Enter Second Value for Addition:"))
        res=addop(x1,x2)
        print("Sum({},{})={}".format(x1,x2,res))
    case 2:

```

```

x1=float(input("Enter First Value for Substraction:"))
x2=float(input("Enter Second Value for Substraction:"))
res=subop(x1,x2)
print("Sub({},{})={}".format(x1,x2,res))

case 3:
    x1=float(input("Enter First Value for Multiplication:"))
    x2=float(input("Enter Second Value for Multiplication:"))
    res=mulop(x1,x2)
    print("Mul({},{})={}".format(x1,x2,res))

case 4:
    x1=float(input("Enter First Value for Division:"))
    x2=float(input("Enter Second Value for Division:"))
    res=divop(x1,x2)
    print("Div({},{})={}".format(x1,x2,res))

case 5:
    x1=float(input("Enter First Value for Floor Division:"))
    x2=float(input("Enter Second Value for Floor Division:"))
    res=floordivop(x1,x2)
    print("FloorDiv({},{})={}".format(x1,x2,res))

case 6:
    x1=float(input("Enter First Value for Modulo Division:"))
    x2=float(input("Enter Second Value for Modulo Division:"))
    res=modop(x1,x2)
    print("Mod({},{})={}".format(x1,x2,res))

case 7:
    x1=float(input("Enter Base:"))
    x2=float(input("Enter Power:"))
    res=expop(x1,x2)
    print("expop({},{})={}".format(x1,x2,res))

case 8:
    print("Thanks for this program")
    sys.exit()

case _:
    print("Ur Selection of Operation is wrong!--try again")

```

#Write a python program which will calculate all types of academic operations (Use menu driven application along with anonymous function).

```

#aopmenuop1.py
# anonymous function definitions
import sys
addop=lambda a,b:a+b
subop=lambda a,b:a-b
mulop=lambda a,b:a*b
divop=lambda a,b:a/b
floordivop=lambda k,v:k//v
modop=lambda k,v:k%v
expop=lambda k,v:k**v

```

```

def aopmenu(): # Normal Function Definition
    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.Floor Division")
    print("\t6.Modulo Division")
    print("\t7.Exponentiation")
    print("\t8.Exit")
    print("*50")

#main program
aopmenu()
ch=int(input("Enter Ur Choice:"))
match (ch):
    case 1:
        x1=float(input("Enter First Value for Addition:"))
        x2=float(input("Enter Second Value for Addition:"))
        res=addop(x1,x2)
        print("Sum({},{})={}".format(x1,x2,res))
    case 2:
        x1=float(input("Enter First Value for Substration:"))
        x2=float(input("Enter Second Value for Substration:"))
        res=subop(x1,x2)
        print("Sub({},{})={}".format(x1,x2,res))
    case 3:
        x1=float(input("Enter First Value for Multiplication:"))
        x2=float(input("Enter Second Value for Multiplication:"))
        res=mulop(x1,x2)
        print("Mul({},{})={}".format(x1,x2,res))
    case 4:
        x1=float(input("Enter First Value for Division:"))
        x2=float(input("Enter Second Value for Division:"))
        res=divop(x1,x2)
        print("Div({},{})={}".format(x1,x2,res))
    case 5:
        x1=float(input("Enter First Value for Floor Division:"))
        x2=float(input("Enter Second Value for Floor Division:"))
        res=floordivop(x1,x2)
        print("FloorDiv({},{})={}".format(x1,x2,res))
    case 6:
        x1=float(input("Enter First Value for Modulo Division:"))
        x2=float(input("Enter Second Value for Modulo Division:"))
        res=modop(x1,x2)
        print("Mod({},{})={}".format(x1,x2,res))
    case 7:

```

```

x1=float(input("Enter Base:"))
x2=float(input("Enter Power:"))
res=expop(x1,x2)
print("expop({},{})={}".format(x1,x2,res))

case 8:
    print("Thanks for this program")
    sys.exit()

case _:
    print("Ur Selection of Operation is wrong!--try again")

```

```

#Write a python program which will calculate all types of academic operations (Use menu driven application along with anonymous function).
#aopmenuop2.py
# anonymous function definitions
import sys
addop=lambda a,b:a+b
subop=lambda a,b:a-b
mulop=lambda a,b:a*b
divop=lambda a,b:a/b
floordivop=lambda k,v:k//v
modop=lambda k,v:k%v
expop=lambda k,v:k**v

def aopmenu(): # Normal Function Definition
    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.Floor Division")
    print("\t6.Modulo Division")
    print("\t7.Exponentiation")
    print("\t8.Exit")
    print("*50")

#main program
while(True):
    aopmenu()
    ch=int(input("Enter Ur Choice:"))
    match (ch):
        case 1:

```

```

x1=float(input("Enter First Value for Addition:"))
x2=float(input("Enter Second Value for Addition:"))
res=addop(x1,x2)
print("Sum({},{})={}".format(x1,x2,res))

case 2:
    x1=float(input("Enter First Value for
Substraction:"))
    x2=float(input("Enter Second Value for
Substraction:"))
    res=subop(x1,x2)
    print("Sub({},{})={}".format(x1,x2,res))

case 3:
    x1=float(input("Enter First Value for
Multiplication:"))
    x2=float(input("Enter Second Value for
Multiplication:"))
    res=mulop(x1,x2)
    print("Mul({},{})={}".format(x1,x2,res))

case 4:
    x1=float(input("Enter First Value for Division:"))
    x2=float(input("Enter Second Value for Division:"))
    res=divop(x1,x2)
    print("Div({},{})={}".format(x1,x2,res))

case 5:
    x1=float(input("Enter First Value for Floor
Division:"))
    x2=float(input("Enter Second Value for Floor
Division:"))
    res=floordivop(x1,x2)
    print("FloorDiv({},{})={}".format(x1,x2,res))

case 6:
    x1=float(input("Enter First Value for Modulo
Division:"))
    x2=float(input("Enter Second Value for Modulo
Division:"))
    res=modop(x1,x2)
    print("Mod({},{})={}".format(x1,x2,res))

case 7:
    x1=float(input("Enter Base:"))
    x2=float(input("Enter Power:"))
    res=expop(x1,x2)
    print("expop({},{})={}".format(x1,x2,res))

case 8:
    print("Thanks for this program")
    sys.exit()

case _:
    print("Ur Selection of Operation is wrong!--try
again")

```

```

#Write a python program which will calculate all types of academic operations (Use menu driven application along with anonymous function).
#aopmenuop3.py
# anonymous function definitions
import sys
addop=lambda a,b:a+b
subop=lambda a,b:a-b
mulop=lambda a,b:a*b
divop=lambda a,b:a/b
floordivop=lambda k,v:k//v
modop=lambda k,v:k%v
expop=lambda k,v:k**v

def readvalues(op):
    x1=float(input("Enter First Value for {}:".format(op)))
    x2=float(input("Enter Second Value for {}:".format(op)))
    return x1,x2

def aopmenu(): # Normal Function Definition
    print("*50")
    print("\tA r i t h m e t i c \t 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.Floor Division")
    print("\t6.Modulo Division")
    print("\t7.Exponentiation")
    print("\t8.Exit")
    print("*50")

#main program
while(True):
    aopmenu()
    ch=int(input("Enter Ur Choice:"))
    match (ch):
        case 1:
            x1,x2=readvalues("Addition")
            res=addop(x1,x2)
            print("Sum({},{})={}".format(x1,x2,res))
        case 2:
            a,b=readvalues("Substraction")
            res=subop(a,b)
            print("Sub({},{})={}".format(a,b,res))
        case 3:
            x1,x2=readvalues("Multiplication")
            res=mulop(x1,x2)

```

```

        print("Mul({},{})={}".format(x1,x2,res))
case 4:
    x1,x2=readvalues("Division")
    res=divop(x1,x2)
    print("Div({},{})={}".format(x1,x2,res))
case 5:
    x1,x2=readvalues("Floor Division")
    res=floordivop(x1,x2)
    print("FloorDiv({},{})={}".format(x1,x2,res))
case 6:
    x1,x2=readvalues("Modulo Division")
    res=modop(x1,x2)
    print("Mod({},{})={}".format(x1,x2,res))
case 7:
    x1=float(input("Enter Base:"))
    x2=float(input("Enter Power:"))
    res=expop(x1,x2)
    print("expop({},{})={}".format(x1,x2,res))
case 8:
    print("Thanks for this program")
    sys.exit()
case _:
    print("Ur Selection of Operation is wrong!--try
again")

```

```

#Program accepting two integer values and find biggest by using anonymous
functions
#bigex1.py
big=lambda a,b: a if a>b else b # anonymous functions
small=lambda a,b: a if a<b else b # anonymous functions
#main program
bv=big(float(input("Enter First Value:")),float(input("Enter Second Value:")) )
print("Biggest={}".format(bv))
sv=small(float(input("Enter First Value:")),float(input("Enter Second Value:")) )
print("smallest={}".format(sv))

```

```

#Program accepting three integer values and find biggest by using anonymous
functions
#bigex2.py
big=lambda a,b,c: a if(a>b) and (a>c) else b if (b>c) and (b>a) else c

#main program
print("Enter Three values:")
bv=big(int(input()), int(input()), int(input()) )
print("Biggest=",bv)

```

```

#Program accepting three integer values and find biggest by using anonymous
functions
#bigex3.py
big=lambda a,b,c: "ALL VALUES EQUAL" if (a==b) and (b==c) else a if(a>b) and (a>c)
else b if (b>c) and (b>a) else c

#main program
print("Enter Three values:")
bv=big(int(input()), int(input()), int(input()))
print("Biggest=",bv)


---


#Program accepting list of integer values and find biggest and sammlest by using
anonymous functions
#bigsmall.py

big= lambda lst:max(lst) # anonymous functions
small=lambda hyd: min(hyd) # anonymous functions

#main program
print("Enter List of Values separated by space:")
lst=[int(val) for val in input().split()]
bv=big(lst)
sv=small(lst)
print("biggest({})={}".format(lst,bv))
print("smallest({})={}".format(lst,sv))


---


=====
```

Differences between Normal Functions and Anonymous Functions

- =>Normal Functions always used for performing Long Term Operation (nothing but we can re-use in every part of project), where Anonymous Functions are those which are meant for performing Instant Operations.
- =>Normal Functions definitions contains Block of statements where Anonymous Functions contains single executable statement.
- =>Normal Function Definition starts with "def" keyword where Anonymous Functions definition starts with "lambda"
- =>To return the value from normal function, it is mandatory to use return statement. where as Anonymous Functions returns the value automatically / implicitly (no need to use return statement).
- => Syntax for normal function:


```
def  functionname (list of formal params):
-----
```

Block of statements

```
-----
```
- =>Syntax for Anonymous Functions:


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

Special Functions in Python

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

- 1) filter()
 - 2) map()
 - 3) reduce()
-

1) filter()

=>filter() is used for filtering out some elements from given list of elements by applying to given function.

=>**Syntax:**

varname=filter(Functionname,Iterable object)

Explanation:

=>varname is an object of <class,'filter'> and we can convert into any iterable object type by using Iterable type casting functions.

=>Function name can be either Normal function or anonymous function.

=>Iterable object can be sequence, list , set or dict type.

=>The execution process of filter() is that "Each element of Iterable object passing to specified function, the function decides True or False based on condition, if the function returns True then the corresponding element will be filtered and if the function returns False then the corresponding element will be neglected (not filtered)". This Process will be continued until all elements of iterable completed .

#This program takes list of values and extract only +ve values

#FilterEx1.py

```
def positive(n):
    if n>0:
        return True
    else:
        return False
```

```
def negative(n):
```

```
if(n<0):
    return True
else:
    return False

#main program
lst=[10,-20,30,0,40,-50,-60,70,23]
obj1=filter(positive,lst)
print("type of obj=",type(obj1)) #<class 'filter'>
pslist=list(obj1)
print("Original Elements=",lst)
print("Positive Elements=",pslist)
obj2=filter(negative,lst)
ngtpl=tuple(obj2)
print("Negative Elements=",ngtpl)
```

#This program takes list of values and extract only +ve values

#FilterEx2.py

```
def positive(n):
    if n>0:
        return True
    else:
        return False

def negative(n):
    if(n<0):
        return True
    else:
        return False
```

```
#main program
print("Enter list of values separated by space:")
lst=[int(val) for val in input().split()]
obj1=filter(positive,lst)
print("type of obj=",type(obj1)) #<class 'filter'>
pslist=list(obj1)
print("Original Elements=",lst)
print("Positive Elements=",pslist)
obj2=filter(negative,lst)
ngtpl=tuple(obj2)
print("Negative Elements=",ngtpl)
```

#This program takes list of values and extract only +ve values

#FilterEx3.py

```
positive=lambda x: x>0
negative=lambda k : k<0
#main program
print("Enter list of values separated by space:")
lst=[int(val) for val in input().split()]
pslist=list(filter(positive,lst))
nglist=tuple(filter(negative,lst))
print("-----")
print("Original Elements=",lst)
```

```
print("Positive Elements=",pslist)
print("Negative Elements=",nglist)
print("-----")
```

#This program takes list of values and extract only +ve values

#FilterEx4.py

```
print("Enter list of values separated by space:")
lst=[int(val) for val in input().split()]
pslist=list(filter(lambda x: x>0,lst))
nglist=tuple(filter(lambda k : k<0,lst))
print("-----")
print("Original Elements=",lst)
print("Positive Elements=",pslist)
print("Negative Elements=",nglist)
print("-----")
```

#This program accepts list of values and extract Even Numbers

#FilterEx5.py

```
print("Enter list values separated by space:")
lst=[int(val) for val in input().split()] # lst=[4 6 7 12 6 34 81]
evenlst=tuple(filter(lambda n : n%2==0,lst))
print("Original list=",lst)
print("Even list=",evenlst)
```

#write a python program which will accept a line of text and filter vowels and print their count

#FilterEx6.py

```
s=input("Enter line of text:")
vowels=list(filter(lambda ch: ch in ['a','e','o','i','u','A','E','I','O','U'], s))
print("-----")
print("Given Line of Text={}".format(s))
print("Vowels={}".format(vowels))
print("Number of vowels={}".format(len(vowels)))
print("-----")
cons=list(filter(lambda ch: ch not in ['a','e','o','i','u','A','E','I','O','U'] and not ch.isspace()
and not ch.isdigit(), s))
print("consonants={}".format(cons))
print("Number of consonants={}".format(len(cons)))
print("-----")
```

#This program takes list of values and extract only +ve values

#nonFilterEx.py

```
def positivenegative(lst):
    pslist=[]
    nglist=[]
    for val in lst:
        if (val>0):
            pslist.append(val)
        else:
            nglist.append(val)
```

```
return pslist,nlist

#main program
print("Enter list of values separated by space:")
lst=[int(val) for val in input().split()]
psl,ngl=positivenegative(lst)
print("Orginal Elements:",lst)
print("Positive Elements:",psl)
print("Negative Elements:",ngl)
```

2) map()

=>The purpose of map() is that "To convert Old Iterable object elements into new Iterable object Elements by applying to the function".
=>In otherwords , map() converts old list elements into new list elements by applying to the function

=>Syntax:-

varname=map(Functionname,Iterable object)

=>varname is an object of <class,'map'> and we can convert into any iterable object type by using Iterable type casting functions.

=>Function name can be either Normal function or anonymous function.

=>Iterable object can be sequence, list , set or dict type.

=>The execution process of map() is that " Every element of iterable object sending to the specified function, function will execute business logic and returned the result from function and it will be placed in an object of type <class, map>". This Process will be continued until all elements of iterable object completed .

#This program accepts list of elements and gets its square list.

#mapex1.py

```
def square(n):
    result=n**2
    return result
```

#main program

```
print("Enter list of elements separated by space:")
oldlst=[int(val) for val in input().split()]
mapobj=map(square,oldlst)
print("\nType of mapobj={}".format(type(mapobj)))
sqrlst=list(mapobj)
print("Original Elements:{}".format(oldlst))
print("New Elements:{}".format(sqrlst))
```

#This program accepts list of elements and gets its square list.

#mapex2.py

```
print("Enter list of elements separated by space:")
oldlst=[int(val) for val in input().split()]
```

```
sqrlst=tuple(map(lambda val:val**2 , oldlst))
print("Original Elements:{}".format(oldlst))
print("New Elements:{}".format(sqrlst))
```

#This program accepts list of elements and gets its square list.

#mapex3.py

```
print("Enter list of elements separated by space:")
oldlst=[int(val) for val in input().split()]
sqrootlist=tuple(map(lambda val:val**0.5 , oldlst))
print("Original Elements:{}".format(oldlst))
print("New Elements:{}".format(sqrootlist))
```

#This program accepts list of elements and gets its square list.

#mapex4.py

```
def disp(lst):
    print("-"*40)
    for val in lst:
        print("%0.2f" %val)
    print("-"*40)
```

#mapex4.py

```
print("Enter list of elements separated by space:")
oldlst=[int(val) for val in input().split()]
sqrootlist=tuple(map(lambda val:val**0.5 , oldlst))
print("Original Elements:")
disp(oldlst)
print("New Elements:")
disp(sqrootlist)
```

#This program accepts list of elements and gets its square list.

#mapex5.py

```
print("Enter list of elements separated by space:")
oldlst=[int(val) for val in input().split()]
sqrootlist=tuple(map(lambda val:val**0.5 , oldlst))
print("-"*50)
print("Original Elements\tSquare root Elements:")
print("-"*50)
for on,sn in zip(oldlst,sqrootlist):
    print("\t{}\t{}".format(on,round(sn,3)))
print("-"*50)
```

#This program accepts list of elements and gets its square list and square root .

#mapex6.py

```
print("Enter list of elements separated by space:")
oldlst=[int(val) for val in input().split()]
sqrootlist=tuple(map(lambda val:val**0.5 , oldlst))
sqlist=tuple(map(lambda val:val**2 , oldlst))
print("-"*50)
```

```
print("Original\tSquareRoot\tSquare:")
print("-"*50)
for on,sqrtno,sqrno in zip(olilst,sqrootlist,sqlist):
    print("\t{}\t{}\t{}".format(on,round(sqrtno,2),sqrno))
print("-"*50)
```

#This program accepts list of Old Salaries of employees and gets new salary list

#mapex7.py

```
print("Enter list of old salaries of employees:")
oldsal=[float(val) for val in input().split()]
newsal=list(map(lambda sal:sal+sal*0.15,oldsal))
print("=*50")
print("Old Salaries \t New Salaries")
print("=*50")
for old,new in zip(oldsal,newsal):
    print("\t{}\t{}".format(old,round(new,3)))
print("=*50")
```

#This program accepts two lists of elements and find their sum.

#mapex8.py

```
print("Enter list of elements for First List")
lst1=[float(val) for val in input().split()]
print("Enter list of elements for Second List")
lst2=[float(val) for val in input().split()]
sumlst=list(map(lambda x,y:x+y,lst1,lst2))
print("=*50")
print("\tList1\tList2\tList1+List2")
print("=*50")
for x,y,z in zip(lst1,lst2,sumlst):
    print("\t{}\t{}\t{}".format(x,y,z))
print("=*50")
```

#write a python program which will accept list of employees salaries between 1000 to any salary . give 15% hike to those employees who is drawing more than 5000 and give 30 % hike to those employees whose drawing less than 5000.

#mapex9.py

```
print("Enter List of Salaries of Employee:")
orginalsal=[int(sal) for sal in input().split()]
salless5000=list(filter(lambda sal:sal<=5000 ,orginalsal))
hikesallessthan5000=list(map(lambda sal:sal+sal*(30/100),salless5000 ))
salmore5000=list(filter(lambda sal:sal>5000,orginalsal))
hikesalmorethan5000=list(map(lambda sal:sal+sal*(15/100), salmore5000 ))
print("=*50")
print("\tSal Less Than 5000\t Hike Salaries")
print("=*50")
for sal,hsal in zip(salless5000,hikesallessthan5000):
    print("\t\t{}\t\t{}".format(sal,hsal))
```

```

print("*50)
print("*50)
print("Sal More Than 5000 Hike Salaries")
print("*50)
for sal,hSal in zip(salmore5000,hikesalmorethan5000):
    print("\t\t{}\t\t{}".format(sal,hSal))
print("*50)

```

reduce()

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

=>Syntax:-

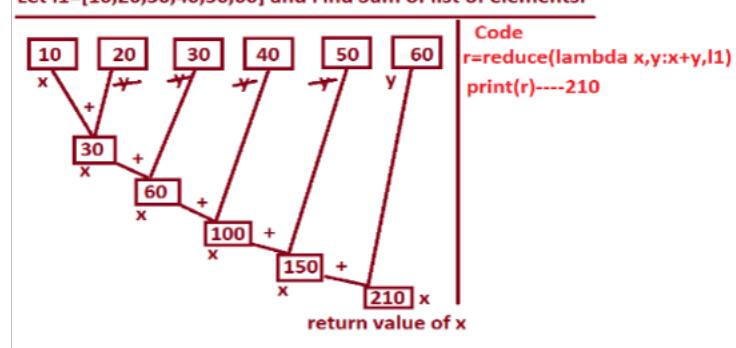
varname=reduce(function-name,iterable-object)

=>here varname is an object of int, float,bool,complex,str only

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

Internal Flow of reduce()

Let l1=[10,20,30,40,50,60] and Find Sum of list of elements.



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

step-2:- The function-name utilizes the values of First var and

Second var applied to the specified logic and obtains the result.

Step-3:- reduce () places the result of function-name in First variable and reduce() selects the succeeding element of Iterable object and places in second variable.

Step-4: repeat Step-2 and Step-3 until all elements completed in Iterable object and returns the result of First Variable

```

#This program accepts list of values and find their sum by using reduce()
#redceex1.py
import functools
print("Enter List of values separated by space:")
lst=[int(val) for val in input().split()]
result=functools.reduce(lambda a,b:a+b,lst)
print("\nList of elements=",lst)

```

```
print("Sum=",result)
```

```
#This program accepts list of values and find max and min from list of elements.
```

```
#reduceex2.py
```

```
import functools
```

```
print("Enter List of values separated by space:")
```

```
lst=[int(val) for val 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("\nList of elements={}".format(lst))
```

```
print("Biggest Element={}".format(big))
```

```
print("Smallest Element={}".format(small))
```

```
#This program accepts list of values and find max and min from list of elements.
```

```
#reduceex3.py
```

```
from functools import reduce
```

```
print("Enter List of values separated by space:")
```

```
lst=[int(val) for val in input().split()]
```

```
big=reduce(lambda x,y: x if x>y else y, lst)
```

```
small=reduce(lambda x,y: x if x<y else y, lst)
```

```
print("\nList of elements={}".format(lst))
```

```
print("Biggest Element={}".format(big))
```

```
print("Smallest Element={}".format(small))
```

```
#This program accepts list of Srings separated by comma and obtain single line.
```

```
#reduceex4.py
```

```
import functools
```

```
print("Enter List of values separated by comma:")
```

```
lst=[ str(val) for val in input().split(",")]
```

```
print("content of list=",lst)
```

```
singleline=functools.reduce(lambda x,y:x+" "+y,lst)
```

```
print("Result={}".format(singleline))
```

Iterators in python

- => Iterator is a kind of object which is a collection of other elements.
- => For example :- list and tuple are iterables .
- =>In Python , iter() converts Iterable objects (list ,tuple , set , frozenset....etc) into iterator Object and it takes less memory space and provides value when we request.
- =>An iterator is an object that contains countable number of values.
- =>An iterator is an object that can be iterated with all values.

Example: lst=[“apple”, “mango”, “kiwi”, “guava”]
 for frt in lst:
 print(frt)

- =>Here lst is by default iterable .
 - =>Programmatically ,to convert an object which contains multiples values 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 StopIteration when no value present in iterator object.
-

Example-1:

```
#iteratorex1.py
lst=["apple", "mango", "kiwi", "guava"]
for frt in lst:
    print(frt)
print("=====OR====")
itrlst=iter(lst)
print("we are in iteration part")
print("type of itrlst",type(itrlst))
print("content of list iterator:",itrlst)
print("-"*50)
while(True):
    try:
        print(next(itrlst))
    except StopIteration:
        break
```

OUTPUT:

```
apple  
mango  
kiwi  
guava
```

=====OR=====

we are in iteration part

type of itelst <class 'list_iterator'>

content of list iterator: <list_iterator object at 0x0000020B02DB5660>

```
apple  
mango  
kiwi  
guava
```

Decorators

=>A Decorator is a function that's provides additional Functionality to the existing Functionality of normal function.

=> A decorator takes the result of normal function ,Modifies the result and returns modified result.

=>A decorators always takes normal function name as an argument.

EXAMPLE-1:

Program for generating square of no without using decorator:

```
#program for getting square of given number
```

```
#non_decorex1.py
```

```
def getvalues(): #normal function
```

```
    return(float(input("Enter a number:")))
```

```
def square(): # provides square for a number return by getvalues()
```

```
    x=getvalues()
```

```
    return x,x**2
```

```
#main program
```

```
n,result=square()
```

```
print("square({})={}".format(n,result))
```

OUTPUT:

Enter a number:6

square(6.0)=36.0

EXAMPLE-2:

#decorex1.py-----WAY ONE

```
def getvalues(): #normal function
```

```

return(float(input("Enter a number :")))

def square(getv):
    def cal():#inner function name
        val=getv()
        x=val**2
        return x
    return cal
#main program
squ=square(getvalues)
s=squ()
print("Square=",s)
OUTPUT:
Enter a number :5
Square= 25.0


---


#decorex2.py-----WAY2
def square(getv): #here square is called decorator
    def calculation(): # inner function name
        val=getv()
        s=val**2
        return s
    return calculation

def squareroot(getv): #here squareroot is called decorator
    def calculation(): # inner function name
        val=getv()
        s=val**0.5
        return s
    return calculation

@square # Programmer defined decorator
def getvalue(): # Normal function
    return(float(input("Enter a number for square:")))

@squareroot
def getnumber(): #Normal function
    return(float(input("Enter a number for squareroot:")))

#main program
result=getvalue()
print("Square:",result)
print("-"*50)
result=getnumber()
print("Square:",result)

```

OUTPUT:

Enter a number for square:12

Square: 144.0

Enter a number for squareroot:400

Square: 20.0

Modules in Python

=>We know that Functions concept makes us understand How to perform operations and we can re-use within the same program but not able to re-use the functions across the programs.

=>To reuse the functions across the programs, we must use the concept of Modules.

=>**Definition of Modules:**

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

=>**Types of Modules:**

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

- 1) Pre-defined (or) Built-in Modules
- 2) Programmer or user or custom-defined modules.

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

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

Examples: math cmath functools sys calendar os
re threading pickle random.....etc

=>Out of many pre-defined modules, in python programming one implicit pre-defined module imported to every python program called "builtins" .

2) **Programmer or user or custom-defined modules:**

=>These modules are developed by Python Programmers and they are available in Python Project and they are used by other python programmers who are in project development to deal with common requirements.

=>Examples:- aop mathsinfoetc

Development of Programmer-Defined Module

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

Step-1 : Define Variables (Global variables)

Step-2: Defined Functions

Step-3: Define Classes

=>After developing step-1, step-2 and step-3 , we must save on some file name with an extension .py (FileName.py) and it is treated as module name.

=>When a file name treated as a module name , internally Python execution environment creates a folder automatically on the name of __pycache__ and it contains module name on the name filename.cpython-310.pyc.

Examples:

 |
 |
 |**__pycache__**
 |-----
 |
 |**aop.cpathon-310.pyc** <-----Module Name
 |**mathsinfo.cpython-310.pyc**<-----Module Name
 |-----
 |
 |
 |

PROGRAM:

```
#This program cal sum sub and mul of two numbers
#aop.py--File Name-- and treated as module name
def addop(a,b):
    c=a+b
    print("sum({},{})={}".format(a,b,c))
def subop(a,b):
    c=a-b
    print("sub({},{})={}".format(a,b,c))
def mulop(a,b):
    c=a*b
    print("mul({},{})={}".format(a,b,c))
```

#This program accept three numbers and file biggest amont them by using modules
#big.py--file name and treated as module name

```
def big():
    a=int(input("Enter First value:"))
    b=int(input("Enter Second value:"))
    c=int(input("Enter Third value:"))
    if(a==b) and (b==c):
        print("ALL VALUES ARE EQUAL:")
    else:
        if(a>b) and (a>c):
            print("Biggest=",a)
        elif(b>c) and (b>a):
            print("Biggest=",b)
        else:
            print("Biggest=",c)
```

```
# Program for displaying the calendar for year and month
#calendarex1.py
import calendar as c
print(c.month(2022,4))
```

Program for displaying the calendar for year

```
#calendarex2.py
import calendar
print(calendar.calendar(2022))


---


#mathsinfo.py--file name--treated as module name
pi=3.14
e=2.71
k=272 # here pi,e, and x are called global variables
```

```
#SE1.py--File Name--some other programmer
import aop
aop.mulop(10,20) # Function call
aop.addop(10,20) # Function call
aop.subop(23,45) # Function call
```

```
#SE2.py--file name
from mathsinfo import *
print("val of pi=",pi)
print("val of e=",e)
print("val of k=",k)


---


#stud1.py
import big
big.big() # function call
```

Number of approaches to re-use Modules

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

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

1) By using import statement:

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

=>Syntax-1: **import module name**

=>This syntax imports single module

Example: **import big**
 import aop
 import mathsinfo

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

Example: import big,aop,mathsinfo

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

=>This syntax imports single module and aliased with another name

Example: import big as b
import aop as a
import mathsinfo as m

=>Syntax-4:

import module name1 as alias name, module name2 as alias name.....module

=>This syntax imports multiple modules and aliased with another names

Example: import big as b, aop as a , mathsinfo as m

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

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

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

=>Here "from" "import" are the key words

=>The purpose of from.... import statement is that " To refer or access the variable names, function names and class names in current program"

=> we can from.... import statement in 3 ways.

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

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

Example: from calendar import month
from aop import addop,subop,mulop
from mathinfo import pi,e

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

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

Example: from calendar import month as m
from aop import addop as a,subop as s, mulop as m
from mathinfo import pi as p ,e as k

Syntax-3: from module name import *

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

=>This syntax is not recommended to use because it imports required Features of Module and also import un-interested features also imported and leads more main memory space.

Example:

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

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

Variable Name
Function Name
Class Name

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

```
# Program for displaying the calendar for year and month
#calendarex1.py
import calendar as c,aop as a
print(c.month(2022,4))
print(c.month(2022,5))
print(c.month(2022,6))
```

a.addop(100,200)

```
# Program for displaying the calendar for year
#calendarex2.py
import calendar
print(calendar.calendar(2022))
```

```
# Program for displaying the calendar for year and month
#calendarex3.py
from calendar import month as m
from aop import *
from mathsinfo import pi as p
print(m(2022,4))
print(m(2022,5))
print(m(2022,6))
addop(10,15)
subop(100,200)
mulop(20,30)
```

```
print("val of pi=",p)
```

```
#city1.py--file name and treated as module name
def hello(s):
    print("Hi {}, Good Morning:".format(s))
#city2.py--filr name--acts as module name
def hello(s):
    print("Hello {}, Good Evening".format(s))
#citydemo.py--main program
from city2 import hello as e
from city1 import hello as m
m("Rossum")
e("Ritche")
```

```
#write python program which will generate multiplication table for a given number
by using modules concept
#multable.py----file name and module name
def table():
    n=int(input("Enter a number:"))
    if(n<=0):
        print("{} is invalid input".format(n))
    else:
        print("="*50)
        print("Mul Table for {}".format(n))
        print("="*50)
        for i in range(1,11):
            print("\t{} x {}={}".format(n,i,n*i))
        else:
            print("="*50)
#multabledemo.py--main program
from multable import table
table()
```

reloading a modules in Python

=>To reload 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:

=>`reload()` 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()`.

```
#shares.py---file and treated as module name
def sharesinfo():
    d={"Tech":19,"Pharma":11,"Auto":1,"Finance":100}
    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{}".format(sn,sv))
    else:
        print("-"*50)
#main program
d=shares.sharesinfo() #previously imported module
disp(d)
time.sleep(15)
importlib.reload(shares) # reloading previously imported module
d=shares.sharesinfo() # obtaining changed /new values of previously import
module
disp(d)
```

EXAMPLE:

```
#shares.py---file name and acts as module name
def sharesinfo():
    d={"IT":4,"Pharmacy":2,"Auto":3,"Mrkt":2}
    return d

#sharesdemo.py
import shares,time,importlib
def disp(d):
    print("=*50)
    print("\tShare Name\tShare Value:")
    print("=*50)
    for sn,sv in d.items():
        print("\t{}\t{}".format(sn,sv))
```

```
print("=*50)
#main program
d=shares.sharesinfo()
disp(d)
print("i am going to sleep")
time.sleep(20)
print("i am coming out of sleep:")
importlib.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 / environments.

=>The Package Concept is a collection of Modules.

=>The purpose of Packages is that to provide code re-usability across the folders / drives / environments.

=>To deal with the package, we need to 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 it is

considered as **Package Name**

Example:

bank <----Package Name

`__init__.py` <---Empty Python File
`simpleint.py` <--- Module Name

b) re-use the package

=>To re-use the modules of the packages across the folders / drives / environments, we have 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 Environmental Variables:

=>PYTHONPATH is one of the Environmental Variable

Steps for setting PYTHONPATH=E:\\KVR-PYTHON-11AM\\PACKAGES\\BANK

Exception Handling in Python—Most imp

Index:

=>Purpose of Exception handling
=>What is meant by Exception
=>What is meant by Exception Handling
=>Types of Errors

- a) Compile Time Errors
- b) Logical Errors
- c) Runtime Error

=>Type of Exceptions

- a) Pre-defined (or) built-in exceptions
- b) Programmer(or) User (or) Custom Defined exception

=>keywords used in exception handling

- 1) try
- 2) except
- 3) else
- 4) finally
- 5) raise

=>Syntax for Handling Exceptions
=>Programming Examples
=>Development of Programmer-Defined Exceptions
=>Programming Examples
=>ATM Use Case with Exceptions.

Exception Handling in Python—Most imp

=>The purpose of Exception handling is that " To Develop or Build Robust(Strong) Applications".
=>In Real Time, To develop any project or application, we need to choose a language.
=>By using this language, we develop, compile and run / execute the programs.
During process, we get Various errors and they are of 3 types.

- 1) Compile Time Errors
- 2) Logic Errors
- 3) Runtime Errors.

1) Compile Time Errors:

=> Compile Time Errors are those, which are occurring during Compilation Process(.py-->.pyc)
=> Compile Time Errors occurs due to Syntaxes not followed and they are traced by Python Compiler.
=> Compile Time Errors solved by Python Programmers at project development level.

2) Logic Errors:

=>Logic Errors are those which are occurring during Execution Time / Run Time.
=>Logic Errors are those which are occurring due to Wrong representation of logic.
=>Logic Errors always gives Wrong results and they must be solved by Python Programmers during project development level.

3) Runtime Errors:

=> Runtime Errors are those which are occurring during Execution Time / Run Time.
=>Runtime Errors are those which are occurring due to INVALID or WRONG INPUT entered by Application User or End Users.
=>Runtime errors are addressed by Programmers with their Forecasting Knowledge by studying the problem statement

Points to be remembered in Exception Handling

- 1) When the application user enters Invalid or Wrong Input then we get Runtime Errors.
 - 2) Runtime Errors by default gives Technical Error Messages, which are understandable by Programmers but not by end-users. Industry always recommends to give User-Friendly Error Messages instead of Technical Error Messages.
 - 3) Definition of exception: Runtime Errors of a Program are called Exceptions. (Invalid Input-->Runtime Error-->exception). Hence Every Invalid Input gives Exception.
 - 4) Exceptions by default generates technical Error messages. Programmatically, Program can convert Technical Error Messages into User-Friendly Error Messages.
-

5) Definition of Exception Handling:

- =>The Process of Converting technical error messages into User-Friendly Error Messages is called Exception Handling.
- 6) When an exception occurs in python, Internally 3 steps takes automatically.
 - a) PVM stops the program execution abnormally.
 - b) PVM comes out of the Program flow

- c) PVM generates by default Technical Error Messages.
- 7) To do (a),(b) and (c) steps , PVM creates an object of an appropriate exception class
- 8) When an exception occurs then PVM creates an object of appropriate exception class and by default generates Technical Error Messages.
- 9) Hence every exception is one the object of appropriate exception class.

NOTE:

=>Invalid Input-- Gives---->Exception--Makes PVM--->cerate object of exception class--and generates->technical error messages.

Types of Exceptions in Python

=>In Python Programming, we have two types exceptions. They are

- 1) Pre-defined or built-in exceptions
- 2) Programmer or user or custom defined exceptions.

1) Pre-defined or built-in exceptions:

=>These exceptions are developed by Language Developers and they are available in Python APIs and they are always used for dealing with Universal Problems.

=>Some of the Universal Problems are

- 1) division by zero problems (ZeroDivisionError)
 - 2) Invalid Number formats (ValueError)
 - 3) Invalid number values passing (TypeError)
 - 4) Invalid Indices (IndexError)
 - 5) Invalid attributes used in program (AttributeError)
 - 6) Invalid module names importing (ModuleNotFoundError)
 -etc
-

2) Programmer or user or custom defined exceptions.

=>These exceptions developed by Python Programmers and they are available in Python Project and used by other python programmers and they are always used for dealing with Common Problems.

=>Some of the Common Problems are

- 1) Attempting to enter Invalid PIN.
 - 2) Attempting Invalid User name or Password
 - 3) Attempting to withdraw more amount than available balance.
 - 4) Attempting to represent Invalid Pattern...etc
-

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 process, In Python Programming, we have 5 keywords. They are

- 1) try
- 2) except
- 3) else
- 4) finally

5) raise

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 process, In Python Programming, we have 5 keywords. They are

- 1) try
 - 2) except
 - 3) else
 - 4) finally
 - 5) raise
-

Syntax for Handling the exceptions:

```
try:  
    Block of Statements  
    generating exceptions  
except <exception class name-1>:  
    Block of statements generates  
    User-Friendly Error Messages.  
except <exception class name-2>:  
    Block of statements generates  
    User-Friendly Error Messages.  
  
-----  
-----  
except <exception class name-n>:  
    Block of statements generates  
    User-Friendly Error Messages.  
else:  
    Block of statements recommended  
    to generate Results  
finally:  
    Block of statements executed by  
    PVM Compulsorily.
```

```
#program for cal division of two numbers  
#Div1.py  
s1=input("Enter First Value:")  
s2=input("Enter Second Value:")  
a=int(s1)  
b=int(s2)  
c=a/b  
print("val of a=",a)  
print("val of b=",b)  
print("Div=",c)
```

```

#program for cal division of two numbers
#Div2.py
try:
    print("\nI am try Block")
    s1=input("\nEnter First Value:")
    s2=input("Enter Second Value:")
    a=int(s1) #-----X
    b=int(s2) #-----X
    c=a/b #-----X
except ZeroDivisionError:
    print("\nDON'T ENTER ZERO FOR DEN...")
except ValueError:
    print("\nDON'T ENTER strs / symbols / alpha-numeric strs")
else:
    print("----else block---")
    print("val of a=",a)
    print("val of b=",b)
    print("Div=",c)
finally:
    print("\ni am finally block")

```

Explanation for the keywords of exception handling

1. try :

=>It is the block in which we block of statements generating exceptions. In Other words, what are

all the statements generates exceptions , such statements must be written within try block and hence try block is called exception monitoring block.

=>When an exception occurs in try block then PVM comes out of try block and executes

appropriate except block and generates User-Friendly error messages.

=>After executing appropriate except block, PVM Never goes to try block to execute rest of the

statements in try block

=>Every try block must be immediately followed by except block.

=>Every try block must contain atleast one except block and recommended to write multiple

except blocks for generating multiple user-frinedly error messages.

2. except block:

=>This is the block, In which we write block of statements for generating User-Friendly Error

Messages. In otherwords except block supresses Technical error messages and generates User-Friendly Error Messages and hence except is block is called Exception Processing Block.

Note:- Handling Exception=try block + except block

=>except block will execute when an exception occurs in try block.

=>Even we write multiple except blocks, PVM will execute only one except block depends type of exception occurs in try block.

=>except block must be written after try block and before else block (if we write else block)

3.else block:

=>This is the block, in which we write block of statements recommended to generate Result of program.

=>else block will execute when there is no exception occurs in try block.

=>Writing else block is optional.

=>else block must be written after except block and before finally block(if we write).

4. finally block:

=>It is the block in which we write block of statements and they are relinquishing (Close / release /

give-up / clean-up) the resources (files, database) which are obtained in try block

=>finally block will execute compulsorily (if we write)

=>Writing finally block is optional

=>finally block must be written after else block.

5.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)
```

Various forms of except blocks

=>The except block can be in various forms. They are

Form-1 : This form handles one exception at a time

Syntax:- try:

except <exception-class-name-1>

except <exception-class-name-2>

except <exception-class-name-n>

EXAMPLE:

```
#div2.py
try:
    s1=input("Enter first value:")
    s2=input("Enter second value:")
    a=int(s1)-----x
    b=int(s2)-----x
    c=a/b-----x
except ZeroDivisionError:
    print("Dont enter zero for denominator.....")
except ValueError:
    print("Dont enter str/symbol/alphanumeric value.....")
```

Form-2 : This form handles multiple specific exceptions at a time and this feature is called multi exception handling block

Syntax:- try:

except (exception class name-1, exception class name-2,.. exception
class name-n):

EXAMPLE:

```
#div3.py
try:
    s1=input("Enter first value:")
    s2=input("Enter second value:")
    a=int(s1)-----x
    b=int(s2)-----x
```

```
c=a/b#-----x
except(ZeroDivisionError,ValueError):
    print("\nDont enter zero for denominator.....")
    print("Dont enter str/symbol/alphanumeric value.....")
else:
    print("Value of dividend:",a)
    print("Value of divisor:",b)
    print("{} / {} = {}".format(a,b,c))
finally:
    print("\n\nI am in finally block.....")else:
    print("Value of dividend:",a)
    print("Value of divisor:",b)
    print("{} / {} = {}".format(a,b,c))
```

Form-3: This form displays Tech Error message(Problem Name) generated due to exception
occurrence

Syntax:- try:

```
-----  
-----  
except <exception-class-name-1> as alias name:  
-----  
-----
```

```
except <exception-class-name-2> as alias name:  
-----  
-----
```

```
except <exception-class-name-n> as alias name:  
-----  
-----
```

```
#div4.py
try:
    s1=input("Enter first value:")
    s2=input("Enter second value:")
    a=int(s1)#-----x
    b=int(s2)#-----x
    c=a/b#-----x
except ZeroDivisionError as z:
    print(z)
except ValueError as v:
    print(v)
else:
    print("Value of dividend:",a)
    print("Value of divisor:",b)
    print("{} / {} = {}".format(a,b,c))
```

Form-4: This form deals with default except block.

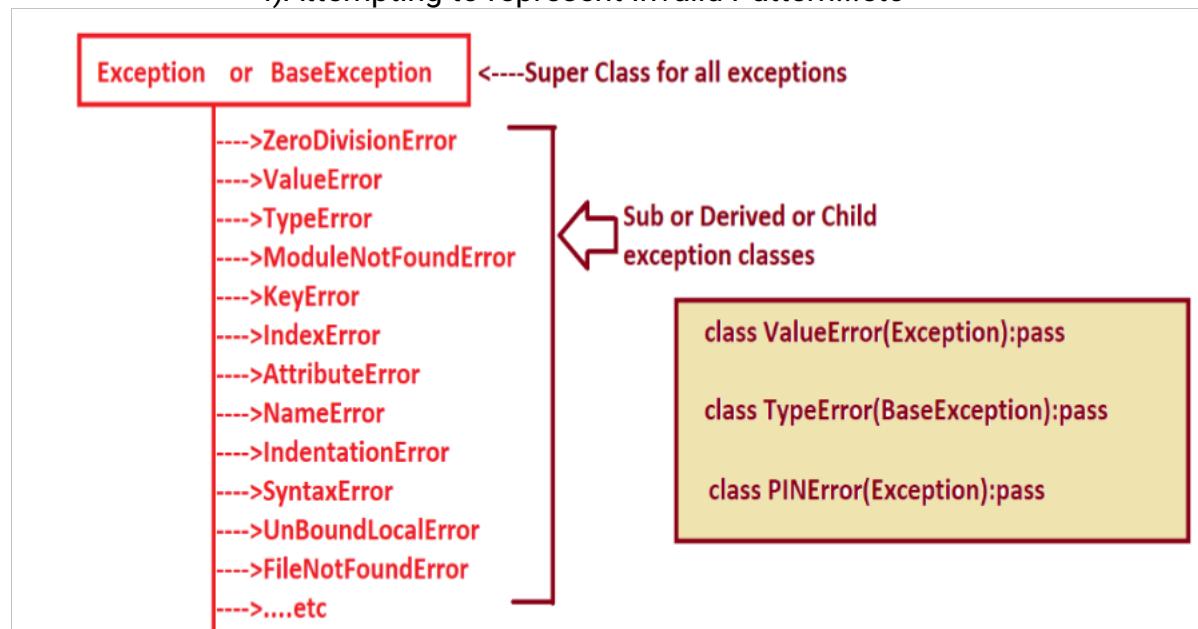
```
#div5.py
try:
    s1=input("Enter first value:")
    s2=input("Enter second value:")
    a=int(s1)-----x
    b=int(s2)-----x
    c=a/b-----x
except:
    print("Ooop's something went wrong.....!!!")
else:
    print("Value of dividend:",a)
    print("Value of divisor:",b)
    print("{} / {} = {}".format(a,b,c))
```

Development of Programmer-Defined Exceptions

=>These exceptions developed by Python Programmers and they are available in Python Project and used by other python programmers and they are always used for dealing with Common Problems.

=>Some of the Common Problems are

- 1) Attempting to enter Invalid PIN.
- 2) Attempting Invalid User name or Password
- 3) Attempting to withdraw more amount than available balance.
- 4) Attempting to represent Invalid Pattern...etc



=>To Develop programmer-defined exceptions, we must follow the following steps

Step-1: Choose the Programmer_defined Class Name

Step-2: The Programmer-defined class must inherit from either Exception or BaseException (Here Exception or BaseExcetion are called base classes in

exception handling)

Examples: class KvrDivisionError(Exception):pass
 class PinError(BaseException):pass

Hence, To develop any programmer-defined exceptions based applications, we must ensure that there exist 3 phases. They are

- 1) Development of Programmer-defined Exception sub class
 - 2) Development a Function which will hit programmer-defined exception
 - 3) development of main program for handling the exception.
-

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)
```

EXCEPTIONS EXAMPLES:

```
#This program defined Programmer defined exception and works like
ZeroDivisionError
#kvr.py--file name and acts as Module Name
#      (1)          (2)
class KvrDivisionError(Exception):pass
#Phase-1: Development of Programmer-defined Exception sub class
```

```
#This Program accept two values from main program and computes their division
#divop.py-----file name and acts as module name
from kvr import KvrDivisionError
def division(a,b):
    if(b==0):
        raise KvrDivisionError # Hitting or generating or raising an exception
when cond is satisfied.
    else:
        return (a/b)
```

#Phase-2 : Development of Function which will hit programmer-defined exception with raise keyword

```
#This program accepts two from KBD (end user) and gets the result
#divdemo.py
from divop import division
from kvr import KvrDivisionError as KS
try:
    a=int(input("Enter Value of a:"))
    b=int(input("Enter Value of b:"))
    result=division(a,b) # Function call
except KS:
    print("\nDon't enter zero for den...")
except ValueError:
    print("\nDon't enter strs / alpha-numerics/symbols")
except :
    print("Some thing went wrong!")
else:
    print("Div=",result)
finally:
    print("I am from finally block")
```

#Phase-3: development of main program for handling the exception (try and except blocks)

```
#This program accepts two from KBD (end user) and gets the result
#divdemo1.py
from divop import division
from kvr import KvrDivisionError
try:
    a=int(input("Enter Value of a:"))
    b=int(input("Enter Value of b:"))
    result=division(a,b) # Function call
except (KvrDivisionError , ValueError):
    print("\nDon't enter zero for den...")
    print("\nDon't enter strs / alpha-numerics/symbols")
except :
    print("Some thing went wrong!")
else:
    print("Div=",result)
finally:
    print("I am from finally block")
```

```

#Sample.py
class KvrDivisionError(Exception):pass # Development of Programmer-defined
Exception sub class

def division(a,b): # Development a Function which will hit programmer-defined
exception
    if(b==0):
        raise KvrDivisionError # Hitting or generating or raising an exception
when cond is satisfied.
    else:
        return (a/b)
#main program---- development of main program for handling the exception.
try:
    a=int(input("Enter Value of a:"))
    b=int(input("Enter Value of b:"))
    result=division(a,b) # Function call
except KvrDivisionError:
    print("\nDon't enter zero for den...")
except ValueError:
    print("\nDon't enter strs / alpha-numerics/symbols")
else:
    print("Div=",result)
finally:
    print("I am from finally block")

```

EXAMPLE-2

GENARATE TABLE OF GIVEN NUMBER.....

```

#Phase-1: development of programmer-defined exceptions
#excepts.py---file name and acts as module name
class NegativeError(Exception):pass
class ZeroError(BaseException):pass


---


Phase-2: development of FUNCTION fe displaying result and hitting exceptions
#multable.py---file name and acts as module name
from excepts import NegativeError,ZeroError
def table(n):
    if(n==0):
        raise ZeroError
    elif(n<0):
        raise NegativeError
    else:
        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)
```

#Phase-3: development of main program which will handle the exceptions

#multabledemo.py

from multable import table

from excepts import NegativeError,ZeroError

try:

n=int(input("Enter a number:"))

table(n)

except ValueError:

print("\nDon't enter strs / alpha-numerics/symbols")

except NegativeError:

print("\nDon't enter -ve numbers")

except ZeroError:

print("\nDon't enter Zero :")

FILE

Files in Python (OR) Stream Handling in Python

Index:

- =>Purpose of Files
 - =>Types of Applications
 - a) Non-Persistent Applications
 - b) Persistent Applications
 - =>Definition of Files
 - =>Operations on Files
 - a) Write Operations
 - b) Read Operations
 - =>Types of Files
 - a) Text Files
 - b) Binary Files
 - =>Syntax for opening the Files
 - a) By Using Open()
 - b) By using " with open() "
 - =>File Opening Modes
 - 1) r
 - 2) w
 - 3) a
 - 4) r+
 - 5) w+
 - 6) a+
 - 7) x
 - =>Function for writing the data to the Files
 - a) write()
 - b) writeline()
 - =>Function for Reading the data to the Files
 - a) read()
 - b) read(no.of chars)
 - c) readline()
 - d) readlines()
 - =>Random Access Files in Python
 - a) tell()
 - b) seek()
 - =>Programming Examples
 - =>Pickling and Un-pickling Concept
 - (Object Serialization and De-serialization)
 - =>"pickle" module
 - =>Programming Examples
 - =>"os" module
 - =>Programming Examples
-

Files in Python (OR) Stream Handling in Python

=>The purpose of Files concept is that " To store the data Permanently ".
=>The process of storing the data permanently is known as "Persistency".

=>In The context of files, we can develop two types of applications. They are
a) Non-persistency Applications
b) Persistent Applications.

=>In Non-persistency Application development, we accept the data from Key board, stores in main memory in the form of objects, process the data and shown the result on the console .

=>The results which are available in the main memory are Temporary.

=>In persistent Application development, we accept the data from Key board, stores in main memory in the form of objects, process the data and whose results are stored permanently.

=>To store the result of any programming language permanently, we have two approaches. they are

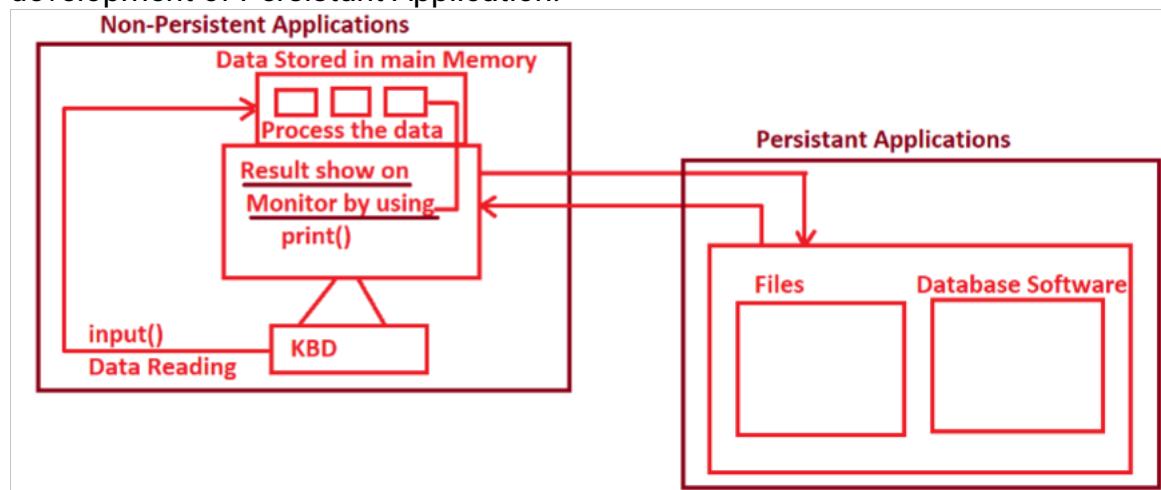
- a) By using Files
- b) By using data base softwares.

=>Note that if we write any python programming by using Files and Database softwares then such type of applications are comes under the example Persistent Applications.

Persistent Application development by using Files in Python

=>File Concept is one language Independent where we can store the data permanently.

=>The communication between Python Program and Files is comes under development of Persistant Application.



=>Definition of Files:

=> A File is a collection of Records
=>A File Name is one of the Named Location in Secondary Memory.
=>Files of any programming resides in Secondary Memory.
=>In Files Programming, Every Object Data becomes a record in a file of secondary memory and every record of file of secondary memory will become an object in main memory.

Definition of Stream:

=>The flow of data between main memory and file of secondary memory is called Stream.

Operations on Files

=>On the files, we can perform Two Types of Operations. They are

- 1) Write Operation.
- 2) Read Operation.

1) Write Operation:

=>The purpose of write operation is that " To transfer or save the object data of main memory as record in the file of secondary memory".

=>Steps:

- 1) Choose the File Name
- 2) Open the File Name in Write Mode
- 3) Perform cycle of Write Operations.

=>While we performing write operations, we get the following exceptions.

- a) IOError
- b) OSError
- c) FileNotFoundError

2) Read Operation:

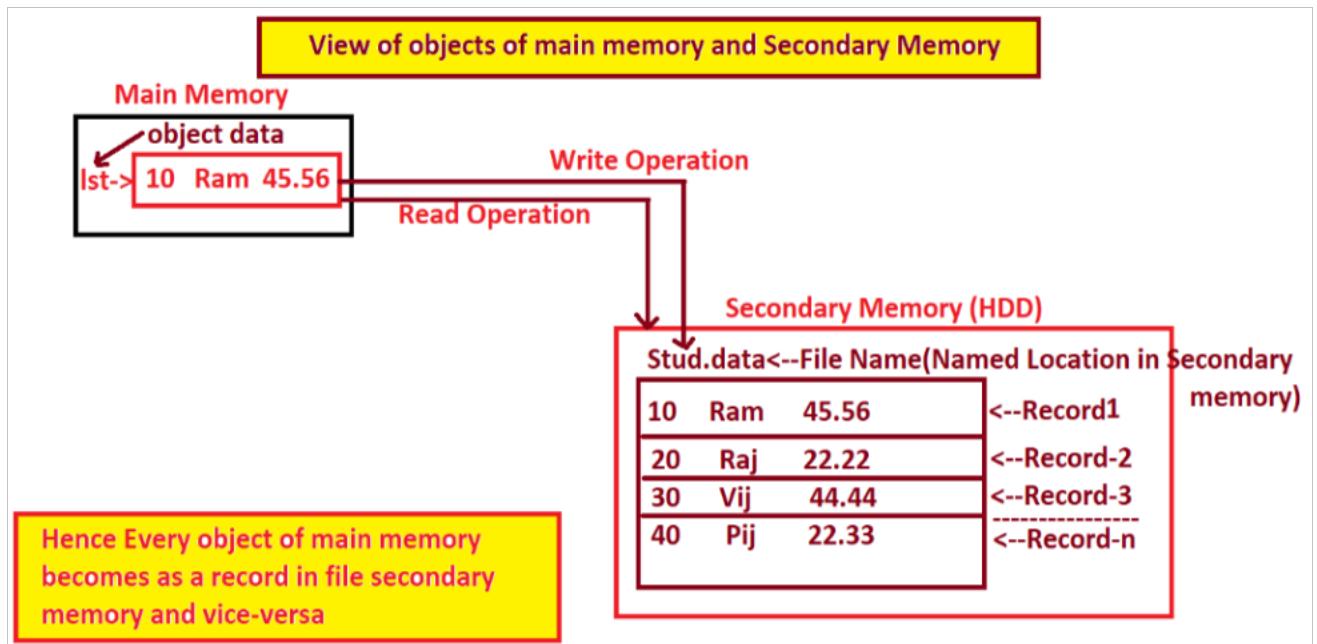
=>The purpose of read operation is that " To transfer or read the record from file of secondary memory into the object of main memory".

=>Steps

- a) Choose the file name
- b) Open the file name in Read Mode
- c) Perform cycle of read operations.

=>While we performing read operations, we get the following exceptions.

- a) FileNotFoundError
 - b) EOFError
-



File Opening Modes in Python

=>To open the file, we must specify type of file mode and they are classified into 7 types. They are

1) r mode

=>This mode is used for opening the file in READ Mode.
=>This is one of default mode.
=>If we open any file name in "r" mode and if that file does not exists in secondary memory we get FileNotFoundError.

2) w mode

=>This mode is used for creating a new file and opens in write mode always.
=>If the file name already exists and if we open it in "w" mode then it opens in write mode and existing data overlaps with new data.

3) a mode

=>This mode is used for opening new file and performs write operation.
=>This mode can also be used for opening existing file in write mode and new data will be appended to the existing data.

4) r+ mode

=>This mode is used for opening the file in read mode.
=>When we open the file name in "r+" mode then on that file(if file exists) we can perform First read operation and later we perform Write Operation.

5) w+ mode

=>This mode is used opening the file always in write mode , First performs Write

operation and later we can perform Read Operation.

=>Note that w+ mode Functionality is similar to "w" mode but with this mode additionally we can perform read operation.

6) a+ mode

=>=>This mode is used for opening new file and performs write operation.

=>This mode can also be used for opening existing file in write mode and new data will be appended to the existing data.

=>Additionally with this mode we can perform Read Operation.(First we perform Write Operation and later we perform read operation)

7) x mode

=>This mode is exclusively used for opening or creating the file in write mode.

=>This mode opens any file in write mode only once.

=>If we open exiting file in "x" mode then we get FileNotFoundError

#This program opens "kvr.data" in read mode

#FileOpenEx1.py

try:

 fp=open("kvr.data") # Here fp is an object of TextIOWrapper
except FileNotFoundError:

 print("File does not exists")

else:

 print("\nFile Opened in read mode successfully")

 print("Type of fp=",type(fp))

 print("-"*40)

 print("File Mode=",fp.mode)

 print("is readable?=",fp.readable())

 print("is writeable?=",fp.writable())

 print("Is file Closed=",fp.closed)

 print("-"*40)

finally:

 print("i am from finally block")

 fp.close() # manual Closing of files

 print("Is file Closed in finally block=",fp.closed)

#This program opens "kvr.data" in write mode

#FileOpenEx2.py

kv=open("kvr.data","w")

print("\n File Created Successfully in write mode:")

print("Type of kv=",type(kv))

Types of Files

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

a) Text Files

b) Binary Files

1) Text File:

=>A Text File always contains Alphabets, Digits and Special Symbols.

=>Text Files always denoted by a letter "t"

=>The default file in python is text file.

Examples: .py .java .c .cpp .txt .doc....etc

2) Binary File:

=>A BinaryFile always contains Data in Binary Format (Pixles)

=>Binary Files always denoted by a letter "b"

Examples: images (.jpg, .jpeg, .png, .gif)
audio and video files
PDF documents

opening the Files

=>To perform various operations on files, First we must open the file. In Python, we have 2 syntaxes to open the file. They are

- 1) by using "open()
 - 2) by using "with open()"
-

1) by using "open()":

=>open() is one of the pre-defined function and it present in default module builtins

=>This function is used for opening any specified file in specified file mode.

=>Syntax:- varname=open("FileName","File Mode")

Explanation:

=>varname is an object of TextIOWrapper and it is pointing to the file and it is called File pointer.

=>FileName represents Type of file

=>File Mode can be either of r, w,a, r+,w+,a+,x

=>Once we open the file with open() , then it is mandatory to close file by using close() and it is one of the manual process. (Not supporting Auto Closeable Property)

EXAMPLES:

```
#This program opens "kvr.data" in read mode
#FileOpenEx1.py
try:
    fp=open("kvr.data") # Here fp is an object of TextIOWrapper
except FileNotFoundError:
    print("File does not exists")
else:
    print("\nFile Opened in read mode successfully")
    print("Type of fp=",type(fp))
```

```
print("-*40)
print("File Mode=",fp.mode)
print("is readable?=",fp.readable())
print("is writeable?=",fp.writable())
print("Is file Closed=",fp.closed)
print("-*40)
finally:
    print("i am from finally block")
    fp.close() # manual Closing of files
    print("Is file Closed in finally block=",fp.closed)
```

```
#This program opens "kvr.data" in write mode
#FileOpenEx2.py
kv=open("kvr.data","w")
print("\n File Created Successfully in write mode:")
print("Type of kv=",type(kv))
```

2) by using " with open()

Syntax: with open("File Name", "File Mode") as <variable name>:

Block of statement--File Operations

Explanation:

- =>here "with" and "as" are the keywords
- =>open() is pre-defined Function used for opening any specified file in specified file mode.
- =>varname is an object of TextIOWrapper and it is pointing to the file and it is called File pointer.
- =>Block of statements represents set of executable statements meant for performing file operations.
- =>The advantage of " with open() as " approach is that " As long as PVM present inside of "with open() as " indentation , File is actively Under opening state and Once PVM comes out of "with open() as " Indentation then file will be closed automatically.
- =>This approach provides auto-closeable of files. (No need to write finally block and no need to use close()).

EXAMPLES:

```
#This program demonstartes various file opening modes with "with open() as " as
approach
#FileOpenEx3.py
try:
    with open("hyd.info","r+") as fp:
        print("-*50)
        print("Name of file=",fp.name)
        print("File Mode=",fp.mode)
        print("is file readable=",fp.readable())
        print("is file writable=",fp.writable())
```

```
        print("Is file closed in with open()=",fp.closed)
        print("-"*50)
    except FileNotFoundError:
        print("File does not exists")
    else:
        print("Is file closed in with open() in else block=",fp.closed)
```

```
#This program demonstartes various file opening modes with "with open() as " as
approach
#FileOpenEx4.py
with open("hyd.info","a+") as fp:
    print("-"*50)
    print("Name of file=",fp.name)
    print("File Mode=",fp.mode)
    print("is file readable=",fp.readable())
    print("is file writable=",fp.writable())
    print("Is file closed in with open()=",fp.closed)
    print("-"*50)
print("\ni am out of with open() indentation:")
print("Is file closed in with open()=",fp.closed)
```

```
#This program demonstartes various file opening modes with "with open() as " as
approach
#FileOpenEx5.py
try:
    with open("hyd1.info","x") as fp:
        print("-"*50)
        print("Name of file=",fp.name)
        print("File Mode=",fp.mode)
        print("is file readable=",fp.readable())
        print("is file writable=",fp.writable())
        print("Is file closed in with open()=",fp.closed)
        print("-"*50)
except FileExistsError:
    print("File already exists--it can't create again")
else:
    print("\ni am out of with open() indentation--else block:")
    print("Is file closed in with open()=",fp.closed)
```

Writing the data to the file

=>To write the data to the file, we have two pre-defined functions. They are

- 1) write()
- 2) writelines()

1) write()

=>This function is used for writing any type of data to the file in the form of str .

=>Syntax:- filepointer.write(data)

=>filepointer is a valid variable name and it is always pointing to the file.

=>data represents any type of value and it must be in the form of str always.

Example:

#This program demonstartes how write address of various people by using write()

#FileWriteEx1.py

with open("addr1.data","a") as fp:

```
    fp.write("Dennis Ritche\n")
    fp.write("5-4,Mountain side\n")
    fp.write("Bell Labs\n")
    fp.write("USA\n")
    print("\nData Written to the file--plz verify:")
```

2) writelines()

=>This function is used for writing any type of Iterable object data in the form of str .

=>Syntax:- filepointer.writelines(data)

=>filepointer is a valid variable name and it is always pointing to the file.

=>data represents any type of Iterable object and it must be in the form of str always.

Example:

#This program demonstrates how write address of various people by using writelines()

#FileWriteEx2.py

d={"Name":"Rossum","HNO":"3-4-Hill Side","Cname":"PSF","country":"Nether Lands"}

with open("addr2.data","a") as fp:

```
    fp.writelines("\n"+str(d))
    print("\nData Written to the file--plz verify:")
```

#This program demonstrates how write the data dynamically to the file by reading from KBD.

#FileWriteEx3.py

fname=input("Enter the file name to write the data:")

with open(fname,"a") as fp:

```
    print("Enter the data and press '@' to stop:")
    while(True):
```

```
        kbddata=input()
```

```
        if(kbddata=="@"):
```

```
            break
```

```
        else:
```

```
            fp.write(kbddata+"\n")
```

```
    print("Data written to the file--verify")
```

Reading the data from the file

=>To read the data from files, 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 in the form of str

=>Syntax:- varname=filepointer.read()

=>Here varname is of type str and it contains the entire content of file.

b) read(no. of chars):

=>This Function is used for reading specified number characters from the file in the form of str

=>Syntax:- varname=filepointer.read(no. of chars)

=>Here varname is of type str and it contains the specified number characters

c) readline():

=>This Function is used for reading One Line at a time from the file in the form of str

=>Syntax:- varname=filepointer.readline()

=>Here varname is of type str and it contains the one Line of data from file.

d) readlines():

=>This Function is used for reading All Lines at a time from the file in the form of list

=>Syntax:- varname=filepointer.readlines()

=>Here varname is of type list and it contains the all Lines of file.

EXAMPLES:

```
#This program reads the content of any file and displays on the console.---read()
```

```
#FileReadEx1.py
```

```
fname=input("Enter the file Name:")
```

```
try:
```

```
    with open(fname,"r") as fp:  
        filedata=fp.read()  
        print("content of file:")  
        print("-----")  
        print(filedata)  
        print("-----")
```

```
except FileNotFoundError:
```

```
    print("File does not exists:")
```

```
#This program reads the specified number characters from file -----read(no. of chars)
```

```
#FileReadEx2.py
```

```
try:  
    with open("addr1.data") as fp:  
        print("Initial Position of fp=",fp.tell()) # 0  
        fdata=fp.read(5)  
        print("file data=",fdata)  
        print("Now Position of fp=",fp.tell()) # 5  
        fdata=fp.read(8)  
        print("file data=",fdata)  
        print("Now Position of fp=",fp.tell()) # 13  
        fdata=fp.read()  
        print("file data=",fdata)  
        print("Now Position of fp=",fp.tell()) # 183  
        fp.seek(0) # Here fp repositioned to starting index  
        print("-----")  
        fdata=fp.read()  
        print("file data=",fdata)  
        print("-----")  
except FileNotFoundError:  
    print("File does not exists:")
```

```
#This program reads the one line at a time from file -----readline()  
#FileReadEx3.py
```

```
try:  
    with open("addr1.data") as fp:  
        fdata=fp.readline()  
        print("File Data=",fdata)  
        fdata=fp.readline()  
        print("File Data=",fdata)  
        fdata=fp.readline()  
        print("File Data=",fdata)  
        print("Index of fp=",fp.tell())  
except FileNotFoundError:  
    print("File does not exists:")
```

```
#This program reads the all lines at a time from file -----readlines()  
#FileReadEx4.py
```

```
try:  
    with open("addr1.data") as fp:  
        lines=fp.readlines()  
        print("-----")  
        print("File Data:")  
        print("-----")  
        for line in lines:  
            print("{}\n".format(line),end="")  
        print("-----")  
except FileNotFoundError:  
    print("File does not exists:")
```

```
#This program copy the content of one file into another file
```

```
#FileCopy.py
```

```
sfile=input("Enter the source File:")
```

```
try:  
    with open(sfile,"r") as rp:  
        dfile=input("Enter Destination File:")  
        with open(dfile,"a") as wp:  
            sfdata=rp.read()  
            wp.write(sfdata)  
            print("\n{} File Data Copied into {}".format(sfile,dfile))  
except FileNotFoundError:  
    print("File does not exists")  


---



```
#This program accept any file name and find number of lines, words and characters .
#FileCountInfo.py
fname=input("Enter File Name:")
try:
 nol=0
 nw=0
 nc=0
 with open(fname,"r") as fp:
 Filelines=fp.readlines()
 for line in Filelines:
 print("{}").format(line),end=""")
 nol=nol+1
 nw=nw+len(line.split())
 nc=nc+len(line)
 else:
 print("\n-----")
 print("Number of lines=",nol)
 print("Number of words=",nw)
 print("Number of Characters=",nc)
 print("-----")
except FileNotFoundError:
 print("File does not exists")
```



---


```

Pickling and Un-Pickling in Python (OR) Object Serialization and Deserialization

Pickling(Object Serialization):

=>Let assume there exists an object with multiple values in main memory. To transfer or write multiple values of object into the file secondary memory we use write() or writelines(). These functions by default writes the object values in the form of value by value and it is one of the time consuming process.
=>To overcome this problem, we use the concept of Pickling.
=>The advantage of pickling concept is that To transfer or write entire object content of main memory into file secondary memory with single write operation.

=>Definition of Pickling:

=>The process of transferring or writing entire object content into the file secondary

memory by performing single Write Operation is called Pickling.

=>Hence Pickling concept participates in write Operation.

=>At the time of dealing with Pickling Operations, we must ensure that the files must be taken as

Binary format(File Type=Binary).

Implementation of Pickling Concept:

=>import pickle module

=>Create an object with collection values

=>Choose the file name and opened in write mode as binary file

=>Use pickle.dump(). The purpose of dump() is that to write or transfer entire object content into file of

secondary memory.

Syntax:- pickle.dump(object, filepointer)

Un-Pickling(Object De-Serialization):

=>Let us assume there exists a record (s) in a file of secondary memory. To read or transfer the entire record content from the file of secondary memory into the object of main memory by using read(), read(no.of chars), readline() and readlines() then these takes reads the record values in the form of value by value and takes more execution time and it is one of the time consuming process.

=>To Overcome this problem, we must use the concept called Un-Pickling.

=>The advantage of un-pickling concept is that To read entire record content at a time from file of secondary memory into object of main memory by performing single read operation.

=>Definition of Un-Pickling:

=>The process reading or transferring the entire record content at a time from file of secondary memory

into object of main memory by performing single read operation is called Un-Pickling.

=>Hence Un-Pickling concept participates in Read Operation.

=>At the time of dealing with Un-Pickling Operations, we must ensure that the files must be taken as

Binary format(File Type=Binary).

=>Implementation of Un-Pickling:

=>Import pickle module

=>Choose File Name and open it into Read Mode as Binary Format

=>use load() of pickle module. The purpose of load() is that to transfer entire record content from file of secondary memory into object of main memory.

Syntax: pickle.load(file pointer)

```
#Program for accepting no of students details such as stdno, stdname, marks and
college name and save them in a file (use pickling)
#studpickex1.py
import pickle
with open("pythstud.data", "ab") as fp:
    nos=int(input("Enter how many students u have:"))
    if(nos<=0):
        print("Invalid input:")
    else:
        for i in range(1,nos+1):
            print("-"*50)
            print("Enter {} Student Information:".format(i))
            print("-"*50)
            #accept student values
            stno=int(input("Enter Student Number:"))
            sname=input("Enter Student Name:")
            marks=float(input("Enter Student Marks:"))
            cname=input("Enter Student College Name:")
            #create an object
            lst=list() # create an empty list
            #append student data
            lst.append(stno)
            lst.append(sname)
            lst.append(marks)
            lst.append(cname)
            #write object data to the file
            pickle.dump(lst,fp)
            print("-"*50)
            print("{} Student data saved successfully in File".format(i))
```

```
#Program reading the record(s) from the file (use un-pickling)
#studunpick.py
import pickle
try:
    with open("pythstud.data", "rb") as fp:
        print("-"*50)
        print("Stno\tName\tMarks\tCname")
        print("-"*50)
        while(True):
            try:
                record=pickle.load(fp)
                for val in record:
                    print("{}\t".format(val),end="")
                print()
            except EOFError:
                print("-"*50)
                break
except FileNotFoundError:
```

```
print("File does not exists:")
```

```
#Program for accepting no of students details such as stdno, stdname, marks and
college name and save them in a file (use pickling)
#studpickex2.py
import pickle
def pickleexample():
    with open("pythstud.data", "ab") as fp:
        nos=int(input("Enter how many students u have:"))
        if(nos<=0):
            print("Invalid input:")
        else:
            for i in range(1,nos+1):
                print("-"*50)
                print("Enter {} Student Information:".format(i))
                print("-"*50)
                #accept student values
                stno=int(input("Enter Student Number:"))
                sname=input("Enter Student Name:")
                marks=float(input("Enter Student Marks:"))
                cname=input("Enter Student College Name:")
                #create an object
                lst=list() # create an empty list
                #append student data
                lst.append(stno)
                lst.append(sname)
                lst.append(marks)
                lst.append(cname)
                #write object data to the file
                pickle.dump(lst,fp)
                print("-"*50)
                print("{} Student data saved successfully in
File".format(i))
```

```
#main program
pickleexample()
```

```
#Program reading the record(s) from the file (use un-pickling)
#studunpickex2.py
import pickle
def unpickleexample():
    try:
        with open("pythstud.data", "rb") as fp:
            print("-"*50)
            print("Stno\tName\tMarks\tCname")
            print("-"*50)
            while(True):
                try:
                    record=pickle.load(fp)
                    for val in record:
```

```
        print("{}\t".format(val),end="")
    print()
except EOFError:
    print("-"*50)
    break
except FileNotFoundError:
    print("File does not exists:")

```

```
#main program
unpickleexample()
```

#write a python program which will accept employee details such as employee number , employee name, salary and company name and store employee details in a file by using pickling concept

```
#EmpPick.py
import pickle,sys
def stroreempdata():
    with open("emp.data","ab") as fp:
        while(True):
            try:
                #accept employee details
                print("-"*50)
                eno=int(input("Enter Employee Number:"))
                ename=input("Enter Employee Name:")
                sal=float(input("Enter Employee Salary:"))
                cname=input("Enter Employee Company:")
                #create an empty list
                lst=[]
                #append the employee data
                lst.append(eno)
                lst.append(ename)
                lst.append(sal)
                lst.append(cname)
                #write or save entire object data to File---dump()

```

of pickle module

```
                pickle.dump(lst,fp)
                print("-"*50)
                print("Employee Record Saved in a file:")
                print("-"*50)
                ch=input("Do u want to insert another Employee
Record(yes/no):")
                if(ch.lower() == "no"):
                    sys.exit()
            except ValueError:
                print("Don't enter strs/sysmbols/alpha-numerics
for number and sal:")

```

```
#main program
stroreempdata()
```

```
#Program for reading all the records from file by using un-pickling
#empunpick.py
import pickle
try:
    with open("emp.data","rb") as fp:
        print("-"*50)
        print("Employee Details")
        print("-"*50)
        while(True):
            try:
                emprecord=pickle.load(fp) # emprecord var of
<class, 'list'>
                for val in emprecord:
                    print("{}\t".format(val),end="")
                print()
            except EOFError:
                print("-"*50)
                break
except FileNotFoundError:
    print("File does not exists")
```

os module

=>"os" is one of pre-defined module

=>The purpose of 'os' module is that " To perform certain Os Level Operations"

=>The essential Os Level Operations are :

- 1) getting current working folder (**getcwd()**)
 - 2) create a folder (**mkdir()**)
 - 3) create a folders hierarchy (**makedirs()**)
 - 4) remove a folder (**rmdir()**)
 - 5) remove a folders Hierarchy (**removedirs()**)
 - 6) rename a folder (**rename()**)
 - 7) list files in folder (**listdir()**)
-

1) getting current working folder:

=>To get current working folder, we use a pre-defined function `getcwd()` of os module

=>Syntax: varname=os.getcwd()

```
# cwdex1.py
import os
cwdname=os.getcwd()
print(cwdname)
```

2) create a folder:

=>To create a folder, we use `mkdir()` of os module.

=>Syntax:- `os.mkdir("folder name")`

=>`mkdir()` can create one folder at a time and it is unable create Folders Hierarchy.

=>`mkdir()` generate `FileExistsError` provided if the folder already exists.

=>`mkdir()` generates `FileNotFoundException` provided when we attempt create Folders Hierarchy.

Examples:

```
# mkdirex.py
import os
try:
    os.mkdir("D:\\kiwi\\mango\\guava")
    print("Folder created successfully--verify")
except FileExistsError:
    print("Folder already exists")
except FileNotFoundError:
    print("Creation of Folders Hierarchy is not possible:")
```

3) create a folders hierarchy:

=>To create a folders hierarchy, we use `makedirs()` of os module

=>Syntax:- `os.makedirs("folders hierarchy")`

=>If the folders Hierarchy already exists then we get FileExistsError.

Examples:

```
#Program for creating a folders Hierarchy  
#mkdирsex.py  
import os  
try:  
    os.makedirs("D:\\India\\Hyd\\Ameerpet\\Python\\Data Science")  
    print("Folders created successfully--verify")  
except FileExistsError:  
    print("Folder already exists")
```

4) remove a folder:

=>To remove a folder, we use rmdir() of os module

=>Syntax: os.rmdir("folder name")

=>If folder name does not exists then we get FileNotFoundError.

=>If the folder contains any files or sub folder then it is not possible to remove folder and we get OSError.

=>rmdir() can remove one folder at a time but possibel to remove Folders Hierarchy.

Examples:

```
#rmdirsex.py  
import os  
try:  
    os.rmdir("D:\\hyd")  
    print("Folder removed successfully--verify")  
except FileNotFoundError:  
    print("Specified Folders does not exists")  
except OSError:  
    print("Specified Folder is not empty--not possible to remove")
```

5) remove a folders Hierarchy:

=>To remove a folders Hierarchy, we use removedirs() of os module

=>Syntax:- os.removedirs("folders Hierarchy")

=>If folders Hierarchy does not exists then we get FileNotFoundError.

=>If the folder hierarchy contains any files or sub folder then it is not possible to remove folders hierarchy and we get OSError.

Examples:

```
#removemdirsex.py  
import os  
try:  
    os.removedirs("D:\\India\\Hyd\\Ameerpet\\Python\\django")  
    print("Folders Hierarhcy removed successfully--verify")  
except FileNotFoundError:  
    print("Specified Folders does not exists")  
except OSError:  
    print("Specified Folders is not empty--not possible to remove")
```

6) rename a folder:

=>To rename a folder , we use rename() of os module

=>Syntax: os.rename("existing folder name", "New Folder name")

=>If existing folder name does not exists then we get FileNotFoundError.

Examples:

```
#renamedirex.py
import os
try:
    os.rename("c:\\mango","c:\\india")
    print("Folder renamed successfully--verify")
except FileNotFoundError:
    print("Folders does not exists")
```

7) list files in folder :

=>To list the files in folder , we use listdir() of os module

=>Syntax:- listobj=os.listdir("folder name")

=>if the "folder name" does not exists then we get FileNotFoundError

Examples:

```
#Program for listing files
#listfiles.py
import os
try:
    foldername=input("Enter Folder Name:")
    listfiles=os.listdir(foldername)
    print("-"*50)
    print("Number of file in '{}' folder={}".format(foldername, len(listfiles)))
    print("-"*50)
    for filename in listfiles:
        print("\t{}".format(filename))
    print("-"*50)
except FileNotFoundError:
    print("Folders does not exists:")
```

#Program for removing a folder

```
#rmdir.py
import os
try:
    os.rmdir("D:\\hyd")
    print("Folder removed successfully--verify")
except FileNotFoundError:
    print("Specified Folders does not exists")
except OSError:
    print("Specified Folder is not empty--not possible to remove")
```

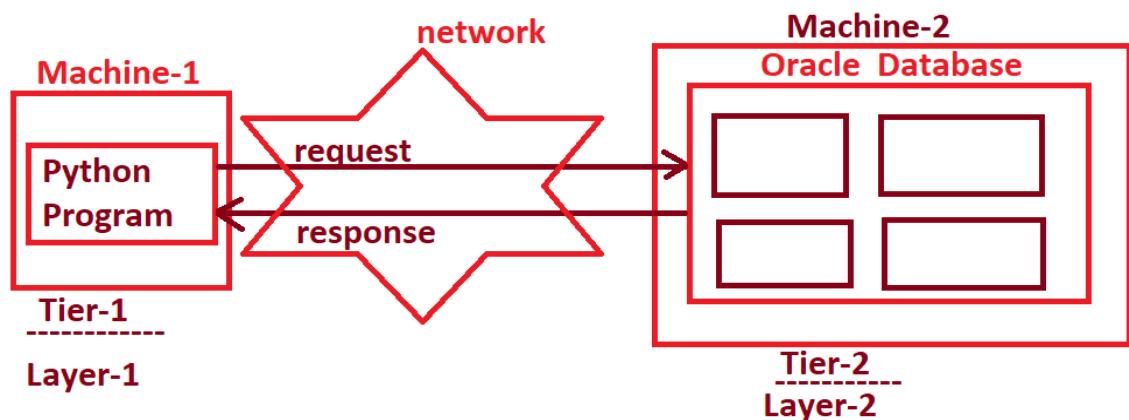
#Program for searching for files

```
#searchfile.py
import os
```

```
try:  
    foldername=input("Enter Folder Name:")  
    listfiles=os.listdir(foldername)  
    print("-"*50)  
    print("Number of file in '{}' folder={}".format(foldername, len(listfiles)))  
    print("-"*50)  
    for filename in listfiles:  
        print("\t{}".format(filename))  
    print("-"*50)  
    fname=input("Enter file name to search:")  
    if (fname in listfiles):  
        print("{} file exists in {}".format(fname, foldername))  
    else:  
        print("{} file does not exists in {}".format(fname, foldername))  
except FileNotFoundError:  
    print("Folders does not exists:")
```

=>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 password 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.

=>Programmatically, 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 performing 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

Python DataBase Communication (PDBC)

=>Even we achieved the Data Persistence 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 Software for achieving the data persistence, we get the following advantages.

1. DataBase software are Fully Secured because they provides User Name and password

2. To Process or Extract or Querying the data from DataBase Software is very easy bcoz the data present in tables of Database software are qualified with Column Names.

3. Data Base Software are able to store large Volume of Data.
 4. Data Base Software are InDependent from OS.
-

=>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
```

#This program test the connection with Oracle

#Testcon1.py

import cx_Oracle

try:

```
    con=cx_Oracle.connect("scott/tiger@localhost/orcl")  
    print("\nType of con=",type(con)) # <class 'cx_Oracle.Connection'>  
    print("\nPython Program obtains connection from Oracle DB")
```

except cx_Oracle.DatabaseError as db:

```
    print("\nProblem in database:", db)
```

#This program test the connection with Oracle

#Testcon2.py

import cx_Oracle # step-1

try:

```

con=cx_Oracle.connect("scott/tiger@127.0.0.1/orcl") # step-2
except cx_Oracle.DatabaseError as db:
    print("\nProblem in database:", db)
else:
    print("\nType of con=",type(con)) # <class 'cx_Oracle.Connection'>
    print("\nPython Program obtains connection from Oracle DB")
finally:
    con.close() # step-6
    print("\nConnection Closed ")

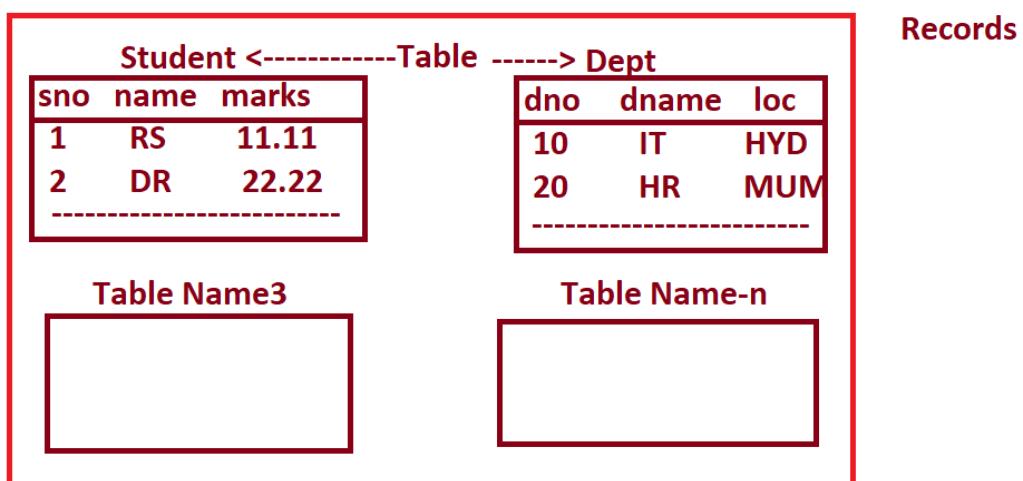
```

```

#This program create an object of cursor
#cursorex1.py
import cx_Oracle
con=cx_Oracle.connect("scott/tiger@127.0.0.1/orcl")
print("\nPython Program got collection from Oracle DB:")
cur=con.cursor()
print("\nPython creates an object of cursor")
print("Type of cur=",type(cur)) # <class 'cx_Oracle.Cursor'>

```

View about Oracle Database <---Collection of table<--Collection



1) DDL statements (Data Definition Language)

=>The purpose of DDL statements is that "To create , alter and dropping 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 dropping or deleting the entire table.

=>Syntax:- **SQL> drop table table-name**

=>Example: delete / drop student table

SQL> drop table student;

#write a python program which will create employee table with empno,empname and salary in oracle database

#tablecreate.py

import cx_Oracle # step-1

try:

con=cx_Oracle.connect("scott/tiger@localhost/orcl") #step-2

cur=con.cursor() # step-3

#design the query and execute----step-4

ctq="create table employee(eno number(2) primary key,ename varchar2(10) not null, sal number(8,2)) "

cur.execute(ctq)

print("Employee Table Created Successfully in Oracle DB--verify") # step-5

except cx_Oracle.DatabaseError as db:

print("\nProblem in database:", db)

#write a python program which will alter the employee table,change the column size and add new cloumn

#alterwithadd.py

import cx_Oracle # step-1

```
try:  
    con=cx_Oracle.connect("scott/tiger@localhost/orcl") #step-2  
    cur=con.cursor() # step-3  
    #design the query and execute---step-4  
    aq="alter table employee add(cname varchar2(10))"  
    cur.execute(aq)  
    print("Employee Table altered Sucecssfully in Oracle DB--verify") # step-5  
except cx_Oracle.DatabaseError as db:  
    print("\nProblem in database:", db)
```

```
#write a python program which will alter the employee table,change the column size  
and add new cloumn  
#alterwithmodify.py  
import cx_Oracle # step-1  
try:  
    con=cx_Oracle.connect("scott/tiger@localhost/orcl") #step-2  
    cur=con.cursor() # step-3  
    #design the query and execute---step-4  
    aq="alter table employee modify(eno number(3),ename varchar2(15 ))"  
    cur.execute(aq)  
    print("Employee Table altered Sucecssfully in Oracle DB--verify") # step-5  
except cx_Oracle.DatabaseError as db:  
    print("\nProblem in database:", db)
```

```
#write a python program which will alter the employee table,change the column size  
and add new cloumn  
#droptable.py  
import cx_Oracle # step-1  
try:  
    con=cx_Oracle.connect("scott/tiger@localhost/orcl") #step-2  
    cur=con.cursor() # step-3  
    #design the query and execute---step-4  
    cur.execute("drop table student")  
    print("Student Table Dropped Sucecssfully in Oracle DB--verify") # step-5  
except cx_Oracle.DatabaseError as db:  
    print("\nProblem 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 database 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 for 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 and sal<6.0;

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 Particular 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;

#write a python program which will insert employee number, ename,salary and company name as a record in employee table of oracle database

#recordinsertex1.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
    iq="insert into employee values(103,'Ajay',5.5,'TCS')"
    cur.execute(iq)
    con.commit()
    print("Number of records inserted :{}".format(cur.rowcount)) # 1
    print("Employee Record Inserted successfully in Employee table:")

except cx_Oracle.DatabaseError as db:
    print("Problem in Database:",db)

```

"""Note:

 in cur object, we have rowcount, which will give number of records updated in database.

```

#write a python program which will insert employee number, ename,salary and
company name as a record in employee table of oracle database
#emp.py--file name and acts as module name
import cx_Oracle # step-1
def empinsert():
    try:
        con=cx_Oracle.connect("scott/tiger@localhost/orcl") #step-2
        cur=con.cursor() # step-3
        #design and execute the query
        iq="insert into employee values(105,'Siva',4.5,'HCL')"
        cur.execute(iq)
        con.commit()
        print("Number of records inserted :{}".format(cur.rowcount)) # 1
        print("Employee Record Inserted successfully in Employee table:")

    except cx_Oracle.DatabaseError as db:
        print("Problem in Database:",db)

```

```

#write a python program which will insert employee number, ename,salary and
company name as a record in employee table of oracle database
#recordinsertex1.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
iq="insert into employee values(103,'Ajay',5.5,'TCS')"
cur.execute(iq)
con.commit()
print("Number of records inserted :{}".format(cur.rowcount)) # 1
print("Employee Record Inserted successfully in Employee table:")
except cx_Oracle.DatabaseError as db:
    print("Problem in Database:",db)

```

"""Note:

in cur object, we have rowcount, which will give number of records updated in database.

#write a python program which will insert employee number, ename,salary and company name as a record in employee table of oracle database

#emp1.py—file name and acts as module name

import cx_Oracle # step-1

def empinsert():

try:

 while(True):

 try:

 con=cx_Oracle.connect("scott/tiger@localhost/orcl") #step-2

 cur=con.cursor() # step-3

 #accept employee data

 empno=int(input("Enter Employee Number:"))

 ename=input("Enter Employee Name:")

 sal=float(input("Enter Employee Salary:"))

 cname=input("Enter Employee Company Name:")

 #design the query and execute

 cur.execute("insert into employee values(%d,'%s',%f,'%s')

 " %(empno,ename,sal,cname))

 con.commit()

 print("{} Employee Record Inserted :".format(cur.rowcount))

 print("-"*50)

 ch=input("Do u want to another Record(yes/no):")

 if(ch.lower() == "no"):

 print("Thanks for using this program")

 break

 except ValueError:

 print("\nDon't enter strs/symbols/alpha-numerics for Emp

Number and salary..")

 except cx_Oracle.DatabaseError as db:

 print("Problem in Database:",db)

#recordinsertex3.py

from emp1 import empinsert

empinsert()

```

#write a python program which will delete a record from employee table based on
employee number.
#recorddeleteex.py
import cx_Oracle
def empdelete():
    try:
        while(True):
            try:
                con=cx_Oracle.connect("scott/tiger@localhost/orcl") #step-2
                cur=con.cursor() # step-3
                #accept employee data
                empno=int(input("Enter Employee Number:"))
                #design and execute the query
                cur.execute("delete from employee where eno=%d" %empno)
                con.commit()
                if(cur.rowcount>0):
                    print("{} Employee Record(s) delete from employee
table:".format(cur.rowcount))
                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("Thanks for using this program")
                        break
            except ValueError:
                print("\nDon't enter strs/symbols/alpha-numerics for Emp Number ")
            except cx_Oracle.DatabaseError as db:
                print("Problem in Database:",db)

```

```

#main program
empdelete()

```

**#write a python program which will update salary of an emp,cname of an emp
based on emp no**

```

#recordupdateex.py
import cx_Oracle
def empupdate():
    try:
        while(True):
            try:
                con=cx_Oracle.connect("scott/tiger@localhost/orcl") #step-2
                cur=con.cursor() # step-3
                #accept employee data
                empno=int(input("Enter Employee Number:"))
                newsal=float(input("Enter New salary for Employee:"))
                newcomp=input("Enter new Comp Name of Employee:")
                #design and execute the query
                cur.execute("update employee set sal=%f , cname='%s' where eno=%d"
                %(newsal,newcomp,empno) )
                con.commit()

```

```
if(cur.rowcount>0):
    print("{} Employee Record(s) updated in employee
table:".format(cur.rowcount))
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("Thanks for using this program")
        break
except ValueError:
    print("\nDon't enter strs/symbols/alpha-numerics for Emp Number and
Salary ")
except cx_Oracle.DatabaseError as db:
    print("Problem in Database:",db)

#main program
empupdate()
```

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 tablename1, tablename2...Colname-n from table_name;

SQL> select tablename1, tablename2...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()

#write a python program which will read all the records from employee table–select Query

#selectrecordsex1.py

import cx_Oracle

```
def recordsselect():
    try:
        con=cx_Oracle.connect("scott/tiger@localhost/orcl")
        cur=con.cursor()
        cur.execute("select * from employee")
        while(True):
            rec=cur.fetchone()
            if(rec!=None):
                for val in rec:
                    print("{}".format(val),end=" ")
                    print()
            else:
                break
    except cx_Oracle.DatabaseError as db:
        print("Problem In Database:", db)
```

```
#main porogram
recordsselect()
```

#write a python program which will read all the records from employee table–select Query

#selectrecordsex2.py

```
import cx_Oracle
def recordsselect():
    try:
        con=cx_Oracle.connect("scott/tiger@localhost/orcl")
        cur=con.cursor()
        cur.execute("select * from employee")
        records=cur.fetchmany(3)
        for record in records:
            for val in record:
                print("{}".format(val), end=" ")
                print()
    except cx_Oracle.DatabaseError as db:
        print("Problem In Database:", db)
```

```
#main porogram
recordsselect()
```

#write a python program which will read all the records from employee table–select Query

#selectrecordsex3.py

```
import cx_Oracle
def recordsselect():
    try:
```

```

con=cx_Oracle.connect("scott/tiger@localhost/orcl")
cur=con.cursor()
cur.execute("select * from employee")
records=cur.fetchall()
for record in records:
    for val in record:
        print("{}".format(val),end=" ")
    print()
except cx_Oracle.DatabaseError as db:
    print("Problem In Database:", db)

#main porogram
recordsselect()



---


#write a python program which will read all the records from employee table–select Query
#colnames.py
import cx_Oracle
def colnames():
    try:
        con=cx_Oracle.connect("scott/tiger@localhost/orcl")
        cur=con.cursor()
        cur.execute("select * from emp")
        colinfos=cur.description
        colnames= [colinfo[0] for colinfo in colinfos]
        print("*50")
        for colname in colnames:
            print("{}".format(colname),end=" ")
        print()
        print("*50)
    except cx_Oracle.DatabaseError as db:
        print("Problem In Database:", db)
#main porogram
colnames()



---


#write a python program which will read all the records from employee table–select Query
#colnamesdir.py
import cx_Oracle
def colnames():
    try:
        con=cx_Oracle.connect("scott/tiger@localhost/orcl")
        cur=con.cursor()
        tname=input("Enter table name:")
        cur.execute("select * from %s" %tname)
        print("*50)
        for colname in [colinfo[0] for colinfo in cur.description]:
            print("{}".format(colname),end=" ")
        print()
        print("*50)
    except cx_Oracle.DatabaseError as db:
        print("Problem In Database:", db)

```

```
#main porogram
colnames()

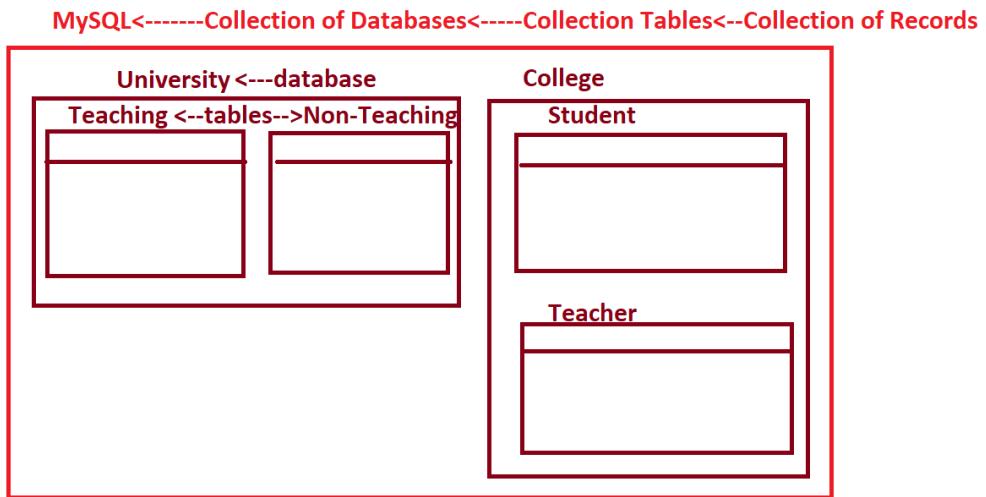

---


#write a python program which will display all the records of any table along with
column names
#tabledataex.py
import cx_Oracle
def allrecords():
    try:
        con=cx_Oracle.connect("scott/tiger@localhost/orcl")
        cur=con.cursor()
        tname=input("Enter table name:")
        cur.execute("select * from %s" %tname)
        #printing column names
        print("*70)
        for colname in [colinfo[0] for colinfo in cur.description]:
            print("{}".format(colname),end=" ")
        print()
        print("*70)
        #display records of table
        #nor=0
        records=cur.fetchall()
        for record in records:
            #nor=nor+1
            for val in record:
                print("{}".format(val),end=" ")
            print()
            print("*70)
            print("Total Number of Records={}".format(len(records)))
            print("*70)
    except cx_Oracle.DatabaseError as db:
        print("Problem In Database:", db)
#main porogram
allrecords()
```

Introduction to MySQL

=>MySQL is one RDBMS Software

- =>MySQL is of the Freeware and Open Source
 - =>MySQL is suitable for small and large scale application for Data Persistency.
 - =>MySQL is Very Fast , reliable and Scalable and Easy to use for maintaining Data Persistency.
 - =>MySQL was first released in the year 1995
-



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 carry 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")

#This program obtains connection MySQL DB

#contestmysql.py

import mysql.connector

conobj=mysql.connector.connect(host="localhost",

print("type of conobj=",type(conobj))#<class
'mysql.connector.connection.MySQLConnection'>
print("python program got connection from MYSQL")

#This program create as an object cursor

#cursorex.py

import mysql.connector

conobj=mysql.connector.connect(host="localhost",user="root",passwd="root")

print("python program got connection from MYSQL")

cur=conobj.cursor()

print("cursor object created:")

print("type cur=",type(cur)) # type cur= <class
'mysql.connector.cursor.MySQLCursor'>

#Program for creating a database on the name of "batch9am"

#databaserecreate.py

import mysql.connector

try:

conobj=mysql.connector.connect(host="localhost", user="root", passwd="root")

```
cur=conobj.cursor()
#design the query and execute
dq="create database batch9am"
cur.execute(dq)
print("Data Base created in MySQL and Verify:")
except mysql.connector.DatabaseError as d:
    print("Problem in Data Base:",d)



---

#write a python program which will delete a record from employee table based on employee number.
#recorddeleteex.py
import mysql.connector
def empdelete():
    try:
        while(True):
            try:
                con=mysql.connector.connect(host="localhost",
                    user="root", passwd="root", database="batch9am" )
                cur=con.cursor() # step-3
                #accept employee data
                empno=int(input("Enter Employee Number:"))
                #design and execute the query
                cur.execute("delete from employee where eno=%d" %empno)
                con.commit()
                if(cur.rowcount>0):
                    print("{} Employee Record(s) delete from employee table:".format(cur.rowcount))
                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("Thanks for using this program")
                    break
            except ValueError:
                print("\nDon't enter strs/symbols/alpha-numerics for Emp Number ")
    except mysql.connector.DatabaseError as db:
        print("Problem in Database:",db)



---

#main program
empdelete()
```

```
#write a python program which will insert employee number, ename,salary and company name as a record in employee table of oracle database
#recordinsertex.py--
import mysql.connector # step-1
def empinsert():
    try:
        while(True):
```

```

try:
    con=mysql.connector.connect(host="localhost",
                                user="root",
                                passwd="root",
                                database="batch9am" )

    cur=con.cursor() # step-3
    #accept employee data
    empno=int(input("Enter Employee Number:"))
    ename=input("Enter Employee Name:")
    sal=float(input("Enter Employee Salary:"))
    cname=input("Enter Employee Company Name:")
    #design the query and execute
    cur.execute("insert into employee
values(%d,'%s',%f,'%s') " %(empno,ename,sal,cname) )
    con.commit()
    print("{} Employee Record Inserted
:".format(cur.rowcount))
    print("-"*50)
    ch=input("Do u want to another Record(yes/no):")
    if(ch.lower() == "no"):
        print("Thanks for using this program")
        break
    except ValueError:
        print("\nDon't enter strs/symbols/alpha-numerics
for Emp Number and salary..")
    except mysql.connector.DatabaseError as db:
        print("Problem in Database:",db)

#main program
empinsert()



---


#write a python program which will update salary of an emp,cname of an emp
based on emp no
#recordupdateex.py
import mysql.connector
def empupdate():
    try:
        while(True):
            try:
                con=mysql.connector.connect(host="localhost", user="root",
                                            passwd="root", database="batch9am" )
                cur=con.cursor() # step-3
                #accept employee data
                empno=int(input("Enter Employee Number:"))
                newsal=float(input("Enter New salary for Employee:"))
                newcomp=input("Enter new Comp Name of Employee:")
                #design and execute the query
                cur.execute("update employee set sal=%f ,
cname='%s' where eno=%d" %(newsal,newcomp,empno) )
                con.commit()
                if(cur.rowcount>0):

```

```
        print("{} Employee Record(s) updated in employee
table:".format(cur.rowcount))
    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("Thanks for using this program")
            break
    except ValueError:
        print("\nDon't enter strs/symbols/alpha-numerics for Emp Number and
Salary ")
    except mysql.connector.DatabaseError as db:
        print("Problem in Database:",db)

#main program
empupdate()
```

```
#write a python program which will create employee table with column name
eno,ename,sal and cname in MYSQL

import mysql.connector
try:
    conobj=mysql.connector.connect(host="localhost", user="root", passwd="root",
                                    database="batch9am")
    cur=conobj.cursor()
    #design the query and execute
    tq="create table student (sno int primary key, sname varchar(10) not null, marks
real not null, cname varchar(10) not null )"
    cur.execute(tq)
    print("Table created in batch9am database of MySQL and Verify:")
except mysql.connector.DatabaseError as d:
    print("Problem in Data Base:",d)
```

```
#write a python program which will display all the records of any table along with
column names

import mysql.connector
def allrecords():
    try:
        con=mysql.connector.connect(host="localhost", user="root", passwd="root",
                                    database="batch9am")
        cur=con.cursor()
        tname=input("Enter table name:")
        cur.execute("select * from %s" %tname)
        #printing column names
        print("*70)
```

```
for colname in [colinfo[0] for colinfo in cur.description]:
    print("{}{}".format(colname),end=" ")
print()
print("*70)
#display records of table
records=cur.fetchall()
for record in records:
    for val in record:
        print("{}{}".format(val),end=" ")
    print()
    print("*70)
    print("Total Number of Records={}".format(len(records)))
    print("*70)
except mysql.connector.DatabaseError as db:
    print("Problem In Database:", db)
#main porogram
allrecords()
```

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 operations 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=strobject.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 occurrence 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=strobject.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 original 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
```

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 otherwise it returns False.

=>Syntax:- varname=strobj.issapce()
 (or)
 strobj.isapce()

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 concatenating all the sequence of values which are available in the form str

Syntax:- varname=strobj1.join(iterable obj)

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 on splitting value. The default splitting value is space
=>This Function returns splitting values in the form of list.

Syntax:- listobj=strobj.split()
OR
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']
```

#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)))
```

#write a python program which will accept a line of text and separate them with vowels and consonants and special symbols

```
#alhabets.py
line=input("Enter a line of text:") # a3$b56hq3rt$#P"
vs=[]
cs=[]
```

```

ds=[]
ss=[]
for ch in line:
    if ch in ['a','e','i','o','u','A','E','I','O','U']:
        vs.append(ch)
    elif( ch not in ['a','e','i','o','u','A','E','I','O','U'] and ch.isalpha() ):
        cs.append(ch)
    elif( ch not in ['a','e','i','o','u','A','E','I','O','U'] and ch.isdigit() ):
        ds.append(ch)
    else:
        ss.append(ch)
else:
    print("-----")
    print("Given Line={}".format(line))
    print("Vowels List={}".format(vs))
    print("Cons. List={}".format(cs))
    print("Digits List={}".format(ds))
    print("Special Symbols List={}".format(ss))
    print("-----")

```

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
-

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 " (or) Advantages of OOPs

=>In Object, we can store large Volume of Data and achieves Platform Independency (Python)

=>The confidential Data can be transferred between multiple remote machine in the form cipher text (Encrypted format). So that security can be enhanced (Improved).

=>The Large of Volume of Data Transferred 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.

#Program for studentmarks report generation with Classes and Object along Database.

#StudentMarksReportMySQL.py

import mysql.connector

class StudentMarksReportMySQL:

 def getstudentdetails(self): #getting stno,name,CM,CPPM,PYTM

 #validation of Student Number

 while(True):

 self.sno=int(input("Enter Student Number:"))

 if(self.sno>0) and (self.sno<=100):

 break

 self.sname=input("Enter Student Name:")

 #validation of marks in C

 while(True):

```

        self.cm=int(input("Enter Marks in C:"))
        if(self.cm>=0) and (self.cm<=100):
            break
    #validation of marks in CPP
    while(True):
        self.cppm=int(input("Enter Marks in CPP:"))
        if(self.cppm>=0) and (self.cppm<=100):
            break
    #validation of marks in PYTHON
    while(True):
        self.pytm=int(input("Enter Marks in PYTHON:"))
        if(self.pytm>=0) and (self.pytm<=100):
            break
    def decidegrade(self): # cal totmarks,percentage and grade
        #calculate total and percantage of marks
        self.totmarks=self.cm+self.cppm+self.pytm
        self.percent=(self.totmarks/300)*100
        #decide the grade
        if(self.cm<40) or (self.cppm<40) or (self.pytm<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 storedtsudentdata(self): # Python Data Base Communication code
        try:
            con=mysql.connector.connect(host="localhost",
                                         user="root",
                                         password="",
                                         database="student")
            cur=con.cursor()
            #design the query and execute
            iq="insert into result values(%d,'%s',%d,%d,%d,%d,%f,'%s') "
            cur.execute(iq
                        %(self.sno,self.sname,self.cm,self.cppm,self.pytm,self.totmarks,self.percent,self.grade))
            con.commit()
            print("\n{} Student Record Inserted in Result Table--"
                  .format(cur.rowcount))
        except mysql.connector.DatabaseError as db:
            print("Problem in Database:",db)

#main program
s=StudentMarksReportMySQL()
```

```
s.getstudentdetails()
s.decidegrade()
s.storestsudentdata()
```

#Program for studentmarks report generation with Classes and Object along Database.

```
import cx_Oracle
class StudentMarksReport:
    def getstudentdetails(self): #getting stno,name,CM,CPPM,PYTM
        #validation of Student Number
        while(True):
            self.sno=int(input("Enter Student Number:"))
            if(self.sno>0) and (self.sno<=100):
                break
        self.sname=input("Enter Student Name:")
        #validation of marks in C
        while(True):
            self.cm=int(input("Enter Marks in C:"))
            if(self.cm>=0) and (self.cm<=100):
                break
        #validation of marks in CPP
        while(True):
            self.cppm=int(input("Enter Marks in CPP:"))
            if(self.cppm>=0) and (self.cppm<=100):
                break
        #validation of marks in PYTHON
        while(True):
            self.pytm=int(input("Enter Marks in PYTHON:"))
            if(self.pytm>=0) and (self.pytm<=100):
                break
        print(self.sno,self.sname,self.cm,self.cppm,self.pytm)

    def decidegrade(self): # cal totmarks,percentage and grade
        #calculate total and percentage of marks
        self.totmarks=self.cm+self.cppm+self.pytm
        self.percent=(self.totmarks/300)*100
        #decide the grade
        if(self.cm<40) or (self.cppm<40) or (self.pytm<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 storedtsudentdata(self): # Python Data Base Communication code
    try:
        con=cx_Oracle.connect("scott/tiger@localhost/orcl")
        cur=con.cursor()
        #design the query and execute
        iq="insert into result values(%d,'%s',%d,%d,%d,%d,%f,'%s') "
        cur.execute(iq)
        %(self.sno,self.sname,self.cm,self.cppm,self.pytm,self.totmarks,self.percent,self.gra
        de) )
        con.commit()
        print("\n{} Student Record Inserted in Result Table--"
verify".format(cur.rowcount))
    except cx_Oracle.DatabaseError as db:
        print("Problem in Database:",db)

#main program
s=StudentMarksReport()
s.getstudentdetails()
s.decidegrade()
s.storedtsudentdata()

```

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 customized 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 will be created when we create an object.
-
-

Syntax for defining a class in python

```
class <Class Name>:  
    def instancemethodname(self,list of formal params if any):  
        -----  
        ---Specify Instance Data Members---  
        -->Performs Specific Operations----  
    @classmethod  
    def classlevelmethodname(cls, list of formal params if any):  
        -----  
        ---Specify Class Level Data Members---  
        -->Performs 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

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
=>Instanace Data memebrs memory space created Every Time when an object created.
=>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 memory space created Only Once Irrespective number of objects created.

=>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 available to all the objects 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

#This program stores stno,name and marks

#studentex1.py

class Student:pass # here Student is called Programmer-defined data type

#main program

s=Student()

print("content of s=",s.__dict__) # {}

print("Number of values in s=", len(s.__dict__))

#This program stores stno,name and marks

#studentex2.py

class Student:pass # here Student is called Programmer-defined data type

#main program

s1=Student()

s2=Student()

print("Content of s1 before adding the data=",s1.__dict__)

print("Content of s2 before adding the data=",s2.__dict__)

print("-"*50)

```
#add the data to s1
s1.sno=100
s1.sname="RS"
s1.marks=22.22 # here sno,sname and marks are called Instance data members
#add the data to s2
s2.sno=101
s2.sname="TR"
s2.marks=33.33 # here sno,sname and marks are called Instance data members
print("Content of s1 after adding the data=",s1.__dict__)
print("Content of s2 after adding the data=",s2.__dict__)
```

#This program stores stno,name and marks

#studentex3.py

```
class Student:pass # here Student is called Programmer-defined data type
```

```
#main program
s1=Student()
s2=Student()
print("Content of s1 before adding the data=",s1.__dict__)
print("Content of s2 before adding the data=",s2.__dict__)
print("-"*50)
#add the data to s1
s1.sno=100
s1.sname="RS"
s1.marks=22.22 # here sno,sname and marks are called Instance data members
#add the data to s2
s2.sno=101
s2.sname="TR"
s2.marks=33.33 # here sno,sname and marks are called Instance data members
print("-"*50)
print("First Student Information:")
print("-"*50)
print("Student Number=",s1.sno)
print("Student Name=",s1.sname)
print("Student Marks=",s1.marks)
print("-"*50)
print("Second Student Information:")
print("-"*50)
print("Student Number=",s2.sno)
print("Student Name=",s2.sname)
print("Student Marks=",s2.marks)
print("-"*50)
```

#This program stores stno,name and marks

#studentex4.py

```
class Student:pass # here Student is called Programmer-defined data type
```

#main program

```
s1=Student()
```

```
s2=Student()
```

```
print("Content of s1 before adding the data=",s1.__dict__)
print("Content of s2 before adding the data=",s2.__dict__)
```

```
print("-*50)
#add the data to s1
print("Enter First Student Information")
print("-*50)
s1.sno=int(input("Enter Student Number:"))
s1.sname=input("Enter Student Name:")
s1.marks=float(input("Enter Student Marks:")) # here sno,sname and marks are
called Instance data members
#add the data to s2
print("Enter Second Student Information")
print("-*50)
s2.sno=int(input("Enter Student Number:"))
s2.sname=input("Enter Student Name:")
s2.marks=float(input("Enter Student Marks:")) # here sno,sname and marks are
called Instance data members
print("-*50)
print("First Student Information:")
print("-*50)
print("Student Number=",s1.sno)
print("Student Name=",s1.sname)
print("Student Marks=",s1.marks)
print("Student Course=",s1.crs)
print("-*50)
print("Second Student Information:")
print("-*50)
print("Student Number=",s2.sno)
print("Student Name=",s2.sname)
print("Student Marks=",s2.marks)
print("Student Course=",s2.crs)
print("-*50)
```

#This program stores stno,name and marks

#studentex5.py

```
class Student: # here Student is called Programmer-defined data type
    crs="PYTHON" # here crs is called Class Level Data members
```

```
#main program
s1=Student()
s2=Student()
print("Content of s1 before adding the data=",s1.__dict__)
print("Content of s2 before adding the data=",s2.__dict__)
print("-*50)
#add the data to s1
print("Enter First Student Information")
print("-*50)
s1.sno=int(input("Enter Student Number:"))
s1.sname=input("Enter Student Name:")
s1.marks=float(input("Enter Student Marks:")) # here sno,sname and marks are
called Instance data members
#add the data to s2
```

```
print("Enter Second Student Information")
print("-"*50)
s2.sno=int(input("Enter Student Number:"))
s2.sname=input("Enter Student Name:")
s2.marks=float(input("Enter Student Marks:")) # here sno,sname and marks are
called Instance data members
print("-"*50)
print("First Student Information:")
print("-"*50)
print("Student Number=",s1.sno)
print("Student Name=",s1.sname)
print("Student Marks=",s1.marks)
print("Student Course=",Student.crs) # accessing Class Level Data Members w.r.t
Class Name
print("-"*50)
print("Second Student Information:")
print("-"*50)
print("Student Number=",s2.sno)
print("Student Name=",s2.sname)
print("Student Marks=",s2.marks)
print("Student Course=",Student.crs) # accessing Class Level Data Members w.r.t
Class Name
print("-"*50)
```

#This program stores stno,name and marks

#studentex6.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()
print("Enter Second Student Information:")
s2.readstuddata()
print("First Student Information:")
s1.dispstuddata()
print("Second Student Information:")
s2.dispstuddata()
```

```
#This program stores stno,name and marks
#studentex7.py
class Student:
    crs="PYTHON" # Class Level Data Members
    def readstuddata(self): # Instance Method
        print("-"*50)
        self.sno=int(input("Enter Student Number:"))
        self.sname=input("Enter Student Name:")
        self.marks=float(input("Enter Student Marks:"))
        #here self.sno,self.sname, self.marks are called Instance Data
members.
        print("-"*50)
    def dispstuddata(self): # Instance Method
        print("-"*50)
        print("Student Number:{}".format(self.sno))
        print("Student Name:{}".format(self.sname))
        print("Student Marks:{}".format(self.marks))
        print("Student Course:{}".format(Student.crs))
        print("-"*50)

#main program
s1=Student()
s2=Student()
print("Enter First Student Information:")
s1.readstuddata()
print("Enter Second Student Information:")
s2.readstuddata()
print("First Student Information:")
s1.dispstuddata()
print("Second Student Information:")
s2.dispstuddata()
```

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):
```

—Common 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.
=>static 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()

#Program for calculating Simple Interest by using Classes and Objects

#SimpleInt.py

class SimpleInt:

```
    def readvalues(self):  
        self.p=float(input("Enter Principle Amount:"))  
        self.t=float(input("Enter Time:"))  
        self.r=float(input("Enter Rate Of Interest:"))  
    def calSimpleInt(self):  
        self.si=(self.p*self.t*self.r)/100  
    def dispresult(self):  
        print("*50)  
        print("Principle Amount:{} ".format(self.p))  
        print("Time :{} ".format(self.t))  
        print("Rate of Interest :{} ".format(self.r))  
        print("Simple Interest:{} ".format(self.si))  
        print("*50)
```

#main program

si=SimpleInt()

si.readvalues()

si.calSimpleInt()

si.dispresult()

#Program for calculating Simple Interest by using Classes and Objects

#SimpleInt1.py

class SimpleInt:

```
    def readvalues(self):  
        self.p=float(input("Enter Principle Amount:"))  
        self.t=float(input("Enter Time:"))  
        self.r=float(input("Enter Rate Of Interest:"))  
        self.calSimpleInt()  
        self.dispresult() # One Instance Method calling another Instance
```

Method

```
    def calSimpleInt(self):  
        self.si=(self.p*self.t*self.r)/100
```

```

def dispresult(self):
    print("*50")
    print("Principle Amount:{}".format(self.p))
    print("Time :{}".format(self.t))
    print("Rate of Interest :{}".format(self.r))
    print("Simple Interest:{}".format(self.si))
    print("*50)

#main program
si=SimpleInt()
si.readvalues()

```

#This program demonstartes about class Level method

#ClassLevelMethodEx1.py

```

class Employee:
    @classmethod
    def getcompnames(cls):
        cls.cname="IBM" # Class Level Data Member
    @classmethod
    def getcomploc(cls):
        cls.cloc="HYD" # Class Level Data Member

    def getempdet(self):
        print("-*50")
        self.eno=int(input("Enter Employee Number:"))
        self.ename=input("Enter Employee Name:")
        self.sal=float(input("Enter Employee Salary:"))
        print("-*50)

    def dispempdet(self):
        print("-*50")
        print("Employee Number:{}".format(self.eno))
        print("Employee Name:{}".format(self.ename))
        print("Employee Salary:{}".format(self.sal))
        print("Employee Comp Name:{}".format(Employee.cname))
        print("Employee Comp Loc:{}".format(Employee.cloc))
        print("-*50)

#main program
Employee.getcompnames()
Employee.getcomploc() # calling Class Level Methods
e1=Employee()
e2=Employee()
print("Enter First Employee Information:")
e1.getempdet()
print("Enter Second Employee Information:")
e2.getempdet()
print("\nFirst Employee Information:")
e1.dispempdet()
print("\nSecond Employee Information:")

```

e2.dispempdet()

```
#This program demonstartes about class Level method
#ClassLevelMethodEx2.py
class Employee:
    @classmethod
    def getcompnames(cls):
        cls.cname="IBM" # Class Level Data Member
        cls.getcomploc() # or Employee.getcomploc()---calling Class
Level Method from another Class Level Method
    @classmethod
    def getcomploc(cls):
        cls.cloc="HYD" # Class Level Data Member

    def getempdet(self):
        print("-"*50)
        self.eno=int(input("Enter Employee Number:"))
        self.ename=input("Enter Employee Name:")
        self.sal=float(input("Enter Employee Salary:"))
        print("-"*50)
    def dispempdet(self):
        print("-"*50)
        print("Employee Number:{}".format(self.eno))
        print("Employee Name:{}".format(self.ename))
        print("Employee Salary:{}".format(self.sal))
        print("Employee Comp Name:{}".format(Employee.cname))
        print("Employee Comp Loc:{}".format(Employee.cloc))
        print("-"*50)

#main program
Employee.getcompnames()
e1=Employee()
e2=Employee()
print("Enter First Employee Information:")
e1.getempdet()
print("Enter Second Employee Information:")
e2.getempdet()
print("\nFirst Employee Information:")
e1.dispempdet()
print("\nSecond Employee Information:")
e2.dispempdet()
```

#This program demonstartes about class Level method

#ClassLevelMethodEx3.py

```
class Employee:
    @classmethod # Pre-defined decorator
    def getcompnames(cls):
        cls.cname="IBM" # Class Level Data Member
    @classmethod
    def getcomploc(cls):
        cls.cloc="HYD" # Class Level Data Member
```

```

def getempdet(self):
    print("-"*50)
    self.eno=int(input("Enter Employee Number:"))
    self.ename=input("Enter Employee Name:")
    self.sal=float(input("Enter Employee Salary:"))
    print("-"*50)
def dispempdet(self):
    print("-"*50)
    print("Employee Number:{}".format(self.eno))
    print("Employee Name:{}".format(self.ename))
    print("Employee Salary:{}".format(self.sal))
    self.getcompnames()
    self.getcomploc() # calling Class Level Methods from Instance
Methods
    print("Employee Comp Name:{}".format(Employee cname))
    print("Employee Comp Loc:{}".format(Employee cloc))
    print("-"*50)
#main program
e1=Employee()
e2=Employee()
print("Enter First Employee Information:")
e1.getempdet()
print("Enter Second Employee Information:")
e2.getempdet()
print("\nFirst Employee Information:")
e1.dispempdet()
print("\nSecond Employee Information:")
e2.dispempdet()

```

#This program demonstrates about class Level method

#ClassLevelMethodEx4.py

```

class Employee:
    @classmethod # Pre-defined decorator
    def getcompnames(cls):
        cls.cname="IBM" # Class Level Data Member
    @classmethod
    def getcomploc(cls):
        cls.cloc="HYD" # Class Level Data Member

    def getempdet(self):
        print("-"*50)
        self.eno=int(input("Enter Employee Number:"))
        self.ename=input("Enter Employee Name:")
        self.sal=float(input("Enter Employee Salary:"))
        print("-"*50)
    def dispempdet(self):
        print("-"*50)
        print("Employee Number:{}".format(self.eno))
        print("Employee Name:{}".format(self.ename))

```

```

print("Employee Salary:{}".format(self.sal))
print("Employee Comp Name:{}".format(Employee cname))
print("Employee Comp Loc:{}".format(Employee cloc))
print("-"*50)
#main program
e1=Employee()
e2=Employee()
print("Enter First Employee Information:")
e1.getempdet()
print("Enter Second Employee Information:")
e2.getempdet()
#calling class level methods w.r.t object name
e1.getcompnames()
e2.getcomploc()
print("\nFirst Employee Information:")
e1.dispempdet()
print("\nSecond Employee Information:")
e2.dispempdet()

```

--- **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()

Difference between classes and object:

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 particular 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 create an object after loading the class definition otherwise we get NameError
-

Constructors in Python

=>The purpose of Constructor is that "To Initialize the Object when an object created".

=>Initializing the object is nothing but placing our own values without leaving the object empty when an object is created.

Definition:

=>A Constructor is one of the Special Method, which is automatically or implicitly called by PVM during Object Creation and whose purpose is that to place our own values without leaving the object empty (Initialization of object).

Syntax for defining constructor in class:

def __init__(self, list of formal params if any):

Block of Statements–Initialization

Properties / rules / Characteristics of Constructor:

1. The Constructor will automatically / implicitly called by PVM during object creation

2. The Constructors are used for Initializing Objects
 3. The Name of the constructor is always `__init__(self,.....)`
 5. Constructors will not return value.
 4. Constructors are always Participating in Inheritance.
 5. Constructors can be Overridden
-

Types of Constructors in Python

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

1. Default or Parameter-Less or no-argument Constructor
 2. Parameterized Constructor
-

1. Default or Parameter-Less or no-argument Constructor:

=>**Definition:-** A constructor is said to be Default if and only if It never Takes any Formal

----- Parameters (except self).

=>The purpose of Default Constructor is that "To Initialize the multiple objects of same class with same Values".

=>**Syntax:-** `def __init__(self): # Default Constructor`

—Block of statement-Initlization-

Examples:

```
#TestEx1.py
class Test:
    def __init__(self): # Constructors--Object Initlization
        print("i am from Default Constructor")
        self.a=10
        self.b=20
        print("*50")
        print("Val of a={}".format(self.a))
        print("Val of b={}".format(self.b))
        print("*50)
#main program
t1=Test()                                # object creation
t2=Test()      # object creation
t3=Test()      # object creation
```

b) Parameterized Constructor

=>The purpose of 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–Initialization

Examples:

```
#TestEx2.py
class Test:
    def __init__(self,a,b): # Constructors--Object Initialization
        print("i am from Parameterized Constructor")
        self.a=a
        self.b=b
        print("*50")
        print("Val of a={}".format(self.a))
        print("Val of b={}".format(self.b))
        print("*50")
#main program
t1=Test(10,20)                                # object creation
t2=Test(100,200)                               # object creation
t3=Test(1,2)                                    # object creation
```

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.

```
#TestEx4.py
class Test:
    def __init__(self,a=1,b=2): # Constructors--Object Initialization
        print("i am from Default / Parameterized Constructor")
        self.a=a
        self.b=b
        print("*50")
        print("Val of a={}".format(self.a))
        print("Val of b={}".format(self.b))
        print("*50")
#main program
t1=Test()                                      # object creation
t2=Test(100,200)                             # object creation
t3=Test(20,30)
```

```
#StudEx1.py
class Student:
    def __init__(self): # Constructors
        print("i am from Constructor")
```

```
#main program
s=Student()      # object creation
print("content of s during object creation=",s.__dict__)


---


#StudEx2.py
class Student:
    def __init__(self): # Constructors--Object Initlization
        print("i am from Constructor")
        self.sno=10
        self.name="Raj"
        self.marks=44.44
```

```
#main program
s=Student()      # object creation
print("content of s during object creation=",s.__dict__)


---


```

```
#StudEx3.py
class Student:
    def __init__(self): # Constructors--Object Initlization
        print("i am from Default Constructor")
        self.sno=int(input("Enter Student Number:"))
        self.name=input("Enter Student Name:")
```

```
#main program
s1=Student()      # object creation
print("content of s1 during object creation=",s1.__dict__)
s2=Student()      # object creation
print("content of s2 during object creation=",s2.__dict__)


---


```

```
#StudEx4.py
class Student:
    def __init__(self,sno,sname): # Constructors--Object Initlization
        print("i am from Parameterized Constructor")
        self.sno=sno
        self.name=sname
    def dispstuddet(self):
        print("*50")
        print("Student Number={}".format(self.sno))
        print("Student Name={}".format(self.name))
```

```
#main program
s1=Student(10,"Rossum")      # object creation
s2=Student(20,"Gosling")      # object creation
s3=Student(30,"Travis")      # object creation
s1.dispstuddet()
s2.dispstuddet()
s3.dispstuddet()
```

```
#StudEx5.py
class Student:
    def __init__(self,sno,sname): # Constructors--Object Initlization
        print("i am from Parameterized Constructor")
        self.sno=sno
        self.name=sname
        print("*50")
```

```
        print("Student Number={}".format(self.sno))
        print("Student Name={}".format(self.name))
        print("*50)
#main program
s1=Student(10,"Rossum")      # object creation
s2=Student(20,"Gosling")     # object creation
s3=Student(30,"Travis")     # object creation
```

```
#TestEx1.py
class Test:
    def __init__(self): # Constructors--Object Initlization
        print("i am from Default Constructor")
        self.a=10
        self.b=20
        print("*50)
        print("Val of a={}".format(self.a))
        print("Val of b={}".format(self.b))
        print("*50)
#main program
t1=Test()                      # object creation
t2=Test()      # object creation
t3=Test()          # object creation
```

```
#TestEx2.py
class Test:
    def __init__(self,a,b): # Constructors--Object Initlization
        print("i am from Parameterized Constructor")
        self.a=a
        self.b=b
        print("*50)
        print("Val of a={}".format(self.a))
        print("Val of b={}".format(self.b))
        print("*50)
        return (self.a+self.b)
#main program
t1=Test(10,20)                  # object creation
t2=Test(100,200)     # object creation
t3=Test(1,2)          # object creation
```

```
#TestEx3.py
class Test:
    def __init__(self,a,b): # Constructors--Object Initlization
        print("i am from Parameterized Constructor")
        self.a=a
        self.b=b
        print("*50)
        print("Val of a={}".format(self.a))
        print("Val of b={}".format(self.b))
```

```

        print("=*50)
#main program
print("Enter Two Value for First Object:")
x=int(input())
y=int(input())
t1=Test(x,y)                                # object creation
print("Enter Two Value for Second Object:")
t2=Test(float(input()), float(input()))       # object creation
print("Enter Two Value for Third Object:")
t3=Test(input(), input() )                   # object creation


---


#TestEx4.py
class Test:
    def __init__(self,a=1,b=2): # Constructors--Object Initialization
        print("i am from Default / Parameterized Constructor")
        self.a=a
        self.b=b
        print("=*50)
        print("Val of a={}".format(self.a))
        print("Val of b={}".format(self.b))
        print("=*50)
#main program
t1=Test()                                     # object creation
t2=Test(100,200)                             # object creation
t3=Test(20,30)


---



```

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 role is to collect un-used memory space and it improves the performance 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 created for initializing 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)

=>Hence , We have THREE programming conditions for calling GC and to make the garbage collector to call destructor Functions.

- a) By default when the program execution completed.
- b) Make the object reference as None by Forcefully
Syntax : `objname=None`
- c) delete the object by using del by Forcefully
Syntax:- `del objname`

=>Syntax:

```
def __del__(self):
-----
-----
```

Garbage Collector

=>Garbage Collector contains a pre-defined module called "gc"
=>Here gc contains the following contains the following Functions.

- 1) isenabled()
 - 2) enable()
 - 3) disable()
-

```
#gcex1.py
import gc
print("is GC enabled =",gc.isenabled() ) # True
print("\nThis is a python class")
gc.disable()
print("is GC enabled after disable() =",gc.isenabled() ) # False
print("\nGoing on Destructor Concept:")
gc.enable()
print("is GC enabled after enable() =",gc.isenabled() ) # True
print("\nThis is OOPs")
```

#Program for demonstrating Functionality of Destructor.

```
#gcex2.py
import time,gc
class Employee:
    def __init__(self,eno,ename):
        self.eno=eno
        self.ename=ename
        print("\t{}\t{}".format(self.eno,self.ename))
    def __del__(self):
        print("\nDestructor called by Garbage Collector")
```

```
#mainm program
print("is GC enabled =",gc.isenabled() ) # True
print("Program execution started:")
gc.disable()
print("is GC enabled after disable() =",gc.isenabled() ) # False
eo1=Employee(10,"RS")
eo2=Employee(20,"DR")
eo3=Employee(30,"TR")
print("Program execution ended")
time.sleep(5)
```

#Program for demonstrating Functionality of Destructor.

```
#destex1.py
import time
```

```
class Employee:  
    def __init__(self,eno,ename):  
        self.eno=eno  
        self.ename=ename  
        print("\t{}\t{}".format(self.eno,self.ename))  
    def __del__(self):  
        print("\nDestructor called by Garbage Collector")  
#mainm program  
print("Program execution started:")  
eo1=Employee(10,"RS")  
eo2=Employee(20,"DR")  
eo3=Employee(30,"TR")  
print("Program execution ended")  
time.sleep(5)
```

#Program for demonstrating Functionality of Destructor.

```
#destex2.py  
import time  
class Employee:  
    def __init__(self,eno,ename):  
        self.eno=eno  
        self.ename=ename  
        print("\t{}\t{}".format(self.eno,self.ename))  
#mainm program  
print("Program execution started:")  
eo1=Employee(10,"RS")  
eo2=Employee(20,"DR")  
print("Program execution ended")  
time.sleep(5)
```

#Program for demonstrating Functionality of Destructor.

```
#destex3.py  
import time  
class Employee:  
    def __init__(self,eno,ename):  
        self.eno=eno  
        self.ename=ename  
        print("\t{}\t{}".format(self.eno,self.ename))  
    def __del__(self):  
        print("\nDestructor called by Garbage Collector")
```

```

#mainm program
print("Program execution started:")
eo1=Employee(10,"RS")
eo2=Employee(20,"DR")
eo3=Employee(30,"TR")
#no longer interested to maintain object eo1
print("no longer interested to maintain object eo1:")
eo1=None # Here the memory space eo1 collected by GC-forcefully
time.sleep(5)
#no longer interested to maintain object eo2
print("no longer interested to maintain object eo2:")
eo2=None # Here the memory space eo2 collected by GC -forcefully
time.sleep(5)
print("Program execution Execution Completed:")
time.sleep(5)
# Here the memory space eo3 collected by GC -automatically

```

#Program for demonstrating Functionality of Destructor.

```

#destex4.py
import time
class Employee:
    def __init__(self,eno,ename):
        self.eno=eno
        self.ename=ename
        print("\t{}\t{}".format(self.eno,self.ename))
    def __del__(self):
        print("\nDestructor called by Garbage Collector")

#main program
print("Program execution started:")
eo1=Employee(10,"RS")
eo2=Employee(20,"DR")
eo3=Employee(30,"TR")
eo4=Employee(40,"TR")
#no longer interested to maintain object eo1
print("no longer interested to maintain object eo1:")
del eo1 # Here the memory space eo1 collected by GC-forcefully
time.sleep(5)
#no longer interested to maintain object eo2
print("no longer interested to maintain object eo2:")
del eo2 # Here the memory space eo2 collected by GC -forcefully
time.sleep(5)

```

```
print("Program execution Execution Completed:")
time.sleep(5)
# Here the memory space eo3 and eo4 collected by GC -automatically
```

```
#Program for demonstrating Functionality of Destructor.
#destex5.py
import time
class Employee:
    def __init__(self,eno,ename):
        self.eno=eno
        self.ename=ename
        print("\t{}\t{}".format(self.eno,self.ename))
    def __del__(self):
        print("\nDestructor called by Garbage Collector")

#mainm program
print("Program execution started:")
eo1=Employee(10,"RS")
eo2=eo1 # Deep Copy
eo3=eo1 # Deep Copy # Here GC calls Desturctor Only once bcoz eo1,eo2,eo3 are
pointing
                                #single memory space.
```

#Program for demonstrating Functionality of Destructor.

```
#destex6.py
import time
class Employee:
    def __init__(self,eno,ename):
        self.eno=eno
        self.ename=ename
        print("\t{}\t{}".format(self.eno,self.ename))
    def __del__(self):
        print("\nDestructor called by Garbage Collector")

#mainm program
print("Program execution started:")
eo1=Employee(10,"RS")
eo2=eo1 # Deep Copy
eo3=eo1 # Deep Copy # Here GC calls Desturctor Only once bcoz eo1,eo2,eo3 are
pointing
                                #single memory space.
```

```
#no longer interested to maintain object eo1
print("no longer interested to maintain object eo1:")
eo1=None # Here the GC will not call Destructor forcefully bcoz still eo2 and eo3
```

```
pointing to same memory space .  
time.sleep(5)  
print("no longer interested to maintain object eo2:")  
eo2=None # Here the GC will not call Destructor forcefully bcoz still eo3 pointing to  
same memory space .  
time.sleep(5)  
print("Program execution ended :")  
time.sleep(5)  
# Here the GC will call Destructor automatically and eliminates same memory  
space of eo3 .
```

#Program for demonstrating Functionality of Destructor.

```
#destex7.py  
import time  
class Employee:  
    def __init__(self,eno,ename):  
        self.eno=eno  
        self.ename=ename  
        print("\t{}\t{}".format(self.eno,self.ename))  
    def __del__(self):  
        print("\nDestructor called by Garbage Collector")  
  
#mainm program  
print("Program execution started:")  
eo1=Employee(10,"RS")  
eo2=eo1 # Deep Copy  
eo3=eo1 # Deep Copy # Here GC calls Desturctor Only once bcoz eo1,eo2,eo3 are  
pointing  
                                #single memory space.  
  
#no longer interested to maintain object eo1  
print("no longer interested to maintain object eo3:")  
del eo3 # Here the GC will not call Destructor forcefully bcoz still eo1 and eo2  
pointing to same memory space .  
time.sleep(5)  
print("no longer interested to maintain object eo1:")
```

```
del eo1 # Here the GC will not call Destructor forcefully bcoz still eo2 pointing to
same memory space .
time.sleep(5)
print("Program execution ended :")
time.sleep(5)
# Here the GC will call Destructor automatically and eliminates same memory
space of eo2 .
```

#Program for demonstrating Functionality of Destructor.

```
#destex8.py
import time
class Employee:
    def __init__(self,eno,ename):
        self.eno=eno
        self.ename=ename
        print("\t{}\t{}".format(self.eno,self.ename))
    def __del__(self):
        print("\nDestructor called by Garbage Collector")

#mainm program
print("Program execution started:")
eo1=Employee(10,"RS")
eo2=eo1 # Deep Copy
eo3=eo1 # Deep Copy # Here GC calls Desturctor Only once bcoz eo1,eo2,eo3 are
pointing
               #single memory space.

#no longer interested to maintain object eo1
print("no longer interested to maintain object eo3:")
del eo3 # Here the GC will not call Destructor forcefully bcoz still eo1 and eo2
pointing to same memory space .
time.sleep(5)
print("no longer interested to maintain object eo1:")
del eo1 # Here the GC will not call Destructor forcefully bcoz still eo2 pointing to
same memory space .
time.sleep(5)
print("no longer interested to maintain object eo2:")
eo2=None # Here the GC will call Destructor forcefully
time.sleep(5)
print("Program execution ended :")
```

Data Encapsulation and Data Abstraction

Data Encapsulation:

=>Data Encapsulation is one of the distinct feature of OOPs

=>The Purpose of Data Encapsulation is that " To hide the confidential Information from External Programmers / Users".

=>The Process of Hiding / securing the confidential Information from external Programmers /

Un-Authorized Users is called Data Encapsulation.

=>The confidential Information represents Data Members and Methods.

=>In Python Programming, we can apply Data Encapsulation at 2 places. They are

- a) At Data Member Level.
- b) At Method Level.

=>To implement Data Encapsulation, in Python Programming, The Data Members and Methods Must Be preceded with __ (Eqv. Private)

=>Syntax1: (Data Encapsulation at Data Member Level)

```
__Data MemberName1= Value1  
__Data MemberName2= Value2  
-----  
__Data MemberName-n= Value-n
```

=>Syntax2: (Data Encapsulation at Method Level):

```
-----  
def __MethodName(self, list of formal params if any):  
-----  
    Block of statements  
-----
```

Data Abstraction

=>The Process of retrieving or extracting Essential details (Data Members or method) without considering Hidden Details is called Data Abstraction.

Example1:—Data Encapsulation at Data Member Level

```
#Others1.py  
from Account import Account  
ao=Account()  
print("-"*50)  
#print("Account Number={}".format(ao.acno)) Not Possible to access  
print("Account Holder Name={}".format(ao.cname))  
#print("Account Holder Bal={}".format(ao.bal)) Not Possible to access  
print("Account Holder Branch Name={}".format(ao.bname))  
#print("Account Holder Pin={}".format(ao.pin)) Not Possible to access  
print("-"*50)  
*****  
#Others1.py #-----Data Abstraction at Data Member Level  
from Account import Account  
ao=Account()  
print("-"*50)  
#print("Account Number={}".format(ao.acno)) Not Possible to access  
print("Account Holder Name={}".format(ao.cname))  
#print("Account Holder Bal={}".format(ao.bal)) Not Possible to access  
print("Account Holder Branch Name={}".format(ao.bname))  
#print("Account Holder Pin={}".format(ao.pin)) Not Possible to access  
print("-"*50)
```

Example2:—Data Encapsulation at Data Method Level

```
#SavingsAccount.py----File Name and Treated as Module Name  
class SAccount:  
    def __getSAccount(self): # Method is Encapsulated  
        self.acno=10  
        self.cname="Rossum"  
        self.bal=4.5  
        self.bname="SBI"  
        self.pin=4567  
    *****  
#Others2.py-----Data Abstraction at Data Method Level  
from SavingsAccount import SAccount  
ao=SAccount()  
#ao.getSAccount()----Not Possible to access bcoz method is hidden / capsulated  
print("-"*50)  
#print("Account Number={}".format(ao.acno))# Not Possible to access
```

```
#print("Account Holder Name={}".format(ao.cname))
#print("Account Holder Bal={}".format(ao.bal)) #Not Possible to access
#print("Account Holder Branch Name={}".format(ao.bname))
#print("Account Holder Pin={}".format(ao.pin)) #Not Possible to access
print("-"*50)
```

#Account.py-----File Name and Treated as Module Name
class Account:

```
    def __init__(self):
        self.__acno=10 # Encapsulated
        self.cname="Rossum"
        self.__bal=4.5           # Encapsulated
        self.bname="SBI"
        self.__pin=4567   # Encapsulated
```

#Others1.py

```
from Account import Account
ao=Account()
print("-"*50)
#print("Account Number={}".format(ao.acno)) Not Possible to access
print("Account Holder Name={}".format(ao.cname))
#print("Account Holder Bal={}".format(ao.bal)) Not Possible to access
print("Account Holder Branch Name={}".format(ao.bname))
#print("Account Holder Pin={}".format(ao.pin)) Not Possible to access
print("-"*50)
```

#SavingsAccount.py-----File Name and Treated as Module Name
class SAccount:

```
    def __getSAccount(self): # Method is Encapsulated
        self.acno=10
        self.cname="Rossum"
        self.bal=4.5
        self.bname="SBI"
        self.pin=4567
```

#Others2.py

```
from SavingsAccount import SAccount
ao=SAccount()
#ao.getSAccount()----Not Possible to access bcoz method is hidden / capsulated
print("-"*50)
#print("Account Number={}".format(ao.acno))# Not Possible to access
#print("Account Holder Name={}".format(ao.cname))
#print("Account Holder Bal={}".format(ao.bal)) #Not Possible to access
#print("Account Holder Branch Name={}".format(ao.bname))
#print("Account Holder Pin={}".format(ao.pin)) #Not Possible to access
print("-"*50)
```

#Sample.py

class Sample:

```
    def __init__(self):
        self.__a=10
        self.b=20
        print("Val of a in constructor={}".format(self.__a)) # # Valid to access
        print("Val of b in constructor={}".format(self.b))
```

```
def getvalues(self):
    print("Val of a in getvalues()={}".format(self.__a)) # Valid to access
    print("Val of b in getvalues()={}".format(self.b))

#main program
s=Sample()
s.getvalues()
#print(s.a,s.b) Invalid to access
```

Inheritance

Index:

- =>Purpose of Inheritance
 - =>Definition of Inheritance
 - =>Memory Management in Inheritance
 - =>Advantages of Inheritance
 - =>Types Of Inheritances
 - 1) Single Inheritance
 - 2) Multi Level Inheritance
 - 3) Hierarchical Inheritance
 - 4) Multiple Inheritance
 - 5) Hybrid Inheritance
 - =>Syntax for Inheritance.
 - =>Programming Examples
-

Inheritance

=> Inheritance is one of distinct features of OOPs

=>The purpose of Inheritance is that " To build Re-usable Applications in Python Object Oriented 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 ".

Advantages 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. Redundancy (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

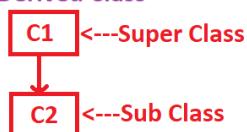
- 1) Single Inheritance
 - 2) Multi Level Inheritance
 - 3) Hierarchical Inheritance
 - 4) Multiple Inheritance
 - 5) Hybrid Inheritance
-

1) Single Inheritance

Definition:

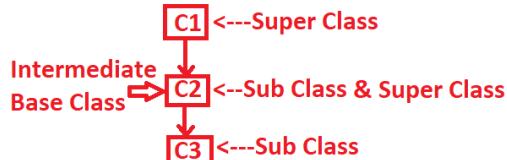
This Inheritance contains Single Base class and Single Derived Class

Diagram:



2. Multi Level Inheritance

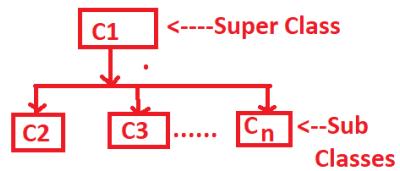
Definition: This Inheritance contains single base class , single derived class and Intermediate base class(es).



3. Hierarchical Inheritance

Definition: This Inheritance Contains Single Super Class and Multiple Sub Classes

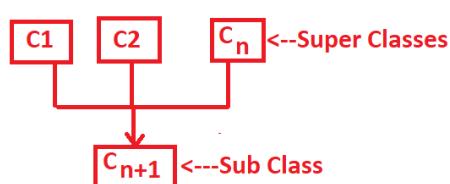
Diagram:



4. Multiple Inheritance

Definition: This Inheritance contains multiple Super Classes and single sub class.

Diagram:



5. Hybrid Inheritance:

Definition:
Hybrid Inheritance= Combination any available Inheritance Types.

Diagram1:

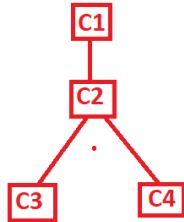
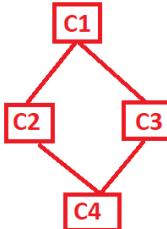


Diagram2:



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.

```
#Inhprog1.py  
class C1:  
    def fun1(self):  
        print("C1--fun1()")  
    def fun3(self):  
        print("C1-Fun3()")  
  
class C2(C1):  
    def fun2(self):  
        print("C2--fun2()")
```

```
#main program  
o2=C2()  
o2.fun2()  
o2.fun1()  
o2.fun3()
```

```
#Inhprog2.py  
class C1:  
    def fun1(self):
```

```

        print("C1-fun1()")
    def fun3(self):
        print("C1-Fun3()")
class C2(C1):
    def fun2(self):
        print("C2-fun2()")
        self.fun1() # here we are calling base class Methods from derived class
methods
        self.fun3() # here we are calling base class Methods from derived class
methods

#main program
o2=C2()
o2.fun2()


---


#University---->College---->Student
#Inhprog3.py
class Univ:
    def getunivdet(self):
        self.uname=input("Enter University Name:")
        self.uloc=input("Enter University Location:")
    def dispunivdet(self):
        print("-"*50)
        print("\tUniversity details:")
        print("-"*50)
        print("University Name:{}".format(self.uname))
        print("University Location:{}".format(self.uloc))
        print("-"*50)

class College(Univ):
    def getcolldet(self):
        self.cname=input("Enter College Name:")
        self.cloc=input("Enter College Location:")
    def dispcolldet(self):
        print("-"*50)
        print("\tCollege Details:")
        print("-"*50)
        print("College Name:{}".format(self.cname))
        print("College Location:{}".format(self.cloc))
        print("-"*50)

class Student(College):
    def getstuddet(self):
        self.sno=int(input("Enter Student Number:"))
        self.sname=input("Enter Student Name:")
        self.crs=input("Enter Student Course:")
    def dispstuddet(self):
        print("-"*50)
        print("\tStudent Details:")
        print("-"*50)
        print("Student Number:{}".format(self.sno))

```

```
print("Student Name:{}".format(self.sname))
print("Student Course:{}".format(self.crs))
print("-"*50)

#main program
s=Student()
s.getstuddet()
s.getcolldet()
s.getunivdet()
s.dispunivdet()
s.dispcolldet()
s.dispstuddet()
```

#Univ.py---File Name and acts as module
class Univ:

```
def getunivdet(self):
    self.uname=input("Enter University Name:")
    self.uloc=input("Enter University Location:")
def dispunivdet(self):
    print("-"*50)
    print("\tUniversity details:")
    print("-"*50)
    print("University Name:{}".format(self.uname))
    print("University Location:{}".format(self.uloc))
    print("-"*50)
```

#College.py---file name and acts as module

from Univ import Univ

class College(Univ):

```
def getcolldet(self):
    self.cname=input("Enter College Name:")
    self.cloc=input("Enter College Location:")
def dispcolldet(self):
    print("-"*50)
    print("\tCollege Details:")
    print("-"*50)
    print("College Name:{}".format(self.cname))
    print("College Location:{}".format(self.cloc))
    print("-"*50)
```

#Student.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.crs=input("Enter Student Course:")
    self.getcolldet()
    self.getunivdet()
```

```
def dispstuddet(self):
```

```
self.dispunivdet()
self.dispcolldet()
print("-"*50)
print("\tStudent Details:")
print("-"*50)
print("Student Number:{}".format(self.sno))
print("Student Name:{}".format(self.sname))
print("Student Course:{}".format(self.crs))
print("-"*50)


---


#main program
#studcolluniv.py
from Student import Student
s=Student()
s.getstuddet()
s.dispstuddet()
```

Polymorphism in Python

=>Polymorphism is one of the distinct features of OOPs
=>The purpose of Polymorphism is that "Efficient Utilization Memory Spacee (OR)
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)".

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()
```

**Number of approaches to call original methods from
Overridden methods**

=>We have two approaches 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 unable 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)

PROGRAMS:

```
#polyex1.py
class Circle(object):
    def draw(self): # Original Method
        print("Drawing Circle--Circle Class")
        #super().draw()-----invalid, bcoz object class does not
contain draw()
class Rect(Circle):
    def draw(self): # Overridden Method
        print("Drawing--Rect--Rect Class")
        super().draw() # will draw() of Circle Class from draw() of Rect
class
class Square(Rect):
    def draw(self): # Overridden Method
        print("Drawing--Square--Square Class")
        super().draw() # will draw() of Rect Class from draw() of Square
class

#main program
s=Square()
s.draw()
```

#WAP program which will calculate area of different figures by using classes and objects with polymorphism

```
#ployex2.py
class Circle:
    def area(self): # original method
        self.r=float(input("Enter Radious:"))
```

```

        ac=3.14*self.r**2
        print("Area of Circle={}".format(ac))

class Rect(Circle):
    def area(self): # overridden method
        self.l=float(input("Enter Length:"))
        self.b=float(input("Enter Breadth:"))
        ar=self.l*self.b
        print("Area of Rect={}".format(ar))
        super().area()

class Square(Rect):
    def area(self): # overridden method
        super().area()
        self.s=float(input("Enter Side:"))
        sa=self.s**2
        print("Area of Square={}".format(sa))

#main program
s=Square()
s.area()

```

#WAP program which will calculate area of different figures by using classes and objects with polymorphism

```

#polyex3.py
class Circle:
    def area(self): # original method
        self.r=float(input("Enter Radious:"))
        ac=3.14*self.r**2
        print("Area of Circle={}".format(ac))

class Rect:
    def area(self): # original method
        self.l=float(input("Enter Length:"))
        self.b=float(input("Enter Breadth:"))
        ar=self.l*self.b
        print("Area of Rect={}".format(ar))

class Square(Rect,Circle): # Multiple Inheritance
    def area(self): # overridden method
        print("-----")
        self.s=float(input("Enter Side:"))
        sa=self.s**2
        print("Area of Square={}".format(sa))
        print("-----")
        Circle.area(self) # calling area() of Circle from Square
        Rect.area(self) # calling area() of Rect from Square

#main program
s=Square()

```

```
s.area()
```

```
#WAP program which will calculate area of different figures by using classes and  
objects with polymorphism
```

```
#polyex4.py  
class Circle:  
    def area(self,r): # original method  
        ac=3.14*r**2  
        print("Area of Circle={}".format(ac))  
  
class Square:  
    def area(self,s): # original method  
        print("-----")  
        sa=s**2  
        print("Area of Square={}".format(sa))  
        print("-----")  
  
class Rect(Square,Circle): # multiple Inheritance  
    def area(self,l,b): # overridden method  
        ar=l*b  
        print("Area of Rect={}".format(ar))  
        super().area(12)  
        Circle.area(self,1.5)  
  
#main program  
l=float(input("Enter Length:"))  
b=float(input("Enter Breadth:"))  
r=Rect()  
r.area(l,b)
```

```
#WAP program which will calculate area of different figures by using classes and  
objects with polymorphism
```

```
#polyex5.py  
class Circle:  
    def area(self,r): # original method  
        ac=3.14*r**2  
        print("Area of Circle={}".format(ac))  
  
class Square:  
    def area(self,s): # original method  
        print("-----")  
        sa=s**2  
        print("Area of Square={}".format(sa))  
        print("-----")  
  
class Rect(Square,Circle): # multiple Inheritance  
    def area(self,l,b): # overridden method  
        ar=l*b  
        print("Area of Rect={}".format(ar))  
        super().area(float(input("Enter Side:")))
```

```
Circle.area(self,float(input("Enter Radious:")))
```

```
#main program
l=float(input("Enter Length:"))
b=float(input("Enter Breadth:"))
r=Rect()
r.area(l,b)
```

```
#polyex6.py-----with constructor
class C1:
```

```
    def __init__(self): # Original Constructor
        print("C1--default Constructor")
```

```
class C2(C1):
    def __init__(self): # Overridden Constructor
        print("C2--default--constructor")
        super().__init__()
```

```
class C3(C2):
    def __init__(self): # Overridden Constructor
        print("C3--default--constructor")
        super().__init__()
```

```
#main program
o3=C3()
```

```
#polyex7.py-----with constructor
class C1:
```

```
    def __init__(self): # Original Constructor
        print("C1--default Constructor")
```

```
class C2(C1):
    def __init__(self): # Overridden Constructor
        print("C2--default--constructor")
```

```
class C3(C2):
    def __init__(self): # Overridden Constructor
        print("C3--default--constructor")
        C1.__init__(self)
        C2.__init__(self)
```

```
#main program
o3=C3()
```

```
#polyex8.py-----with constructor
class C1:
```

```
    def __init__(self,a): # Original Constructor
        print("C1--default Constructor and a={}".format(a))
```

```
class C2:
    def __init__(self,b): # Original Constructor
```

```
        print("C2--default--constructor and b={}".format(b))

class C3(C2,C1):
    def __init__(self): # Overridden Constructor
        print("C3--default--constructor")
        C1.__init__(self,10)
        C2.__init__(self,20)

#main program
o3=C3()
```

```
#hybridinh.py
class C1:
    def x(self):
        print("C1--x()")
    def __init__(self):
        print("C1-- const")

class C2(C1):
    def x(self):
        print("C2--x()")
    def __init__(self):
        print("C2-- const")

class C3(C1):
    def x(self):
        print("C3--x()")
    def __init__(self):
        print("C3-- const")

class C4(C2,C3):
    def __init__(self):
        C1.__init__(self)
        C2.__init__(self)
        C3.__init__(self)

        print("C4-- const")
    def x(self):
        print("C4--x()")
        C1.x(self)
        C2.x(self)
        C3.x(self)

o4=C4()
print("-----")
o4.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 definition and continue the function execution until condition becomes false.

```
#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


---


```

Regular Expressions

Index

=>Purpose of Regular Expressions
=>Definition of Regular Expressions
=>Application of Regular Expression.
=>Module Name for Regular Expressions (re)
=>Pre-Defined Function of "re" Module
 a) finditer()
 b) start()
 c) stop
 d) group()
 e).findall()
 f) search()....etc

=>Programming Examples

=>Programmer-Defined Character Classes
=>Pre-Defined Character Classes
=>Quantifiers in Regular Expression.
=>Programming Examples

=>Programs in Regular Expressions

Regular Expressions

=>Regular Expressions concept is one Programming Language Independent Concept.
=>Here we are learning Regular Expressions with Python Syntaxes.
=>The purpose of Regular Expressions is that " To Evaluate or Validate the Data ".

=>Definition of Regular Expression:

=>A Regular Expression is one of the String Pattern(Combination of Alphabets, Digits and Special Symbols) which is used to search or find or match with the given data and obtains desired Result .

Applications of Regular Expressions:

=>Regular Expressions are used in Development of Editors / IDEs
=>Regular Expressions used in Operating System development
=>Regular Expressions used in Micro Controller Application
=>Regular Expressions used in Electronic Circuits
=>Regular Expressions used in Protocols development.
=>Regular Expressions used in Language Compilers / Interpreters and Execution Environments
=>Regular Expressions used in Pattern Matching Service....etc

```
gd="Python is an OOP lang and Python is also Functional Programming Lang."  
sp="Python"  
mat=re.finditer(sp, gd) # mat is variable of type <class,'callable_iterator'>  
print(mat) # we get Hexa  
Dec Addr      Start Index    End Index    Value  
for m in mat:  
    print("S Index={}.format(m.start())")  
    print("E. Index={}".format(m.end()))  
    print("Value={}".format(m.group()))
```

Dec Addr	Start Index	End Index	Value
	0	6 (5+1)	Python
	26	32(31+1)	Python

```
=====
```

Pre-defined Functions in re module

=>To deal with deal the programming in Regular Expressions, we must use a pre-defined module called "re".
=>The 're' module contains the following 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()

=>This function present in match class of re module

Syntax:- varname=matchtabobj.group()

3) start():

=>This function is used obtaining starting index of matched value

=>This function present in match class of re module

Syntax: varname=matchobj.start()

4) end():

=>This function is used obtaining end index+1 of matched value

=>This function present in match class of re module

Syntax: varname=matchobj.end()

5) search():

Syntax:- varname=re.search("search-pattern","Given data")

=>here varname is an object of <class,'re.match'>

=>This function is used for searching the search pattern in given data for first occurrence / match only but not for other occurrences.

=>if the search pattern found in given data then it returns an object of match class 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 which is type <class, "NoneType"> 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 occurrences / matches and it returns all the matched values in the form an object <class,'list'> but not returning Start and End Index Values.

Programs:

```
#RegExpr1.py
```

#Q) find the word "Python" and no.of times repeated ---findall()

```
import re
```

```
givendata="Python is an OOP lang and Python is also Functional Programming  
Lang."
```

```
sp="Python"
```

```
mat=re.findall(sp,givendata) # mat is variable of type <class,'list'>
```

```
print("No. of occurences of '{}={}'.format(sp,len(mat)))
```

```

#RegExpr2.py
#Q) find the word "Python" and no.of times repeated --finditer()
import re
givendata="Python is an OOP lang and Python is also Functional Programming
Lang."
sp="Python"
mat=re.finditer(sp,givendata)
print("type of mat=",type(mat)) # type of mat= <class 'callable_iterator'>
print("-----")
print("Matching Results:")
print("-----")
for m in mat: # here 'm' is an object of <class 're.Match'>
    print("\nStart Index={} End Index={}\nValue={}".format(m.start(),m.end(),m.group()))
    print("-----")

```

E:\KVR-PYTHON-9AM\REGULAR EXPRESSIONS>py RegExpr2.py
type of mat= <class 'callable_iterator'>

Matching Results:

Start Index=0 End Index=6 Value=Python

Start Index=26 End Index=32 Value=Python

```

#RegExpr3.py
#Q) find the word "Python" and no.of times repeated --finditer()
import re
givendata="Python is an OOP lang and Python is also Functional Programming
Lang."
sp="python".title()
noc=0
mat=re.finditer(sp,givendata) # mat is variable of type <class,'list'>
print("-----")
print("Matching Results:")
print("-----")
for m in mat: # here 'm' is an object of <class 're.Match'>
    print("\nStart Index={} End Index={}\nValue={}".format(m.start(),m.end(),m.group()))
    noc=noc+1

```

```
print("-----")
print("Number of Occurences of '{}'={}".format(sp,noc))
print("-----")


---


#RegExpr4.py
#Q) find the word "Python" and no.of times repeated --finditer()
import re
givendata="Java is an OOP lang and Python is also Functional Programming Lang
and Python is used in AI."
sp="lang1"
mat=re.search(sp,givendata)
print("type of mat=",type(mat)) # here mat is an object of <class 're.Match'> if match
occurs other it returns <class,"NoneType">
if(mat!=None):
    print("\nSearch is Sucessful")
    print("Start Index={} and End Index={} and
Value={}".format(mat.start(),mat.end(),mat.group()))
else:
    print("\nSearch is Un-Sucessful")
    print("{} does not exists in given data".format(sp))


---


```

Programmer-Defined character classes in Regular Expression

=> Programmer-Defined character classes defined or prepared by Python Programmers for building Search Pattern and it is used to search or find or match with given data and obtains desired result.
=> Programmer-Defined character classes are given bellow.
=> Syntax for Programmer-Defined character classes.

[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
 7. [0-9]----->searches for all digits only
 8. [^0-9]----->searches for all except digits.
 9. [A-Za-z]----->searches for all lower and upper case alphabets only
 10. [^A-Za-z]----->searches for all except lower and upper case alphabets.
 11. [A-Za-z0-9]----->searches for all lower and upper case alphabets and digits
 12. [^A-Za-z0-9]----->searches for all special symbols
(except lower and upper case alphabets and digits)
-

```
#RegExpr5.py
#This program searches for either 'a' or 'b' or 'c' only
import re
mat=re.finditer("[abc]" , "cWaBT#6AT&3pk@1LH8bJ")
print("=*50)
```

```
for m in mat:  
    print("start index:{} End Index:{} Value:{}".format(m.start(),m.end(),m.group()))  
print("*50)
```

=====
E:\KVR-PYTHON-9AM\REGULAR EXPRESSIONS>py RegExpr5.py
=====

```
start index:0 End Index:1 Value:c  
start index:2 End Index:3 Value:a  
start index:18 End Index:19 Value:b  
=====
```

```
#RegExpr6.py  
#This program searches for all except either 'a' or 'b' or 'c'  
import re  
mat=re.finditer("[^abc]", "cWaBT#6AT&3pk@1LH8bJ")  
print("*50)  
for m in mat:  
    print("start index:{} End Index:{} Value:{}".format(m.start(),m.end(),m.group()))  
print("*50)
```

=====
E:\KVR-PYTHON-9AM\REGULAR EXPRESSIONS>py RegExpr6.py
=====

```
start index:1 End Index:2 Value:W  
start index:3 End Index:4 Value:B  
start index:4 End Index:5 Value:T  
start index:5 End Index:6 Value:#  
start index:6 End Index:7 Value:6  
start index:7 End Index:8 Value:A  
start index:8 End Index:9 Value:T  
start index:9 End Index:10 Value:&  
start index:10 End Index:11 Value:3  
start index:11 End Index:12 Value:p  
start index:12 End Index:13 Value:k  
start index:13 End Index:14 Value:@  
start index:14 End Index:15 Value:1  
start index:15 End Index:16 Value:L  
start index:16 End Index:17 Value:H  
start index:17 End Index:18 Value:8  
start index:19 End Index:20 Value:J  
=====
```

```
#RegExpr7.py  
#This program searches for all lower alphabets  
import re
```

```
mat=re.finditer("[a-z]" , "cWaBT#6AT&3pk@1LH8bJ")
print("=*50)
for m in mat:
    print("start index:{} End Index:{} Value:{}".format(m.start(),m.end(),m.group()))
print("=*50)
```

```
#RegExpr8.py
#This program searches for all except lower case alphabets
import re
mat=re.finditer("[^a-z]" , "cWaBT#6AT&3pk@1LH8bJ")
print("=*50)
for m in mat:
    print("start index:{} End Index:{} Value:{}".format(m.start(),m.end(),m.group()))
print("=*50)
```

```
#RegExpr9.py
#This program searches for all Upper case alphabets
import re
mat=re.finditer("[A-Z]" , "cWaBT#6AT&3pk@1LH8bJ")
print("=*50)
for m in mat:
    print("start index:{} End Index:{} Value:{}".format(m.start(),m.end(),m.group()))
print("=*50)
```

```
#RegExpr10.py
#This program searches for all except Upper case alphabets
import re
mat=re.finditer("[^A-Z]" , "cWaBT#6AT&3pk@1LH8bJ")
print("=*50)
for m in mat:
    print("start index:{} End Index:{} Value:{}".format(m.start(),m.end(),m.group()))
print("=*50)
```

```
#RegExpr11.py
#This program searches for all digits
import re
mat=re.finditer("[0-9]" , "cWaBT#6AT&3pk@1LH8bJ")
print("=*50)
for m in mat:
    print("start index:{} End Index:{} Value:{}".format(m.start(),m.end(),m.group()))
print("=*50)
```

```
#RegExpr12.py
#This program searches for all except digits
import re
mat=re.finditer("[^0-9]" , "cWaBT#6AT&3pk@1LH8bJ")
```

```
print("=*50)
for m in mat:
    print("start index:{} End Index:{} Value:{}".format(m.start(),m.end(),m.group()))
print("=*50)
```

```
#RegExpr13.py
#This program searches for all lower and upper case alphabets
import re
mat=re.finditer("[A-Za-z]" , "cWaBT#6AT&3pk@1LH8bJ")
print("=*50)
for m in mat:
    print("start index:{} End Index:{} Value:{}".format(m.start(),m.end(),m.group()))
print("=*50)
```

```
#RegExpr14.py
#This program searches for all except lower and upper case alphabets
import re
mat=re.finditer("[^A-Za-z]" , "cWaBT#6AT&3pk@1LH8bJ")
print("=*50)
for m in mat:
    print("start index:{} End Index:{} Value:{}".format(m.start(),m.end(),m.group()))
print("=*50)
```

```
#RegExpr15.py
#This program searches for all lower and upper case alphabets and digits only (except
special symbols)
import re
mat=re.finditer("[A-Za-z0-9]" , "cWaBT#6AT&3pk@1LH8bJ")
print("=*50)
for m in mat:
    print("start index:{} End Index:{} Value:{}".format(m.start(),m.end(),m.group()))
print("=*50)
```

```
#RegExpr16.py
#This program searches for all special symbols ( except alphabets and digits)
import re
mat=re.finditer("[^A-Za-z0-9]" , "$cWa BT#6A木T&3pk@1LH8bJ")
print("=*50)
for m in mat:
    print("start index:{} End Index:{} Value:{}".format(m.start(),m.end(),m.group()))
print("=*50)
```

Quantifiers in Regular Expressions

=>Quantifiers in Regular Expressions are used for searching number of occurrences 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----->Searches exactly one k
 - 2) k+----->searches for one k or more k's
 - 3) k* ----->Searches zero k or one k or more k's
 - 4) k?----->searches zero k or one k
 - 5) . -----> searches for all
-

```
#RegExpr23.py
#This program Searches exactly one k
import re
mat=re.finditer("k" , "kvkkvkkkvkv")
print("*50)
for m in mat:
    print("start index:{} End Index:{} Value:{}".format(m.start(),m.end(),m.group()))
print("*50)

=====
E:\KVR-PYTHON-9AM\REGULAR EXPRESSIONS>py RegExpr23.py
=====
start index:0 End Index:1 Value:k
start index:2 End Index:3 Value:k
start index:3 End Index:4 Value:k
start index:5 End Index:6 Value:k
start index:6 End Index:7 Value:k
start index:7 End Index:8 Value:k
```

```
start index:9 End Index:10 Value:k
=====
=====
```

```
#RegExpr24.py
#This program k+ means searching one k or more k's
import re
mat=re.finditer("k+", "vkkvkkkvkv")
print("*50)
for m in mat:
    print("start index:{} End Index:{} Value:{}".format(m.start(),m.end(),m.group()))
print("*50)
```

```
====
```

```
E:\KVR-PYTHON-9AM\REGULAR EXPRESSIONS>py RegExpr24.py
=====
```

```
start index:0 End Index:1 Value:k
start index:2 End Index:4 Value:kk
start index:5 End Index:8 Value:kkk
start index:9 End Index:10 Value:k
=====
```

```
#RegExpr25.py
#This program k* means searching zero k or one k or more k's
import re
mat=re.finditer("k*", "vkkvkkkvkv")
print("*50)
for m in mat:
    print("start index:{} End Index:{} Value:{}".format(m.start(),m.end(),m.group()))
print("*50)
```

```
====
```

```
E:\KVR-PYTHON-9AM\REGULAR EXPRESSIONS>py RegExpr25.py
=====
```

```
start index:0 End Index:1 Value:k
start index:1 End Index:1 Value:
start index:2 End Index:4 Value:kk
start index:4 End Index:4 Value:
start index:5 End Index:8 Value:kkk
start index:8 End Index:8 Value:
start index:9 End Index:10 Value:k
start index:10 End Index:10 Value:
start index:11 End Index:11 Value:
=====
```

```
#RegExpr25.py
#This program k? means searching zero k or one k only.
import re
mat=re.finditer("k?", "vkkvkkkvkv")
print("*50)
for m in mat:
    print("start index:{} End Index:{} Value:{}".format(m.start(),m.end(),m.group()))
print("*50)
```

```
"""
E:\KVR-PYTHON-9AM\REGULAR EXPRESSIONS>py RegExpr26.py
=====
start index:0 End Index:1 Value:k
start index:1 End Index:1 Value:
start index:2 End Index:3 Value:k
start index:3 End Index:4 Value:k
start index:4 End Index:4 Value:
start index:5 End Index:6 Value:k
start index:6 End Index:7 Value:k
start index:7 End Index:8 Value:k
start index:8 End Index:8 Value:
start index:9 End Index:10 Value:k
start index:10 End Index:10 Value:
start index:11 End Index:11 Value:
====="""


---


```

```
#RegExpr27.py
#This program for searches all ("." means searching all)
import re
mat=re.finditer(".", "kvkkvkkkvkv")
print("*50")
for m in mat:
    print("start index:{} End Index:{} Value:{}".format(m.start(),m.end(),m.group()))
print("*50")

"""

```

```
E:\KVR-PYTHON-9AM\REGULAR EXPRESSIONS>py RegExpr27.py
=====
start index:0 End Index:1 Value:k
start index:1 End Index:2 Value:v
start index:2 End Index:3 Value:k
start index:3 End Index:4 Value:k
start index:4 End Index:5 Value:v
start index:5 End Index:6 Value:k
start index:6 End Index:7 Value:k
start index:7 End Index:8 Value:k
start index:8 End Index:9 Value:v
start index:9 End Index:10 Value:k
start index:10 End Index:11 Value:v
====="""


---


```

Pre-Defined character classes in Regular Expression

=> Pre-Defined character classes defined already in Python Software and used Python programmers for building Search Patterns and it is used to search or find or match with given data and obtains desired result.

=> Syntax for Pre-Defined character classes.

\Pre-Defined Character Class

=> Pre-Defined character classes are given below.

- 1) \s----->Searches for space character only
- 2) \S----->Searches for all except space character
- 3) \w----->searches for a word character (Or) [A-Za-z0-9]
- 4) \W----->searches for special symbols (except word character) OR [^A-Za-z0-9]
- 5) \d----->searches for digits only or [0-9]
- 6) \D----->Searches for all except digits or [^0-9]

Note:

\d{2} ----->Searches two digit no.

\d{1,5}----->Searches for all one to five digit number

```
#RegExpr17.py
#This program searches for space character
import re
mat=re.finditer("\s" , "$cWa BT#6AT&3pk @1LH8bJ")
print("=*50)
for m in mat:
    print("start index:{} End Index:{} Value:{}".format(m.start(),m.end(),m.group()))
```

```
print("=*50)

"""

E:\KVR-PYTHON-9AM\REGULAR EXPRESSIONS>py RegExpr17.py
=====
start index:4  End Index:5 Value:      (space)
start index:15  End Index:16 Value:      (space)
====="""

#RegExpr18.py
#This program searches for all except space character
import re
mat=re.finditer("\S", "$cWa BT#6AT&3pk @1LH8bJ")
print("=*50)
for m in mat:
    print("start index:{}  End Index:{}  Value:{}".format(m.start(),m.end(),m.group()))
print("=*50)
```

```
#RegExpr19.py
#This program searches for all word character
import re
mat=re.finditer("\w", "$cWa BT#6AT-&3pk @1LH8bJ")
print("=*50)
for m in mat:
    print("start index:{}  End Index:{}  Value:{}".format(m.start(),m.end(),m.group()))
print("=*50)
```

```
#RegExpr20.py
#This program searches for all sepecial symbols except word character
import re
mat=re.finditer("\W", "$cWa BT#6AT-&3pk @1LH8bJ")
print("=*50)
for m in mat:
    print("start index:{}  End Index:{}  Value:{}".format(m.start(),m.end(),m.group()))
print("=*50)
```

```
#RegExpr21.py
#This program searching for all digits
import re
mat=re.finditer("\d", "$cWa BT#6AT-&3pk @1LH8bJ")
print("=*50)
for m in mat:
    print("start index:{}  End Index:{}  Value:{}".format(m.start(),m.end(),m.group()))
print("=*50)
```

```
#RegExpr22.py
#This program searching for all except digits
import re
mat=re.finditer("\D", "$cWa BT#6AT-&3pk @1LH8bJ")
print("=*50)
for m in mat:
    print("start index:{}  End Index:{}  Value:{}".format(m.start(),m.end(),m.group()))
print("=*50)
```

```
#reguex23.py
#\d{2}----->Searches for two digit number
import re
mat=re.finditer("\d{2}","poo16&ritu19w22")
print("*"*50)
for m in mat:
    print("start:{} end:{} value:{}".format(m.start(),m.end(),m.group()))
print("*"*50)
```


start:3 end:5 value:16
start:10 end:12 value:19
start:13 end:15 value:22


```
#reguex24.py
#\d{1,5}----->Searches for all one to five digit number
import re
mat=re.finditer("\d{1,5}","6poo16&6ritu19w2022")
print("*"*50)
for m in mat:
    print("start:{} end:{} value:{}".format(m.start(),m.end(),m.group()))
print("*"*50)
```


start:0 end:1 value:6
start:4 end:6 value:16
start:7 end:8 value:6
start:12 end:14 value:19
start:15 end:19 value:2022

extract names of the students which are present in the given text

```
#namesex1.py
import re
studdata="Pranav is student , Ankit is CEO , Santhosh is software eng, Ajay is reasearch
student, Rossum is a developer and Mallesh is a teacher and Kuruva is data eng and
Manikanta is scientist and Umesh is a very good student."
nameslist=re.finditer("[A-Z][a-z]+ ", studdata)
print("-----")
print("Names of Students:")
print("-----")
```

```
for na in nameslist:  
    print(na.group())  
print("-----")
```

```
#WAP which will extract names of the students which are present in the given text  
#namesex2.py  
import re  
studdata="Pranav is student , Ankit is CEO , Santhosh is software eng, Ajay is reasearch  
student, Rossum is a developer and Mallesh is a teacher and Kuruva is data eng and  
Manikanta is scientist and Umesh is a very good student."  
nameslist=re.findall("[A-Z][a-z]+", studdata)  
print("-----")  
print("Names of Students in Ascending Order:")  
print("-----")  
nameslist.sort()  
for name in nameslist:  
    print(name)  
print("-----")
```

```
#WAP which will extract names of the students and their marks which are present in the  
given text and  
#namesmarksex1.py  
import re  
studdata="Pranav is student got marks 60 , Ankit is CEO marks 99 , Santhosh is software  
eng and whose marks 65 , Ajay is reasearch student and marks 67 , Rossum is a developer  
got marks 35 and Mallesh is a teacher got 60 and Kuruva is data eng and got 77 and  
Manikanta is scientist got 88 and Umesh is a very good student got 79 ."  
nameslist=re.finditer("[A-Z][a-z]+",studdata) # Reg Expr for Names  
print("-----")  
print("Names of students:")  
print("-----")  
for name in nameslist:  
    print("\t{}".format(name.group()))  
print("-----")  
markslist=re.finditer("\d{2} ",studdata)  
print("Marks of students:")  
print("-----")  
for marks in markslist:  
    print("\t{}".format(marks.group()))  
print("-----")
```

```
#WAP which will extract names of the students and their marks which are present in the  
given text and  
#namesmarksex2.py  
import re  
studdata="Pranav is student got marks 60 , Ankit is CEO marks 99 , Santhosh is software  
eng and whose marks 65 , Ajay is reasearch student and marks 67 , Rossum is a developer  
got marks 35 and Mallesh is a teacher got 60 and Kuruva is data eng and got 77 and  
Manikanta is scientist got 88 and Umesh is a very good student got 79 ."  
nameslist=re.findall("[A-Z][a-z]+",studdata) # Reg Expr for Names  
markslist=re.findall("\d{2} ",studdata)  
print("-----")
```

```
print("Name of Student\t\tMarks of students:")
print("-----")
for names,marks in zip(nameslist,markslist):
    print("\t{}\t\t{}".format(names,marks))
print("-----")
```

```
#WAP which will read the text data from file which contains multiple student names and the
marks and extract names and marks from thr given file
#namesmarksex3.py
import re
try:
    with open("D:\\ampt\\studinfo.data","r") as fp:
        filedata=fp.read()
        nameslist=re.findall("[A-Z][a-z]+",filedata) # Reg Expr for Names
        markslist=re.findall("\d{2} ",filedata) # # Reg Expr for Marks
        print("-----")
        print("Name of Student\t\tMarks of students:")
        print("-----")
        for names,marks in zip(nameslist,markslist):
            print("\t{}\t\t{}".format(names,marks))
        print("-----")
except FileNotFoundError:
    print("File does not exists")
```

```
#emailswithnames.py
import re
text="Rossum mail id is rossum_12_psf@psf.com , Ritche mail id is ritche_c@belllabs.co.in ,
Gosling mail id is gosling_1_java@sun.net.org and Travis email id is
travis_numpy@numpy.org and Kvr mail id is kvr1.java@gmail.com . "
nameslist=re.findall("[A-Z][a-z]+",text) # Reg Expr for Names
mailslist=re.findall("\S+@\S+",text) # Reg Expr for mail-id extraction
print("-----")
print("Name of Student\t\tMails of students:")
print("-----")
for names-mails in zip(nameslist,mailslist):
    print("\t{}\t\t{}".format(names,mails))
print("-----")
```

```
#write a python program which will extract email id's from the given text
#emailveri.py
import re
text="Rossum mail id is rossum_12_psf@psf.com , ritche mail id is ritche_c@belllabs.co.in ,
Gosling mail id is gosling_1_java@sun.net.org and Travis email id is
travis_numpy@numpy.org and kvr mail id is kvr1.java@gmail.com . "
emails=re.findall("\S+@\S+", text)
for mail in emails:
    print("\t{}".format(mail))
```

```

#mobilenumbervalidation.py
import re
while(True):
    mno=input("Enter ur mobile number:")#12345678ab
    if(len(mno)==10):
        result=re.search("\d{10}",mno)
        if(result!=None):
            print("Ur Mobile Number is Valid")
            break
        else:
            print("Mobile Number should not contain Alphabets / special
symbols")
    else:
        print("Mobile Number must contain 10 digits:")

```

Multi Threading in Python

Index

- =>Purpose of Multi Threading
 - =>Types of Applications
 - =>Definition of Thread
 - =>Purpose of Thread
 - =>Module Name Required For Multi Threading
 - > threading <----Module Name
 - * Variables Names
 - * Function Names
 - * Thread class
 - * Lock Class
-

Programming Examples

- =>Synchronization concept in Python
 - =>Dead Locks in Python
 - =>Avoiding Dead Locks
 - =>Programming Examples
-

Multi Threading in Python

- =>The purpose of Multi Threading is " To provide Concurrent Execution "
- =>The concurrent execution is nothing but Simultaneous or Parallel Execution.
- =>The Advantage of Concurrent Execution is that always takes Less Execution Time.
- =>In the context of Programming Languages, we have 2 types of Languages. They are
 - a) Process Based Applications
 - b) Thread Based Applications.

a) Process Based Applications:

- =>Process Based Applications execution environment contains Single Thread

=>Process Based Applications provides Sequential Execution
=>Process Based Applications Takes More Execution Time
=>Process Based Applications are treated as Heavy Weight Application.
Examples: C,CPP...etc

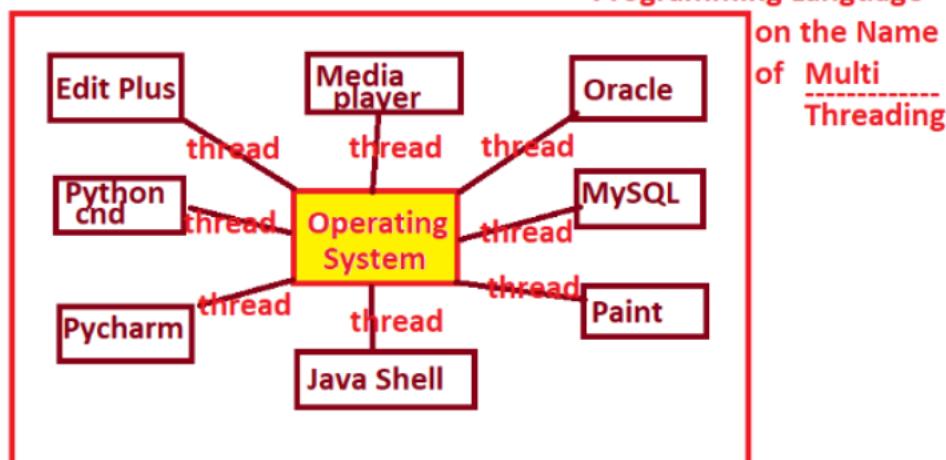
b) Thread Based Applications.

=>Thread Based Applications are those whose execution environment contains Single Thread (by default) and allows us create Multiple Threads programmatically.
=>Thread Based Applications provides Sequential Execution (by default) also provides Concurrent Execution programmatically.
=>Thread Based Applications Takes Less Execution Time
=>Thread Based Applications are treated as Light Weight Applications.
Examples: Python, Java . C#.Net...etc

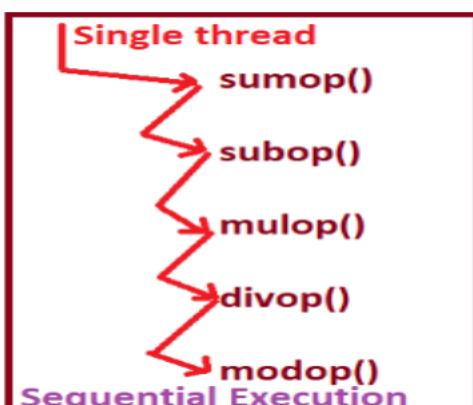
Operating System contains concept Multi Tasking---carried in

Programming Language

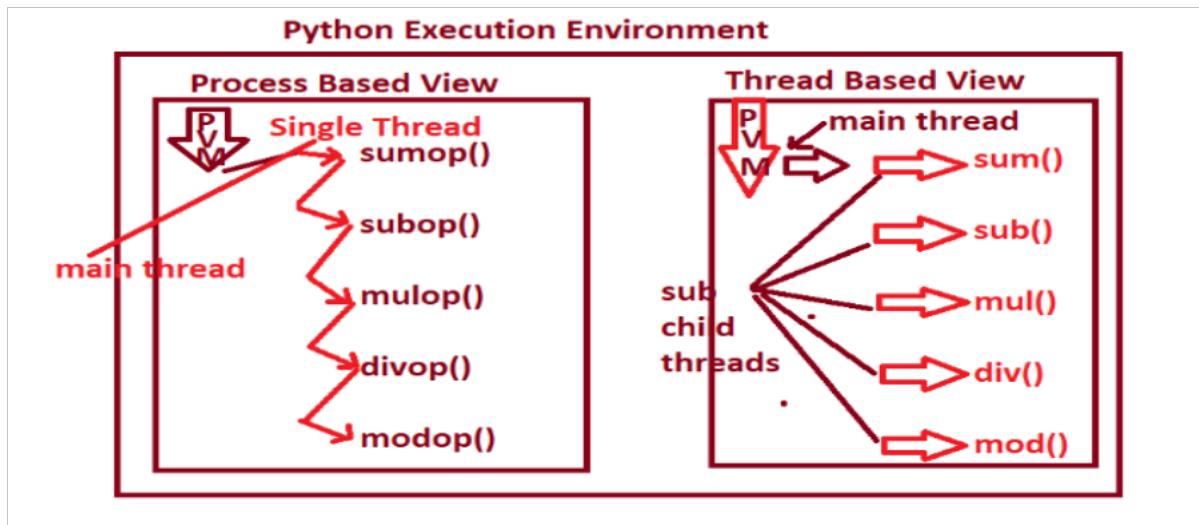
on the Name
of Multi
Threading



C,CPP--no lib
Process Based Application



Not at all possible Concurrent Execution



#ThreadDemo1.py

```
import threading
print("====")
print("default Name of thread=", threading.current_thread().name) # MainThread
print("Number Threads are active=", threading.active_count()) # 1
print("====")
print("This is my python program")
print("This is Multi Threading")
```

#ThreadDemo2.py

```
import threading
def fun1():
    print("i am from fun1()--executed by :",threading.current_thread().name)
def fun2():
    print("i am from fun2()--executed by :",threading.current_thread().name)
def fun3():
    print("i am from fun3()--executed by :",threading.current_thread().name)
def fun4():
    print("i am from fun4()--executed by :",threading.current_thread().name)

#main program
print("Main Program Execution Started:")
tname=threading.current_thread().name
print("Name of main thread in main program=",tname)
print("-"*50)
fun1()
print("main thread came to main program going to fun2()")
fun2()
print("main thread came to main program going to fun3()")
fun3()
print("main thread came to main program going to fun4()")
fun4()
print("-"*50)
print("Main Program Execution stoped:")
```

```
#nonthreadsprogram.py
import threading,time
def squares(lst):
    print("Name of thread executing squares()=",threading.current_thread().name)
    for val in lst:
        print("square({})={}".format(val,val**2))
        time.sleep(1)
def cubes(lst):
    print("Name of thread executing cubes()=",threading.current_thread().name)
    for val in lst:
        print("cubes({})={}".format(val,val**3))
        time.sleep(1)
#main program
bt=time.time()
print("Main Program Execution Started:")
tname=threading.current_thread().name
print("Name of main thread in main program=",tname)

print("-"*50)
lst=[2,5,6,3,8,9]
print("main thread came to main program going to squares()")
squares(lst)
print("main thread came to main program going to cubes()")
cubes(lst)
print("Main Program Execution Stopped:")
#find execution of this non-threading program
et=time.time()
print("\Execution time={}".format(et-bt))
```

```
#withthreadsprogram.py
import threading,time
def squares(lst):
    print("Name of thread executing squares()=",threading.current_thread().name) #
Thread-1
    for val in lst:
        print("square({})={}".format(val,val**2))
        time.sleep(1)

def cubes(lst):
    print("Name of thread executing cubes()=",threading.current_thread().name) #
Thread-2
    for val in lst:
        print("cubes({})={}".format(val,val**3))
        time.sleep(1)

#main program
bt=time.time()
print("Main Program Execution Started:")
tname=threading.current_thread().name
print("Name of main thread in main program=",tname) # MainThread
print("-"*50)
```

```
lst=[2,5,6,3,8,9]
t1=threading.Thread(target=squares,args=(lst,) ) # here t1 is an object of Thread
                                                class--Therad-1
t2=threading.Thread(target=cubes,args=(lst,) ) # here t2 is an object of Thread
t1.start()
t2.start()
t1.join()
t2.join()
print("Main Program Execution Completed:")
#find execution of this non-threading program
et=time.time()
print("\Execution time={}".format(et-bt))
```

Introduction to Thread Based Applications

- =>The purpose of multi threading is that "To provide Concurrent / Simultaneous execution / Parallel Execution".
- =>Concurrent Execution is nothing but executing the operations all at once.
- =>The advantage of Concurrent execution is that to get less execution time.
- =>If a Python Program contains multiple threads then it is called Multi Threading program.

=>**Def. of thread:**

- =>A flow of Control is called thread.
- =>The purpose of thread is that "To Perform certain operation whose logic developed in Functions / Methods concurrently."

- =>By default Every Python contains Single Thread and whose name is "MainThread" and It provides Sequential Execution.
- =>Programmatically, In a Python Program we can create multiple sub / Child threads and whose purpose is that "To execute operations whose logic is written in Functions / Methods Concurrently ".
- =>Hence Programmatically a Python Program two types of Threads. They are
 - a) MainThread
 - b) Sub / Child Threads
- =>MainThread is created / Initiated by PVM when program execution starts and the role of mainThread is to execute main program statements and Monitor the execution status of Sub threads.
- =>The Sub / Child Threads always executes operations whose logic is written in Functions / Methods Concurrently ".

Module Required for Developing Thread Based Applications

- =>The module name Required for Developing Thread Based Applications is "threading".

=>Before developing thread based applications, we must import "threading" module.

=>We know that a module is a collection of Variable Names, Function Names and Class

Names.

=>Here "threading" module contains the following Variable Names, Function Names and class names.

Module Name: threading

Function Names:

- 1) `current_thread()`---Syntax: `threading.current_thread()`
 - 2) `active_count()`----syntax: `threading.active_count()`

Class Name-1: Thread---use of Thread class is to create sub thread(s)

1) Thread(target,args) → t1=threading.Thread(target=funcname,args=())

2) start()—used for dispatching / sending sub threads to targeted Function
Syntax:- `subthreadobj.start()`

Syntax:- `subthread.setName("User-FriendlyName")`
(OR)
`subthread.name=User-FriendlyName`

4) getName()--- getting the thread name-----depreciated on "name"

Syntax:- `subthread.getName()`
(OR)
`subthread.name`

5) `is_alive()`--Returns True provided thread is running otherwise False

Syntax:- `subthreadobj.is_alive()`

6) join()---It is used for making the sub threads to join with main thread after complete execution of sub threads.

Syntax:- subthread.join()

7) run(self):- It is one of the null body method of Thread class of threading module
It is used defining logic of Python program which is executed by Sub thread.

This method is automatically called by start().

Example:

Example: class classname(threading.Thread):

```
    def run(self):
```

Block of stmts--executed by sub threads

```
#main program  
obj=classname()
```

```
obj.start() # here start() is internally calling run() of <classname>
```

Class Name-2: Lock

- 1) acquire()
 - 2) release()
-

Number of approaches to develop thread based applications

=>In Python, we can develop thread based applications in 3 approaches. They are

- 1) By Using Functional Approach
 - 2) By Using Sub Class of Thread class of threading module (with OOPs and Inheritance)
 - 3) By Using Non-Sub Class of Thread class of threading module (with OOPs without Inheritance)
-

1) By Using Functional Approach

Step-1: import threading module

Step-2: Define a Function with Logic which is executed by Sub Thread

Step-3: create sub thread

Step-4: dispatch or send the sub thread to the target function.

Examples:

```
#FirstApproachEx1.py
import threading # step-1
def hello(s): # step-2
    print("-"*50)
    print("\nThis Function executed by Sub Thread and Whose Name
is:{}".format(threading.current_thread().name))
    print("Hi,{} wel come to threading".format(s))
    print("-"*50)
```

```
#main program
t=threading.Thread(target=hello,args=("pooja-kavya",)) # step-3
t.start() # step-4
```

2) By Using Sub Class of Thread class of threading module (with OOPs and Inheritance)

Step-1: import threading

Step-2: Choose the Programmer-Defined Class

Step-3: The Programmer-defined class must inherit from Thread class of threading module.

Step-4: create an object of Programmer-defined class

Step-5: call the start() w.r.t the object of Programmer-defined class
object.start()

Here start() internally calling run() of Programmer-defined class

Examples:

SecondApproachEx1.py

```
import threading # step-1
```

```
# step-2 step-3
```

```
class Sample(threading.Thread):
```

```
    def run(self): # step-4
```

```
        print("\nName of sub thread=",threading.current_thread().name)
```

```
        print("i am from run())")
```

```
#main program
```

```
print("Default Name of thread=",threading.current_thread().name) #MainThread
```

```
#create a sub thread
```

```
t1=Sample()
```

```
t1.name="OOPS"
```

```
print("Is sub thread alive before start=",t1.is_alive())#False
```

```
t1.start() # step-5
```

```
print("Is sub thread alive after start=",t1.is_alive())#True
```

3) By Using Non-Sub Class of Thread class of threading module (with OOPs without Inheritance)

Step1: import threading

Step-2: Choose the Programmer-Defined Class

Step-3: Define Programmer-defined method in side of Programmer-Defined Class

Step-4: Create an object Programmer-defined class

Step-5: create sub thread w.r.t Thread class of threading module

Step-6: Call the start() upon the sub thread object

Examples:

#ThirdApproachEx1.py

```
import threading # step-1
```

```
class Test: # Step-2
```

```
    def hello(self,s): # step-3
```

```
        print("\nName of sub thread=",threading.current_thread().name)
```

```
        print("Hello ,{}, Good Morning".format(s))
```

```
        print("I am from hello() of Test Class")
```

```
        print("here hello() executed by sub thread")
```

```
#main program
```

```
t=Test() # Create an object Programmer-defined class--step-4
```

```
subth=threading.Thread(target=t.hello, args=( ("Rossum",) ))# # create sub thread--  
Step-5
```

```
subth.start() # # step-6
```

```
#subthreadsdemo.py
import threading
def fun1():
    print("Hi")
def fun2(s):
    print("Hi, ",s)
#main program
print("Default Name of thread=",threading.current_thread().name) # Main Thread
t1=threading.Thread(target=fun1) # here t1 is called sub thread
t2=threading.Thread(target=fun2,args=("Rossum", )) # here t2 is called sub thread
print("Default Name of sub thread=", t1.name) # Thread-1
print("default Name of sub thread=", t2.name) # Thread-2
#assign User-Friendly Name to the sub threads
t1.name="Rossum"      # t1.setName("Rossum")---is deprecated to an attribute
"name"
t2.name="Ritche"      # t2.setName("Ritche")---is deprecated to an attribute
"name"
print("Name of sub thread=", t1.name)
print("Name of sub thread=", t2.name)
```

```
#subthreadsstatus.py
```

```
import threading,time
def fun1():
    print("Hi")
    time.sleep(5)
def fun2(s):
    print("Hi, ",s)
    time.sleep(7)
#main program
print("Default Name of thread=",threading.current_thread().name) # Main Thread
t1=threading.Thread(target=fun1) # here t1 is called sub thread
t2=threading.Thread(target=fun2,args=("Rossum", )) # here t2 is called sub thread
print("Default Name of sub thread=", t1.name) # Thread-1
print("default Name of sub thread=", t2.name) # Thread-2
#assign User-Friendly Name to the sub threads
t1.name="Rossum"      # t1.setName("Rossum")---is deprecated to an attribute
"name"
t2.name="Ritche"      # t2.setName("Ritche")---is deprecated to an attribute
"name"
print("Name of sub thread=", t1.name)
print("Name of sub thread=", t2.name)
print("\nNumber of threads active before start()",threading.active_count())
print("Execution status of sub thread t1 before start()", t1.is_alive()) # False
print("Execution status of sub thread t2 before start()", t2.is_alive()) # False
t1.start()
t2.start()
print("\nNumber of threads active after start()",threading.active_count())
print("Execution status of sub thread t1 after start()", t1.is_alive()) # True
print("Execution status of sub thread t2 after start()", t2.is_alive()) # True
```

```
#subthreadsactivecount.py
import threading,time
def fun1():
    print("Hi")
    time.sleep(5)
def fun2(s):
    print("Hi, ",s)
    time.sleep(7)

#main program
print("Default Name of thread=",threading.current_thread().name) # Main Thread
t1=threading.Thread(target=fun1) # here t1 is called sub thread
t2=threading.Thread(target=fun2,args=("Rossum", )) # here t2 is called sub thread
print("Default Name of sub thread=", t1.name) # Thread-1
print("default Name of sub thread=", t2.name) # Thread-2
#assign User-Freindly Name to the sub threads
t1.name="Rossum"      # t1.setName("Rossum")---is deprecated to an attribute
"name"
t2.name="Ritche"      # t2.setName("Ritche")---is deprecated to an attribute
"name"
print("Name of sub thread=", t1.name)
print("Name of sub thread=", t2.name)
print("\nNumber of threads active before start()=",threading.active_count())
t1.start()
t2.start()
print("\nNumber of threads active after start()=",threading.active_count())
```

```
#subthreadsjoin.py
import threading,time
def fun1():
    print("Hi")
    time.sleep(10)

def fun2(s):
    print("Hi, ",s)
    time.sleep(12)

#main program
print("Default Name of thread=",threading.current_thread().name) # Main Thread
t1=threading.Thread(target=fun1) # here t1 is called sub thread
t2=threading.Thread(target=fun2,args=("Rossum", )) # here t2 is called sub thread
print("Default Name of sub thread=", t1.name) # Thread-1
print("default Name of sub thread=", t2.name) # Thread-2
#assign User-Freindly Name to the sub threads
t1.name="Rossum"      # t1.setName("Rossum")---is deprecated to an attribute
"name"
t2.name="Ritche"      # t2.setName("Ritche")---is deprecated to an attribute
"name"
```

```

print("Name of sub thread=", t1.name)
print("Name of sub thread=", t2.name)
print("\nNumber of threads active before start()=",threading.active_count()) # 1
print("Execution status of sub thread t1 before start()=", t1.is_alive()) # False
print("Execution status of sub thread t2 before start()=", t2.is_alive()) # False
t1.start()
t2.start()
print("\nNumber of threads active after start()=",threading.active_count()) # 3
print("Execution status of sub thread t1 after start()=", t1.is_alive()) # True
print("Execution status of sub thread t2 after start()=", t2.is_alive()) # True
#make the MainThread to wait until sub threads join.
t1.join()
t2.join()
print("\nLine-33:Number of threads active after completion of
exec=",threading.active_count()) # 1
print("Execution status of sub thread t1 after completion of exec=", t1.is_alive()) #
False
print("Execution status of sub thread t2 after completion of exec=", t2.is_alive()) #
False

```

```

#SecondApproachEx1.py
import threading # step-1
# step-2 step-3
class Sample(threading.Thread):
    def run(self): # step-4
        print("\nName of sub thread=",threading.current_thread().name)
        print("i am from run()")
#main program
print("Default Name of thread=",threading.current_thread().name) #MainThread
#create a sub thread
t1=Sample()
t1.name="OOPS"
print("Is sub thread alive before start=",t1.is_alive())#False
t1.start()
print("Is sub thread alive after start=",t1.is_alive())#True

```

```

#write a thread based application which will generate a multiplication table for a
given number by using threads with oop approach.
#SecondApproachEx2.py
import threading,time
class MulTable(threading.Thread):
    def run(self): # overriding the run()
        self.n=int(input("Enter a number:"))
        if(self.n<=0):
            print("{} is invalid number:".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))
            time.sleep(1)
        print("-"*50)

#main program
mt=MulTable() # creating sub thread
mt.start()
```

```
#ThirdApproachEx1.py
import threading # step-1
class Test: # Step-2
    def hello(self,s): # step-3
        print("\nName of sub thread=",threading.current_thread().name)
        print("Hello ,{}, Good Morning".format(s))
        print("I am from hello() of Test Class")
        print("here hello() executed by sub thread")

#main program
t=Test() # Create an object Programmer-defined class--step-4
subth=threading.Thread(target=t.hello, args=( ("Rossum",) )) # # create sub thread--Step-5
subth.start() # # step-6
```

```
#ThirdApproachEx2.py
import threading # step-1
import time
class Roohi: # Step-2
    def table(self,n): # step-3
        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
R=Roohi() # Create an object Programmer-defined class--step-4
t1=threading.Thread(target=R.table,args=( (int(input("Enter a number:")),) ) ) # create sub thread--Step-5
t1.start() # step-6
```

```

#EvenOddFirstApparch.py-----Functional Approach
import time,threading
def evennumbers(n):
    print("=*50")
    tn=threading.current_thread().name
    #print("Name of Sub Thread={}".format(tn))
    if(n<=0):
        print("{} is invalid input:".format(n))
    else:
        for i in range(2,n+1,2):
            print("\tEven Number generated by {}={}".format(tn,i))
            time.sleep(1)
    print("=*50")
def oddnumbers(n):
    print("=*50")
    tname=threading.current_thread().name
    #print("Name of Sub Thread={}".format(tname))
    if(n<=0):
        print("{} is invalid input:".format(n))
    else:
        for i in range(1,n+1,2):
            print("\tOdd Number generated by {} ={}".format(tname,i))
            time.sleep(1)
    print("=*50")

#main program
ont=threading.Thread(target=oddnumbers,args=((50,)) ) # Creating sub thread-1
ent=threading.Thread(target=evennumbers,args=((50,)) ) # Creating sub thread-2
#displatch the sub threads
ont.start()
ent.start()

```

```

#EvenOddSecondApparch.py-----OOPs Approach with Inheritance
import threading
import time
class EvenNumber(threading.Thread):
    def setnumber(self,n):
        self.n=n
    def run(self):
        print("=*50")
        tn=threading.current_thread().name
        if(self.n<=0):
            print("{} is invalid input:".format(self.n))
        else:
            for i in range(2,self.n+1,2):
                print("\tEven Number generated by"
{}={}.format(tn,i))
                time.sleep(1)

```

```

print("*50)

class OddNumber(threading.Thread):
    def setnumber(self,n):
        self.n=n
    def run(self):
        print("*50)
        tn=threading.current_thread().name
        if(self.n<=0):
            print("{} is invalid input:".format(self.n))
        else:
            for i in range(1,self.n+1,2):
                print("\tOdd Number generated by
{}={}.format(tn,i))
                time.sleep(1)
        print("*50)

#main program
#create two sub threads
ent=EvenNumber()
ont=OddNumber()
#set the value for sub threads
ent.setnumber(50)
ont.setnumber(50)
#dipatch the threads
ont.start()
ent.start()

```

```

#EvenOddThirdApparch.py-----OOPs Approach without Inheritance
import threading
import time
class EvenNumber:
    def evennums(self,n):
        print("*50)
        tn=threading.current_thread().name
        if(n<=0):
            print("{} is invalid input:".format(n))
        else:
            for i in range(2,n+1,2):
                print("\tEven Number generated by
{}={}.format(tn,i))
                time.sleep(1)
        print("*50)

class OddNumber:
    def oddnums(self,n):
        print("*50)
        tn=threading.current_thread().name
        if(n<=0):
            print("{} is invalid input:".format(n))

```

```

else:
    for i in range(1,n+1,2):
        print("\tOdd Number generated by
{}{}".format(tn,i))
        time.sleep(1)
    print("*"*50)

#main program
otn=OddNumber()
ent=EvenNumber()
#create sub threads
st1=threading.Thread(target=otn.oddnums,args=((40,)) )
st2=threading.Thread(target=ent.evennums,args=((40,)) )
#dispatch the threads
st1.start()
st2.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.

Syntax:- **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 current 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 sharable resource. This process of acquiring the releasing the lock will be continued until all the objects completed their execution.

#A Program for showing Inconsistent Result (or) non-thread safety result in threading program

#nonlockingFunctional.py

import time

import threading

```

def multable(n):
    print("Name of Sub Thread in multable()=",threading.current_thread().name)
    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
t1=threading.Thread(target=multable,args=((6,)) )
t2=threading.Thread(target=multable,args=((16,)) )
t3=threading.Thread(target=multable,args=((19,)) )
t4=threading.Thread(target=multable,args=((14,)) )
t1.start()
t2.start()
t3.start()
t4.start()

```

```

#A Program for showing Inconsistent Result (or) non-thread safety result in
threading program
#lockingFunctional.py
import time
import threading
def multable(n):
    L.acquire() # obtain the lock--Step-2
    print("Name of Sub Thread in multable()=",threading.current_thread().name)
    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)
    L.release() # release the lock--Step-3

#main program
#create an object of Lock of threading module--Step-1
L=threading.Lock() # here L is an object of Lock class
t1=threading.Thread(target=multable,args=((6,)) )
t2=threading.Thread(target=multable,args=((16,)) )
t3=threading.Thread(target=multable,args=((19,)) )
t4=threading.Thread(target=multable,args=((14,)) )

```

```
t1.start()  
t2.start()  
t3.start()  
t4.start()
```

```
#A Program for showing Inconsistant Result (or) non-thread safety result in  
threading program
```

```
#NonLockingOopswithInh.py
```

```
import threading,time
```

```
class MulTable(threading.Thread):
```

```
    def setvalue(self,n):
```

```
        self.n=n
```

```
    def run(self):
```

```
        print("Name of Sub Thread in
```

```
run()=",threading.current_thread().name)
```

```
        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))
```

```
                time.sleep(1)
```

```
            print("-"*50)
```

```
#main program
```

```
t1=MulTable()
```

```
t2=MulTable()
```

```
t3=MulTable()
```

```
t4=MulTable()
```

```
#set the values to the sub threads
```

```
t1.setvalue(3)
```

```
t2.setvalue(14)
```

```
t3.setvalue(20)
```

```
t4.setvalue(-13)
```

```
#dispatch the threads
```

```
t1.start()
```

```
t2.start()
```

```
t3.start()
```

```
t4.start()
```

```
#A Program for showing Inconsistant Result (or) non-thread safety result in  
threading program
```

```
#LockingOopswithInh.py
```

```
import threading,time
```

```
class MulTable(threading.Thread):
```

```
    def setvalue(self,n):
```

```

        self.n=n
    def run(self):
        L.acquire() #step-2
        print("Name of Sub Thread in
run()=",threading.current_thread().name)
        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))
                time.sleep(1)
            print("-"*50)
        L.release() #step-3
#main program
L=threading.Lock() # Step-1
t1=MulTable()
t2=MulTable()
t3=MulTable()
t4=MulTable()
#set the values to the sub threads
t1.setvalue(3)
t2.setvalue(14)
t3.setvalue(20)
t4.setvalue(-13)
#dispatch the threads
t1.start()
t2.start()
t3.start()
t4.start()

```

```

#A Program for showing Inconsistent Result (or) non-thread safety result in
threading program
#NonLockingOopswithoutInhex.py
import threading,time
class MulTable:
    def __init__(self):
        self.n=int(input("Enter a Number for Mul Table:"))
    def table(self):
        print("Name of Sub Thread in
run()=",threading.current_thread().name)
        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))
            time.sleep(1)
        print("-"*50)

#main program
t1=MulTable()
t2=MulTable()
t3=MulTable()
t4=MulTable()
#create sub threads
st1=threading.Thread(target=t1.table)
st2=threading.Thread(target=t2.table)
st3=threading.Thread(target=t3.table)
st4=threading.Thread(target=t4.table)

#dispatch the threads
st1.start()
st2.start()
st3.start()
st4.start()

```

```

#A Program for showing Inconsistent Result (or) non-thread safety result in
threading program
#LockingOopswithoutInhex1.py
import threading,time
class MulTable:
    @classmethod
    def getlock(cls):
        cls.KVR=threading.Lock() # creating a lock on the name of "KVR"---
Step-1
    def __init__(self):
        self.n=int(input("Enter a Number for Mul Table:"))
    def table(self):
        MulTable.KVR.acquire() #Step-2
        print("Name of Sub Thread in
run()=",threading.current_thread().name)
        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))
                time.sleep(1)
            print("-"*50)
        MulTable.KVR.release() #Step-3
#main program
MulTable.getlock() # Calling Class Level Method

```

```
t1=MulTable()  
t2=MulTable()  
t3=MulTable()  
t4=MulTable()  
#create sub threads  
st1=threading.Thread(target=t1.table)  
st2=threading.Thread(target=t2.table)  
st3=threading.Thread(target=t3.table)  
st4=threading.Thread(target=t4.table)  
#dispatch the threads  
st1.start()  
st2.start()  
st3.start()  
st4.start()
```

NumPy

Introduction to Numpy:

- =>Numpy stands for Numerical Python.
- =>Numpy is one of the pre-defined third party module / Library and numpy module is not a pre-defined module in Python Language.
- =>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.
-

Advantages of using NumPy

Need of NumPy:

=>With the revolution of data science, data analysis libraries like NumPy, SciPy, Scikit, Pandas, etc. have seen a lot of growth. With a much easier syntax than other programming languages, python is the first choice language for the data scientist.
=>NumPy provides a convenient and efficient way to handle the vast amount of data. NumPy is also very convenient with Matrix Operations and data reshaping. NumPy is fast which makes it reasonable to work with a large set of data.

The advantages of Numpy Programming are:

- 1) With Numpy Programming, we can deal with Arrays such 1-D, 2-D and Multi Dimensional Arrays.
 - 2) NumPy maintains minimal memory:
 - 3) Numpy provides Fast in Performing Operations because internally its data is available at same address.
 - 4) NumPy performs array-oriented computing.
 - 5) It efficiently implements the multidimensional arrays.
 - 6) It performs scientific computations.
 - 7) It is capable of performing reshaping the data stored in multidimensional arrays.
 - 8) NumPy provides Many in-built functions for Various Complex Mathematical Operations such as statistical , financial, trigonometric Operations etc.
-

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. Hence an object of ndarray can store same type or different type and both types (unique +duplicates) in single object.
=>The objects of ndarray and list are mutable (changes can takes place)

Differences between Python Traditional List and ndarray object of Numpy Module:

=>An object of list contains both homogeneous and heterogeneous 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 by treating all values of type "object").
=>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 efficient because list object values takes more

time to extract or retrieve (they are available at different Address) where as numpy based applications are efficient because of ndarray object values takes less time to extract or retrieve(they are available at same Address).

=>List object can't perform complex mathematical operations where as an object of ndarray can perform complex mathematical operations.

ndarray

=>ndarray is one of the pre-defined class present in numpy module.

=>The purpose of ndarray class and whose is that " To store the data in entire numpy programming in the form Array."

=>An object of ndarray can be created in 7 ways.

1. array()
 2. arange()
 3. zeros()
 4. ones()
 5. full()
 6. eye()
 7. identity()
-

Number of approaches to create an object of ndarray

=>In numpy programming, we have 7 approaches to create an object of ndarray. They are

1. array()
 2. arange()
 3. zeros()
 4. ones()
 5. full()
 6. eye()
 7. identity()
-

1) array():

=>This Function is used for converting Traditional Python Objects into ndarray object.

=>Syntax:- varname=numpy.array(Object,dtype)

Here **varname** is an object of <class,ndarray>
here **array()** is pre-defined function of numpy module used for converting Traditional Python Objects into ndrray object.

object represents any Traditional Python Objects

dtype represents any numpy data type such as int8,int16,int32,float16, float 32, float64,...etc

Examples:

```
>>> import numpy as np
>>> l1=[10,20,30,40,50,60]
>>> print(l1,type(l1))-----[10, 20, 30, 40, 50, 60] <class 'list'>
>>> a=np.array(l1)
>>> print(a,type(a))-----[10 20 30 40 50 60] <class 'numpy.ndarray'>
>>> t=(10,20,30,40,50,60,70)
>>> print(t,type(t))----(10, 20, 30, 40, 50, 60, 70) <class 'tuple'>
>>> a=np.array(t)
>>> print(a,type(a))-----[10 20 30 40 50 60 70] <class 'numpy.ndarray'>
>>> d1={10:1.2,20:4.5,30:6.7}
>>> a=np.array(d1)
>>> a---array({10: 1.2, 20: 4.5, 30: 6.7}, dtype=object)
-----
>>> t=(10,20,30,40,50,60)
>>> a=np.array(t)
>>> a-----array([10, 20, 30, 40, 50, 60])
>>> a.ndim-----1
>>> a.dtype-----dtype('int32')
>>> a.shape-----(6,)
>>> b=a.reshape(3,2)
>>> c=a.reshape(2,3)

>>> b-----
      array([[10, 20],
             [30, 40],
             [50, 60]])
>>> c
      array([[10, 20, 30],
             [40, 50, 60]])
>>> print(b,type(b))
      [[10 20]
       [30 40]
       [50 60]] <class 'numpy.ndarray'>
>>> d=a.reshape(3,3)-----ValueError: cannot reshape array of size 6 into shape (3,3)
-----
>>> t1=((10,20),(30,40))
>>> print(t1,type(t1))----((10, 20), (30, 40)) <class 'tuple'>
>>> a=np.array(t1)
>>> a
      array([[10, 20],
             [30, 40]])
```

```

>>> a.ndim-----2
>>> a.shape----(2, 2)
-----
>>> t1=((10,20,15),(30,40,25)),((50,60,18),(70,80,35))
>>> print(t1,type(t1))
((10, 20, 15), (30, 40, 25)), ((50, 60, 18), (70, 80, 35)) <class 'tuple'>
>>> a=np.array(t1)
>>> print(a)-----
[[[10 20 15]
 [30 40 25]]

 [[50 60 18]
 [70 80 35]]]

>>> a.ndim-----3
>>> a.shape----(2, 2, 3)
>>> b=a.reshape(4,3)
>>> b-----
array([[10, 20, 15],
       [30, 40, 25],
       [50, 60, 18],
       [70, 80, 35]])

>>> c=a.reshape(3,4)
>>> c-----
array([[10, 20, 15, 30],
       [40, 25, 50, 60],
       [18, 70, 80, 35]])

>>> d=a.reshape(3,2,2)
>>> d-----
array([[[10, 20],
        [15, 30]],

       [[40, 25],
        [50, 60]],

       [[18, 70],
        [80, 35]]])

>>> d[0]-----
array([[10, 20],
       [15, 30]])

>>> d[1]-----
array([[40, 25],
       [50, 60]])

>>> d[2]-----
array([[18, 70],
       [80, 35]])
-----
```

2. arange():

Syntax1:- varname=numpy.arange(Value)
Syntax2:- varname=numpy.arange(Start,Stop)
Syntax3:- varname=numpy.arange(Start,Stop,Step)
=>Here var name is an object of <class,ndarray>

=>Syntax-1 creates an object of ndarray with the values from 0 to value-1
=>Syntax-2 creates an object of ndarray with the values from Start to Stop-1
=>Syntax-3 creates an object of ndarray with the values from Start to Stop-1 with equal
Interval of Value-step
=>arange() always create an object of ndarray in 1-D array only but not Possible to create directly 2-D and Multi Dimesional Arrays.
=>To create 2-D and Multi Dimesional Arrays, we must use reshape()

Examples:

```
>>> import numpy as np
>>> a=np.arange(10)
>>> a-----array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
>>> a.ndim-----1
>>> a=np.arange(50,62)
>>> print(a,type(a))---[50 51 52 53 54 55 56 57 58 59 60 61] <class 'numpy.ndarray'>
>>> a.ndim-----1
>>> a=np.arange(10,23,2)
>>> a----array([10, 12, 14, 16, 18, 20, 22])
>>> a=np.arange(10,22,2)
>>> a-----array([10, 12, 14, 16, 18, 20])
>>> b=a.reshape(2,3)
>>> c=a.reshape(3,2)
>>> b-----
        array([[10, 12, 14],
               [16, 18, 20]])
>>> c
        array([[10, 12],
               [14, 16],
               [18, 20]])
>>> b.ndim----- 2
>>> c.ndim----- 2
>>> b.shape----(2, 3)
>>> c.shape----(3, 2)
>>> l1=[ [[10,20],[30,40]], [[15,25],[35,45]] ]
>>> l1-----[[[10, 20], [30, 40]], [[15, 25], [35, 45]]]
>>> a=np.arange(l1)-----TypeError: unsupported operand type(s) for -: 'list' and 'int'
```

3. zeros():

=>This Function is used for building ZERO matrix either with 1-D or 2-D or n-D

=>Syntax: varname=numpy.zeros(shape,dtype)

=>Here Shape can be 1-D(number of Zeros) or 2-D(Rows,Cols) or n-D(Number of Matrices,Number of Rows, Number of Columns)

```
#zero matrix demonstration
#zm.py
import numpy as np
m,n=int(input("Enter Number of Rows:")), int(input("Enter Number of Columns:"))
if(m<=0) or (n<=0):
    print("Invalid Dimensions:")
else:
    zm=np.zeros( (m,n),dtype=int)
    print(zm)
```

Examples:

```
>>> import numpy as np
>>> a=np.zeros(12)
>>> a-----array([0., 0., 0., 0., 0., 0., 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])
>>> a.reshape(3,4)
        array([[0, 0, 0, 0],
               [0, 0, 0, 0],
               [0, 0, 0, 0]])

>>> a.reshape(2,6)
        array([[0, 0, 0, 0, 0, 0],
               [0, 0, 0, 0, 0, 0]])
>>> a.reshape(2,3,2)
        array([[[0, 0],
                  [0, 0],
                  [0, 0]],
                  [[0, 0],
                  [0, 0],
                  [0, 0]]])

>>> a.reshape(2,2,2)----ValueError: cannot reshape array of size 12 into shape
(2,2,2,2)
```

```
>>> import numpy as np
>>> a=np.zeros((3,3),dtype=int)
>>> a
        array([[0, 0, 0],
               [0, 0, 0],
               [0, 0, 0]])
>>> a=np.zeros((2,3))
```

```

>>> a
      array([[0., 0., 0.],
             [0., 0., 0.]])
>>> a=np.zeros((2,3),int)
>>> a
      array([[0, 0, 0],
             [0, 0, 0]])
>>> a=np.zeros((3,2,3),dtype=int)
>>> a
      array([[[0, 0, 0],
              [0, 0, 0]],
              [[0, 0, 0],
              [0, 0, 0]],
              [[0, 0, 0],
              [0, 0, 0]]])
>>> print(a,type(a))
      [[[0 0 0]
      [0 0 0]]
      [[0 0 0]
      [0 0 0]]
      [[0 0 0]
      [0 0 0]]] <class 'numpy.ndarray'>

```

4. ones()

=>This Function is used for building ONEs matrix either with 1-D or 2-D or n-D
=>**Syntax:** **varname=numpy.ones(shape,dtype)**

=>Here Shape can be 1-D(number of Zeros) or 2-D(Rows,Cols) or n-D(Number of Matrices,Number of Rows, Number of Columns)

Examples:

```

>>> import numpy as np
>>> a=np.ones(10)
>>> print(a,type(a))-----[1. 1. 1. 1. 1. 1. 1. 1. 1. 1.] <class 'numpy.ndarray'>
>>> a=np.ones(10,dtype=int)
>>> print(a,type(a))-----[1 1 1 1 1 1 1 1 1 1] <class 'numpy.ndarray'>
>>> a.shape-----(10,)
>>> a.shape=(5,2)
>>> a
      array([[1, 1],
             [1, 1],
             [1, 1],
             [1, 1],
             [1, 1]])
>>> a.ndim----- 2

```

```

>>> a.shape----- (5, 2)
>>> a.shape=(2,5)
>>> a
      array([[1, 1, 1, 1, 1],
             [1, 1, 1, 1, 1]])
>>> a.shape----- (2, 5)
>>>
>>> a=np.ones((3,4),dtype=int)
>>> a
      array([[1, 1, 1, 1],
             [1, 1, 1, 1],
             [1, 1, 1, 1]])
>>> a=np.ones((4,3),dtype=int)
>>> print(a,type(a))
[[1 1 1]
 [1 1 1]
 [1 1 1]
 [1 1 1]] <class 'numpy.ndarray'>
>>> a.shape----- (4, 3)
>>> a.shape=(3,2,2)
>>> a
      array([[[1, 1],
                 [1, 1]],
                 [[1, 1],
                 [1, 1]]])
>>> a=np.ones((4,3,3),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]]]])
>>> a[0][0][0]-----1
>>> a[0,0,0]-----1
>>> a[0][0,0]-----1

```

```
=====
```

5) full()

=>This function is used for building a matrix by specifying fill value either 1-D or 2-D or n-D

=>Syntax:-

```
varname=numpy.full(shape,fill_value,dtype)
```

=>varname is an object of <class, numpy.ndarray>

=>Here Shape can be 1-D(number of Zeros) or 2-D(Rows,Cols) or n-D(Number of Matrices,Number of Rows, Number of Columns)

=>fill_value can be any number of programmer choice

Examples:

```
>>> a=np.full(3,1)
>>> a-----array([1, 1, 1])
>>>print(type(a))-----<class,numpy.ndarray>
>>> a=np.full(6,8)
>>> a-----array([8, 8, 8, 8, 8, 8])
>>> a.shape=(3,2)
>>> a
array([[8, 8],
       [8, 8],
       [8, 8]])
>>> a=np.full(6,9)
```

```
>>> a=np.full((3,3,3),7)
```

```
>>> a
array([[[7, 7, 7],
         [7, 7, 7],
         [7, 7, 7]],
        [[7, 7, 7],
         [7, 7, 7],
         [7, 7, 7]],
        [[7, 7, 7],
         [7, 7, 7],
         [7, 7, 7]]])
```

```
=====
```

6. eye():

=>This function is used for building Identity matrix or unit matrix.

=>Syntax:- varname=numpy.eye(N,M=None,K=0,dtype)

=>Here N represents number of rows

=>Here M represents number of Columns

=>If we don't represent M value then N value will be taken M values and prepared Quare Unit Matrix.

=>Here K represents Principle diagonal element. if k=0 then it always points principle diagonal

=>If k=-1,-2,-3...-n represents Below Principle diagonal elements.

=>If k=1,2,3...-n represents Above Principle diagonal elements.

Examples:

```
>>> import numpy as np
>>> #varname=numpy.eye(N,M=None,K=0,dtype
>>> a=np.eye(3)
>>> print(a,type(a))
[[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]] <class 'numpy.ndarray'>
>>> a=np.eye(3,dtype=int)
>>> print(a,type(a))
[[1 0 0]
 [0 1 0]
 [0 0 1]] <class 'numpy.ndarray'>
>>> a=np.eye(3,4,dtype=int)
>>> print(a,type(a))
[[1 0 0 0]
 [0 1 0 0]
 [0 0 1 0]] <class 'numpy.ndarray'>
>>> a=np.eye(3,4,dtype=int)
>>> print(a,type(a))
[[1 0 0 0]
 [0 1 0 0]
 [0 0 1 0]] <class 'numpy.ndarray'>
>>> a=np.eye(5,6,dtype=int)
>>> print(a,type(a))
[[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]] <class 'numpy.ndarray'>
>>> a=np.eye(3,4,k=-1,dtype=int)
>>> print(a,type(a))
[[0 0 0 0]
 [1 0 0 0]
 [0 1 0 0]] <class 'numpy.ndarray'>
>>> a=np.eye(5,6,k=-1,dtype=int)
>>> print(a,type(a))
[[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]] <class 'numpy.ndarray'>
>>> a=np.eye(3,4,k=-2,dtype=int)
>>> a=np.eye(5,6,k=-2,dtype=int)
```

```

>>> print(a,type(a))
[[0 0 0 0 0]
 [0 0 0 0 0]
 [1 0 0 0 0]
 [0 1 0 0 0]
 [0 0 1 0 0]] <class 'numpy.ndarray'>
>>> a=np.eye(5,6,k=-3,dtype=int)
>>> print(a,type(a))
[[0 0 0 0 0]
 [0 0 0 0 0]
 [0 0 0 0 0]
 [1 0 0 0 0]
 [0 1 0 0 0]] <class 'numpy.ndarray'>
```

```

>>> a=np.eye(5,6,k=-4,dtype=int)
>>> print(a,type(a))
[[0 0 0 0 0]
 [0 0 0 0 0]
 [0 0 0 0 0]
 [0 0 0 0 0]
 [1 0 0 0 0]] <class 'numpy.ndarray'>
>>> a=np.eye(5,6,k=1,dtype=int)
>>> print(a,type(a))
[[0 1 0 0 0]
 [0 0 1 0 0]
 [0 0 0 1 0]
 [0 0 0 0 1]
 [0 0 0 0 1]] <class 'numpy.ndarray'>
>>> a=np.eye(5,6,k=2,dtype=int)
>>> print(a,type(a))
[[0 0 1 0 0]
 [0 0 0 1 0]
 [0 0 0 0 1]
 [0 0 0 0 0]
 [0 0 0 0 0]] <class 'numpy.ndarray'>
>>> a=np.eye(5,6,k=3,dtype=int)
>>> print(a,type(a))
[[0 0 0 1 0]
 [0 0 0 0 1]
 [0 0 0 0 0]
 [0 0 0 0 0]
 [0 0 0 0 0]] <class 'numpy.ndarray'>
=====
```

7) identity():

=>This function always build Identity or unit matrix
=>Syntax:- varname=numpy.identity(N,dtype)
=>Here N represents Either we can take Rows or Columns and PVM takes as NXN Matrix (Square Matrix--Unit or Identity)

Examples:

```
>>> import numpy as np  
>>> a=np.identity(3,dtype=int)  
>>> print(a,type(a))-----  
[[1 0 0]  
 [0 1 0]  
 [0 0 1]] <class 'numpy.ndarray'>  
>>> a=np.identity(5,dtype=int)  
>>> print(a,type(a))  
[[1 0 0 0 0]  
 [0 1 0 0 0]  
 [0 0 1 0 0]  
 [0 0 0 1 0]  
 [0 0 0 0 1]] <class 'numpy.ndarray'>
```

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 performed 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
-----
>>> 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,ndarray2)

=>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

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.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)
```

NumPy–Statistical Operations

=>On the object of ndarray, we can perform the following Statistical Operations .

- a) amax()
- b) amin()
- c) mean()
- d) median()
- e) var()
- f) std()

=>These operations we can perform on the entire matrix and we can also perform on columnwise (axis=0) and Rowwise (axis=1)

a) amax(): (biggest value)

=>This function obtains maximum element of the entire matrix.

=>Syntax1:- varname=numpy.amax(ndarrayobject)

=>Syntax2:- varname=numpy.amax(ndarrayobject, axis=0)-->obtains max

=>Syntax3:- varname=numpy.amax(ndarrayobject, axis=1)-->obtains max

Examples:

```
>>> l1=[[1,2,3],[4,2,1],[3,4,2]]
```

```
>>> A=np.array(l1)
```

```
>>> print(A)
```

```
[[1 2 3]
 [4 2 1]
 [3 4 2]]
```

```
>>> max=np.amax(A)
```

```
>>> cmax=np.amax(A, axis=0)
```

```
>>> rmax=np.amax(A, axis=1)
```

```
>>> print("Max element=",max)-----Max element= 4
```

```
>>> print("Column Max elements=",cmax)--Column Max elements= [4 4 3]
```

```
>>> print("Row Max elements=",rmax)--Row Max elements= [3 4 4]
```

b) amin(): (smallest value)

=>This functions obtains minimum element of the entire matrix.

=>Syntax1:- varname=numpy.amin(ndarrayobject)
=>obtain min element of whole matrix

=>Syntax2:- varname=numpy.amin(ndarrayobject, axis=0)
=>obtains min elements on the basis columns.

=>Syntax3:- varname=numpy.amin(ndarrayobject, axis=1)
=>obtains min elements on the basis Rows.

Examples:

```
>>> l1=[[1,2,3],[4,2,1],[3,4,2]]  
>>> A=np.array(l1)  
>>> print(A)  
     [[1 2 3]  
      [4 2 1]  
      [3 4 2]]  
>>> min=np.amin(A)  
>>> cmin=np.amin(A, axis=0)  
>>> rmin=np.amin(A, axis=1)  
>>> print("Min eleemnt=",min)--Min eleemnt= 1  
>>> print("Column Min eleemnts=",cmin)--Column Min eleemnts= [1 2 1]  
>>> print("Row Min eleemnts=",rmin)--Row Min eleemnts= [1 1 2]
```

c) mean(): (average)

=>This is used for cal mean of the total matrix elements.

=>The formula for mean=(sum of all elements of matrix)/ total number of elements.

Syntax1:- varname=numpy.mean(ndarrayobject)

Syntax2:- varname=numpy.mean(ndarrayobject, axis=0)-->Columnwise Mean

Syntax3:- varname=numpy.mean(ndarrayobject, axis=1)-->Rowwise Mean

Examples:

```
>>> l1=[[1,2,3],[4,2,1],[3,4,2]]  
>>> A=np.array(l1)  
>>> print(A)  
     [[1 2 3]  
      [4 2 1]  
      [3 4 2]]  
>>> m=np.mean(A)  
>>> cm=np.mean(A, axis=0)  
>>> rm=np.mean(A, axis=1)  
>>> print("Mean=",m)-----Mean= 2.4444444444444446  
>>> print("Column Mean=",cm)----Column Mean= [2.66666667  2.66666667    2.]  
>>> print("Row Mean=",rm)--Row Mean= [ 2.        2.33333333     3. ]
```

d) median()

=>This is used for calculating / obtaining median of entire matrix elements.
=>Median is nothing but sorting the given data in ascending order and select middle element.
=>If the number sorted elements are odd then centre / middle element becomes median.
=>If the number sorted elements are even then select centre / middle of two elements, add them and divided by 2 and that result becomes median.

Syntax1:- varname=numpy.median(ndarrayobject)

Syntax2:- varname=numpy.median(ndarrayobject, axis=0)

Syntax3:- varname=numpy.median(ndarrayobject, axis=1)

Examples:

```
=====
>>> l1=[[1,2,3],[4,2,1],[3,4,2]]
>>> A=np.array(l1)
>>> print(A)
[[1 2 3]
 [4 2 1]
 [3 4 2]]
>>> md=np.median(A)
>>> cmd=np.median(A,axis=0)
>>> rmd=np.median(A,axis=1)
>>> print("Median=",md)--Median= 2.0
>>> print("Column Median=",cmd)--Column Median= [3. 2. 2.]
>>> print("Row Median=",rmd)-----Row Median= [2. 2. 3.]
>>> l1=[[2,3],[4,1]]
>>> A=np.array(l1)
>>> print(A)
[[2 3]
 [4 1]]
>>> md=np.median(A)
>>> cmd=np.median(A,axis=0)
>>> rmd=np.median(A,axis=1)
>>> print("Median=",md)--Median= 2.5
>>> print("Column Median=",cmd)--Column Median= [3. 2.]
>>> print("Row Median=",rmd)--Row Median= [2.5 2.5]
=====
```

e) var(): $(\text{mean}-x_i)^2 / \text{total number of elements}$

Variance= $\text{sqr}(\text{mean}-x_i) / \text{total number of elements}$
here 'xi' represents each element of matrix.

Syntax1:- varname=numpy.var(ndarrayobject)

Syntax2:- varname=numpy.var(ndarrayobject, axis=0)

Syntax3:- varname=numpy.var(ndarrayobject, axis=1)

Examples:

```
>>> l1=[[1,2,3],[4,2,1],[3,4,2]]  
>>> A=np.array(l1)  
>>> print(A)  
 [[1 2 3]  
 [4 2 1]  
 [3 4 2]]  
  
>>> vr=np.var(A)  
>>> cvr=np.var(A, axis=0)  
>>> rvr=np.var(A, axis=1)  
>>> print("Variance=",vr)---Variance= 1.1358024691358024  
>>> print("Column Variance=",cvr)---Column Variance= [1.55555556 0.88888889  
  
0.66666667]  
>>> print("Row Variance=",rvr)---Row Variance= [0.66666667 1.55555556 0.66666667]  
=====
```

f) std(): standard deviation=sqrt(var)

Syntax1:- varname=numpy.std(ndarrayobject)

Syntax2:- varname=numpy.std(ndarrayobject, axis=0)

Syntax3:- varname=numpy.std(ndarrayobject, axis=1)

Examples:

```
>>> l1=[[1,2,3],[4,2,1],[3,4,2]]  
>>> A=np.array(l1)  
>>> print(A)  
 [[1 2 3]  
 [4 2 1]  
 [3 4 2]]  
  
>>> vr=np.var(A)  
>>> cvr=np.var(A, axis=0)  
>>> rvr=np.var(A, axis=1)  
>>> print("Variance=",vr)---Variance= 1.1358024691358024  
>>> print("Column Variance=",cvr)---Column Variance= [1.55555556 0.88888889  
  
0.66666667]  
>>> print("Row Variance=",rvr)---Row Variance= [0.66666667 1.55555556 0.66666667]  
=====
```



```
>>> sd=np.std(A)  
>>> csd=np.std(A, axis=0)  
>>> rsd=np.std(A, axis=1)
```

```
>>> print("std=",sd)--std= 1.0657403385139377  
>>> print(" column std=",csd)-- column std= [1.24721913 0.94280904 0.81649658]  
>>> print("Row std=",rsd)--Row std= [0.81649658 1.24721913 0.81649658]  
=====
```

```
=====
```

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 → 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]
(matrix) (row) (column)

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]]])  
=====
```

===== 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 full fill 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  
>>>lst=[0,2,7] here [0,2,7] are indexes of 10 30 and 80  
>>> indexes=np.array(lst) # here lst converted into ndarray object  
>>> 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 diagonal 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])
=====
OR
=====

```

```
>>> 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] ] ]
>>> a=np.array(l1)
>>> a
array([[[ 1,  2,  3,  4],
       [ 5,  6,  7,  8],
       [ 9, 10, 11, 12]],

      [[13, 14, 15, 16],
       [17, 18, 19, 20],
       [21, 22, 23, 24]]])
```

```
>>> #ndarrayobj[ [MatrixIndex],[Row Index],[Col Index] ]---Syntax
>>> #ndarrayobj[ [MatrixIndex],[Row Index],[Col Index] ]
```

```
>>> #access 1,8,13,20
>>> matind=(0,0,1,1)
>>> rowind=(0,1,0,1)
>>> colind=(0,3,0,3)
>>> a[matind,rowind,colind]
```

```
array([ 1,  8, 13, 20])
>>> a[ [0,0,0,1,1,1],[0,1,2,0,1,2],[0,1,2,0,1,2] ]
array([ 1,  6, 11, 13, 18, 23])
=====X=====
```

```
a=np.array([10,20,30,40,50,50,60,70,80,15,25,35,45,55,65,75,85])
a.shape=(2,2,2,2)
print(a)
```

```
[[[10 20]
  [30 40]]]
```

```
[[50 60]
  [70 80]]]
```

```
[[[15 25]]]
```

```

[35 45]
[[55 65]
 [75 85]]]
#access 10 from a---4-D
a[0][0][0][0]-----10
# access 10 and 40 from a---4D
a[[0,0],[0,0],[0,1],[0,1]]---array([10, 40])
# access 60,55 and 15 from a---4D
a[ [0,1,1],[1,1,0],[0,0,0],[1,0,0] ]---array([60, 55, 15])
=====
=====
```

NumPy—selecting the elements based on condition

(OR)

Creating Filter Directly From ndarray

=>To select any element from ndarray object, we have 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 Positive 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 Array Copy vs View

=>The Difference Between Copy and View

=>The main difference between a copy and a view of an array is that the copy is a new array, and the view is just a view of the original array.

=>The copy owns the data and any changes made to the copy will not affect original array, and any changes made to the original array will not affect the copy. modifications are Independent (Like Shallow Copy)

=>Syntax:- varname=ndrrayobj.copy()
 (OR)
 ndarrayobj2=numpy.copy(ndarrayobj1)

=>The view does not own the data and any changes made to the view will affect the original array, and any changes made to the original array will affect the view.

=>Syntax:- varname=ndrrayobj.view()

COPY:

Examples:

```
# Make a copy, change the original array, and display both arrays:
import numpy as np
arr = np.array([1, 2, 3, 4, 5])
x = arr.copy()
arr[0] = 42
print(arr)      # [42 2 3 4 5]
print(x)        # [1 2 3 4 5]
```

NOTE: The copy SHOULD NOT be affected by the changes made to the original array.

VIEW:

Example

#Make a view, change the original array, and display both arrays:

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5])
x = arr.view()
arr[0] = 42
print(arr) # [42 2 3 4 5]
print(x) # [42 2 3 4 5]
```

NOTE : The view SHOULD be affected by the changes made to the original array.

Make Changes in the VIEW:

Example

Make a view, change the view, and display both arrays:

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5])
x = arr.view()
x[0] = 31

print(arr) # [31 2 3 4 5]
print(x) # [31 2 3 4 5]
```

NumPy Sorting Arrays

=>Sorting is nothing arranging the elements in an ordered sequence.

=>Ordered sequence is any sequence that has an order corresponding to elements, like numeric or alphabetical, ascending or descending.

=>The NumPy ndarray object has a function called sort(), that will sort a specified array.

Examples:

```
import numpy as np
arr = np.array([3, 2, 0, 1])
print(np.sort(arr)) # [0 1 2 3]
```

```
import numpy as np
arr = np.array(['banana', 'cherry', 'apple'])
print(np.sort(arr)) # ['apple' 'banana' 'cherry']
```

```
import numpy as np
arr = np.array([True, False, True])
print(np.sort(arr)) # [False True True]
```

Sorting a 2-D Array

If you use the sort() method on a 2-D array, both arrays will be sorted:

Examples:

```
import numpy as np
arr = np.array([[3, 2, 4], [5, 0, 1]])
print(np.sort(arr))
=====
#output
[[2 3 4]
 [0 1 5]]
=====
```

1) delete():

=>numpy module contain a delete() to remove an array element with respect to its INDEX.

Syntax: numpy.delete(ndarrayobj , INDEX)

=>here numpy is a module which contain delete function

=> ndarrayobj is variable name of ndarray class

Example:

```
>>> lst=[10,20,30,40,50,60,70,80,90]
>>> arr=np.array(lst)
>>> print(arr)-----[10 20 30 40 50 60 70 80 90]
>>> np.delete(arr,1)-----array([10, 30, 40, 50, 60, 70, 80, 90])
=====
```

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 MC Kinney in the year 2008.
=>The Traditional Python Programming does not contain any Module for Data Analysis and Now Python Programming uses Pandas as a Data analysis tool.
=>Python Pandas can be used in wide range of fields like Financial Services, Statistics , retail marketing sectors ..etc as data analysis tool
=>pandas module developed in C and Python Languages.

Installation 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.
=>Programmatically 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 customised 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
-

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 a Series:

=>A Series object can be created by using the following 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, dictetc (But not set type bcoz they are un-
ordered)
=>'index' represents 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. Programmatically 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
```

```
Ist=[[10,20,30,40],["RS","JS","MCK","TRV"]]  
df=pd.DataFrame(Ist)  
print(df)
```

```
      0 1 2 3  
0 10 20 30 40  
1 RS JS MCK TRV
```

```
Ist=[[10,'RS'],[20,'JG'],[30,'MCK'],[40,'TRA']]  
df=pd.DataFrame(Ist)  
print(df)
```

```
      0 1  
0 10 RS  
1 20 JG  
2 30 MCK  
3 40 TRA
```

```
Ist=[[10,'RS'],[20,'JG'],[30,'MCK'],[40,'TRA']]  
df=pd.DataFrame(Ist, 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":["Python","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
```

===== Accessing the Data of DataFrame =====

1) DataFrameobj.head(no.of rows) :

=>head() is used to access the topmost rows.

=> **no. of rows** represents that how many rows you want to select from top. If we does not pass any value then by default it select 5 rows.

2) DataFrameobj.tail(no.of rows) :

=>tail() is used to access the bottom most rows.

=> **no. of rows** represents that how many rows you want to select from bottom. If we does not pass any value then by default it select 5 rows.

3) DataFrameobj.describe():

=> describe() gives the statistical information about dataframe . Such as count,mean,std,min max and percent..

4) DataFrameobj.shape:

=>shape attribute of dataframe object give the shape of Dataframe in form of Row X column.

5) DataFrameobj [start:stop:step]:

=>using this we can access the number of rows as we desire.

6) DataFrameobj["Col Name"] :

=>with this we can access the one column information by suppling the column name.

7) DataFrameobj[["Col Name1","Col Name-2"...."Col Name-n"]] :

=>with this we can access the n number of columns information by suppling the columns name.

8) DataFrameobj[["Col Name1","Col Name-2"...."Col Name-n"]] [start:stop:step]
=>with this we can access the n number of columns information by suppling the
columns name wrt how many row we want.

9) DataFrameobj.iterrows():

=>If we use iterrows() wrt DataFrame object then it give hexdecimal address of
dataframe object.

Example-1:

```
>>> p.iterrows()-----<generator object DataFrame.iterrows at  
0x000002382938CB30>
```

=>iterrows() gives the total information about one row when we use
DataFrameobj.iterrows() in for loop.

Example-2:

```
p=pd.DataFrame([[10,"pooja",56,68,90],[20,"ritesh",68,79,78],[30,"shubham",87,56,80]],  
index=[1,2,3],columns=["roll_no","name","python","math","datasci"])
```

```
>>> p-----
```

	roll_no	name	python	math	datasci
1	10	pooja	56	68	90
2	20	ritesh	68	79	78
3	30	shubham	87	56	80

```
>>> res=p.iterrows()
```

```
>>> for i in res:  
...     print(i)
```

```
...
```

OUTPUT: -----

```
(1, roll_no    10  
   name      pooja  
   python     56  
   math       68  
   datasci    90  
   Name: 1, dtype: object)  
(2, roll_no    20  
   name      ritesh  
   python     68  
   math       79  
   datasci    78  
   Name: 2, dtype: object)  
(3, roll_no    30  
   name      shubham  
   python     87  
   math       56  
   datasci    80  
   Name: 3, dtype: object)
```

Understanding 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"]
-

Understanding 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) dataframename['new col name']=default value
 - 2) dataframename['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) dataframename.sort_values("colname")
 - 2) dataframename.sort_values("colname",ascending=False)
 - 3) dataframename.sort_values(["colname1","col name2",...col name-n])
-

knowing duplicates in dataframe data:

- 1) dataframename.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)]`
-

Network Programming

=>The purpose of Network Programming is that "To share the data between multiple machines".

=>Def. of Network:

=>A Network is a collection of Inter-connected computers connected with Server".

=>In Network programming, we have two write two types of Programs. They are

- 1) Server Side Program
- 2) Client Side Program

1) Server Side Program:

=>A Server Side Program is of the Python Program, which receives the request from client side program, Process the Request and gives response back to Client Side Program.

2) Client Side Program:

=>A Client Side Program is of the Python Program , which always send request to the server side program and receives the response from server side Program

3) DNS(Domain Naming Service)

=>A DNS is a Name of the Physical machine where Server Side Programs resides

=>The default DNS of every computer is "localhost"

4) IP Address (Internet Protocol address)

=>An IP Address is one of the Numerical Address of the Physical Machine, where server side
program resides.

=>The default IP Address of every machine is 127.0.0.1 (loop back address)

5) port number:

=>A Port Number is a logical numerical id where server side program is running.

Steps for developing Server Side Program

- 1) import socket module
- 2) Every Server Side Program must run at Certain DNS / IP Address and Port Number.
- 3) Server Program must ACCEPT the Client Side Program Request.
- 4) Server Side Program must READ the data coming from client side program.
- 5) Server Side Program must PROCESS the the Data coming from client side program.
- 6) Server Side Program must Send RESPONSE back to client side program.

Note: As long as Client Side Program sending requests to the server side program, The Server Side Program performs Step-(3), Step-(4) , Step-(5) and Step-(6)

Steps for developing Client Side Program

- 1) import socket module
 - 2) Every Client Side Program must get the connection from Server Side Program by passing
 DNS or IP Address and Port Number.
 - 3) Client Side Program must SEND request to the Server Side Program
 - 4) Client Side Program must RECEIVE the response from Server Side Program.
-

socket module

=>To deal with network programming, we must use a pre-defined module called "socket".
=>Socket Module contains Variables , Function,class names.

1) socket()

=>This function is used for creating an object of socket.
=>An object of Socket can be created both at client and server side programs and an object of socket acts as a Bi-Directional Communication object between Server Side and Client Program.

Syntax: varname=socket.socket()

here varname is an object of <class, socket.Socket>

Examples: s=socket.socket()

2) bind()

=>This function is used for binding the server side program at certain DNS and port number.

=>Syntax: socketobj.bind("DNS/IPAddress", portno)

=>Examples : s.bind("localhost",8888)

(OR)

s.bind("127.0.0.1",8888)

3) listen():

=>This Function is used for configuring Server Side Program in such way that to how many clients The Server Side Program can communicate.

=>Syntax: socketobj.listen(number of clients)

=>Examples: s.listen(2)

4) accept():

=>This function is used for accepting the client side program request.

=>This function returns two objects(clientconobj, clientaddress)

=>Syntax: clientsocketobject,clientaddr=socketobj.accept()

=>Examples: clientsocketobj,caddr=s.accept()

5) send() with encode():

>This function is used for sending The response of Server side program to the client side program and Client Side program send request to Server Side Program in the form of Encrypted format (bytes or bytearray) by calling encode()

=>Syntax1: clientsocketobject.send(str(data).encode()) # At server side program

=>Syntax2: clientsocketobject.send(str(data).encode()) # At client code

6) recv() with decode():

=>This function is used for receiving the the request coming from Client Side Program and

and used for receiving the response from Server Side program at client side program.

=>Syntax1:- clientsocketobj.recv(1024 / 2048 / 4096....).decode() # At server side

=>Syntax2:- clientsocketobj.recv(1024 / 2048 / 4096....).decode() # At client Side

7) connect():

=>This Function is used for obtaining connection from server side program at client side program.

=>Syntax:- clientsocketobject.connect("DNS/IPAddress", porno)

=>Examples: s=socket.socket()

```
s.connect( "localhost",8888)
(OR)
s.connect(("127.0.0.1"),8888)
=====x=====
```

EXAMPLE1:

```
#Program for Squaring of the given number coming from client side program
#ServerSquare.py
import socket
s=socket.socket()
s.bind( ("localhost",8888) )
s.listen(2)
print("Server Side Program is ready to accept any client request:")
while(True):
    try:
        clientobj,clientaddr=s.accept()
        cdata=clientobj.recv(1024).decode()
        print("Val of Client at Server={}".format(cdata))
        val=float(cdata)
        result=val**2
        clientobj.send(str(result).encode())
    except ValueError:
        clientobj.send("Don't enter strs, symbols, alpha-numerics".encode() )
```

```
#Program for sending a number to server side program and obtain Square of the
given number coming from Server side program at client side program
#ClientSquare.py
import socket
s=socket.socket()
s.connect( ("localhost",8888) )
print("Client Side Program got connection from Server Side Program")
n=input("Enter a number:")
s.send(n.encode())
sres=s.recv(1024).decode()
print("Result from Server at client={}".format(sres))
```

EXAMPLE2:

```

#ChatServer.py
import socket
#step-1
s=socket.socket()
s.bind(("localhost",11111))
s.listen(2)
print("-"*50)
print("SSP is ready to accept any CSP request")
print("-"*50)
while(True):
    #step-2
    con,adr=s.accept()
    #step-3
    cdata=con.recv(1024).decode()
    print("Client Data at Server->{}".format(cdata))
    sdata=input("SSP Message to Client:")
    #step-4
    con.send(sdata.encode())

```

```

#chatclient.py
import socket
#step-1
s=socket.socket()
#step-2
s.connect(("localhost",11111))
print("-"*50)
print("CSP got connected to SSP")
print("-"*50)
#step-3
cdata=input("Enter Client Message:")
s.send(cdata.encode())
#step-4
sdata=s.recv(1024).decode()
print("SSP data at CSP-->{}".format(sdata))

```

EXAMPLE3:

```

#This program receives Emp number from client side program , connect employee
table , reading other details of employee and send to client side program.
#EmpDataServer.py---Program-(A)
import socket
import cx_Oracle
#step-1
s=socket.socket()
s.bind( ("127.0.0.1",8888) )
s.listen(2)
print("SSP is ready to accept any CSP Request:")
#step-2
while(True):

```

```

try:
    conn,addr=s.accept()
    eno=int(conn.recv(1024).decode())
    #PDBC Code
    oracon=cx_Oracle.connect("scott/tiger@localhost/orcl")
    cur=oracon.cursor()
    cur.execute("select ename,sal,job from emp where empno=%d"
%eno)
    emprec=cur.fetchone()
    if(emprec==None):
        conn.send("Employee Record Does not Exists:".encode())
    else:
        conn.send(str(emprec).encode())
except ValueError:
    conn.send("Don't Enter strs/symbols/alpha-numerics".encode())
except cx_Oracle.DatabaseError as db:
    conn.send("Problem in Database:"+str(db).encode())

```

```

#This program receives Emp Name, Sal,Designation and Comp Name from Server
side program by connecting to employee table
#EmpDataClient.py---Program-(A)
import socket
while(True):
    s=socket.socket()
    s.connect(("127.0.0.1",8888))
    print("CSP Connected to SSP:")
    eno=input("\nEnter Employee Number :")
    s.send(eno.encode())
    emprec=s.recv(1024).decode()
    print("Employee data: {}".format(emprec))
    ch=input("\nDo u want to continue(yes/no):")
    if(ch=="no"):
        print("Thanks for using this program:")
        break
=====
```

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 following 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.randrange(Value)`
 This syntax generates any random value between 0 to Value-1

Syntax-2: `random.randrange(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.randint(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("-----")
=====
```

c) random()

=>Syntax:- random.random()
=>This syntax generates floating point random values between 0.0 and 1.0 (Exclusive)

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))
=====
```

d) uniform()

Syntax:- random.uniform(start,stop)
=>This generates random floating 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))
=====
=====X=====
```

