Python Notes

==

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 (**Centrum Wiskunde & Informatica**) in Nether lands.

=>The Python programming Language maintained by non-commercial Organization Python Software Foundation (PSF).

=>Guido van Rossum was also reading the published scripts from "Monty Python's Flying Circus", a BBC comedy series from the 1970s. Van Rossum thought he needed a name that was short, unique, and slightly mysterious, so he decided to call the language Python.

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

==

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

==

Python Programming Inspired From

==

=> Functional Programming From C

=> Object Oriented Programming From CPP

=>Modular programming from Modulo3

=>Scripting Programming from PERL

==

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, matpoltlib...etc)

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

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

1. Simple

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

=>Python is one of the SIMPLE Programming Language, because 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.

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

==

2. Freeware and Open Source

==

=>Freeware:

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

=>Open Source:

=>The standard python’s 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) Anaconda Python-->It used develop Big Data / Hadoop Based Applications.

6) Stackless Python--->Concurrency

7) PyPy---->It provides JIT(Just -in Time) compiler

.................etc

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

3. 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 depend 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 Types OSes but they can store single value and unable to store multiple values. whereas 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 UN-LIMITED size.

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

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

===

4. 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 because 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============================

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

5. 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:

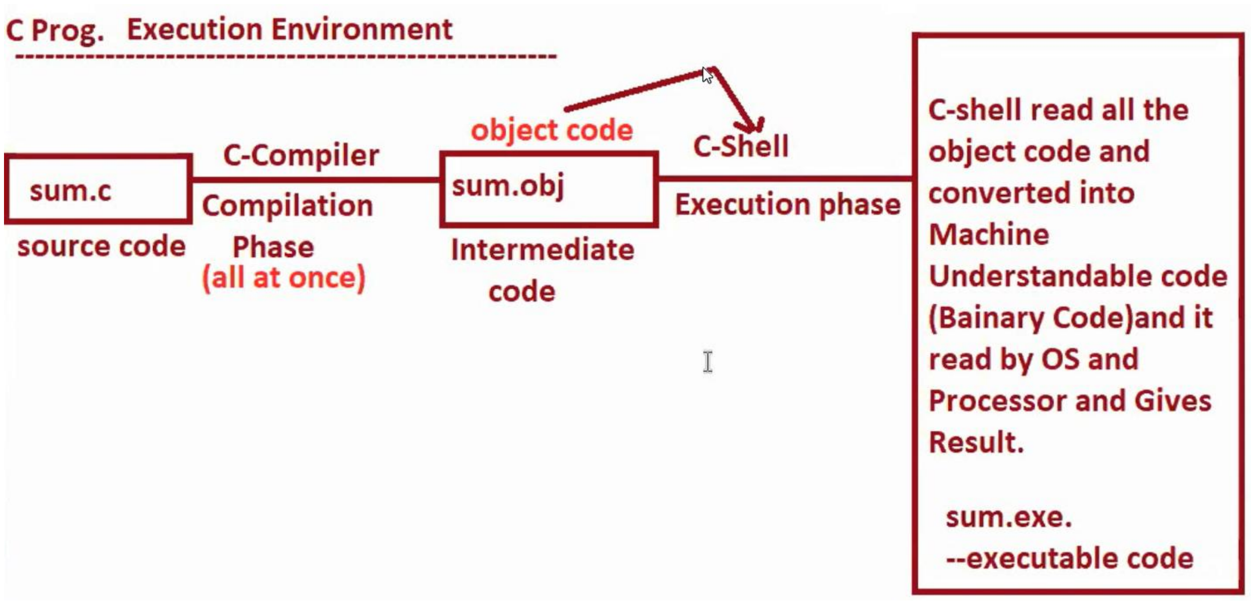
=>The PVM (Python Virtual Machine) reads Line by Line of Byte Code (.pyc) and converted into Machine Understandable Code and It is read by OS and Processor and Gives Final Result of the Program

=>Since this execution performed by PVM Line by Line and hence It is Interpreted Programming.

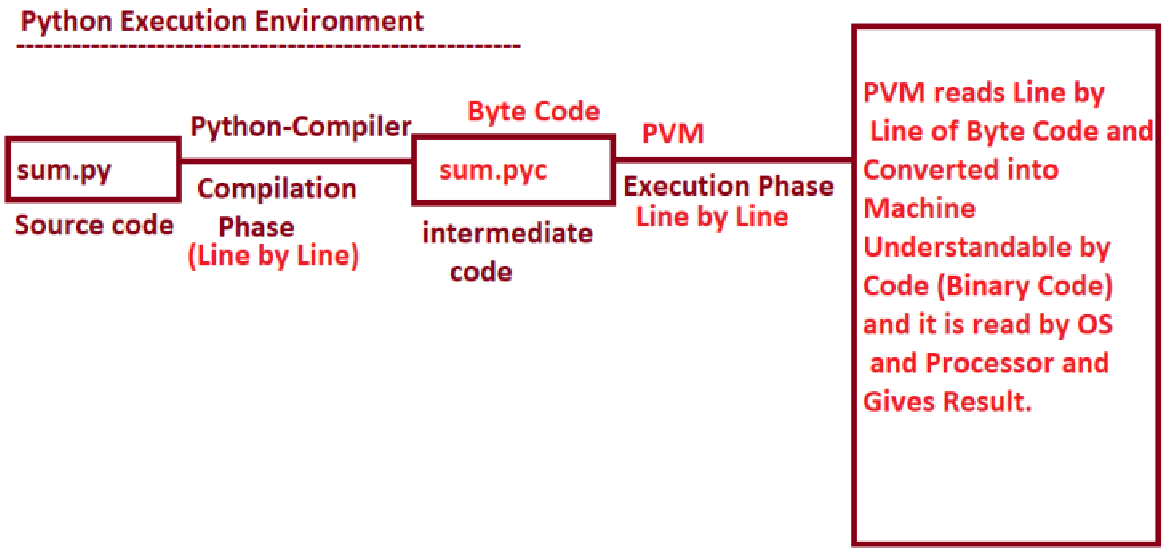
Examples:-

sum.pyc----->PVM (Line by Line)---->Machine Code--->OS+Processor-->Result

=>When we execute Python Program Python Compilation execution performed By Compiler and PVM Line by Line and Hence Python is one of Interpreted Programming language.



C Prog. Execution Environment
object code
C-Shell
sum.c
C-Compiler
Compilation Phase
(all at once)
source code
sum.obj
Intermediate code
Execution phase
C-shell read all the object code and converted into Machine Understandable code (Bainary Code)and it read by OS and Processor and Gives Result.
sum.exe.
--executable code



Python Execution Environment
Python-Compiler
Byte Code
PVM
sum.py
Source code
Compilation Phase
(Line by Line)
sum.pyc
intermediate code
Execution Phase
Line by Line
PVM reads Line by Line of Byte Code and Converted into Machine Understandable by Code (Binary Code) and it is read by OS and Processor and Gives Result.

==

6. 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.

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

==

7. 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---

===

8. Extensible and 9. 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 cab 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================================

==

10. 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 Performance with precise code.

NOTE:

=>First Party---Lang Library (API)---Python In-built 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(tools)

===

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 3 types of Literals. They are

a) Numeric Literals

i) Integers

ii) floats

b) Boolean Literals

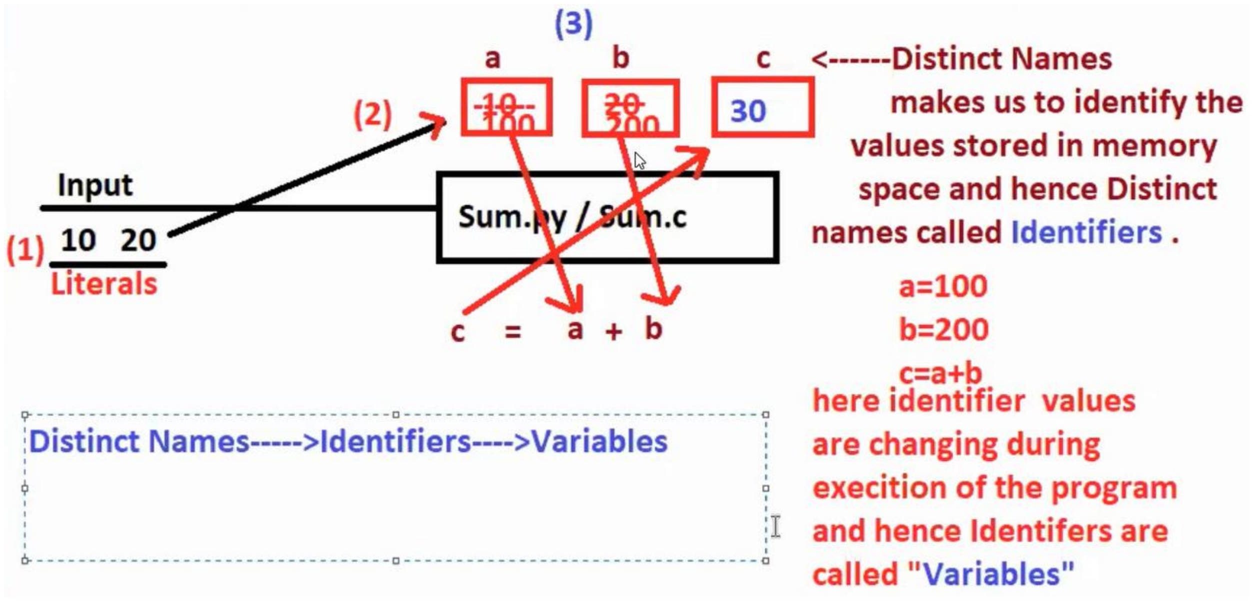
i) True

ii) False

c) String Literals

Examples: "Python"

"Guido Van Rossum"


(3)
a
b
c
<------Distinct Names makes us to identify the values stored in memory space and hence Distinct names called Identifiers .
(2)
~~10~~ 100
~~20~~ 200
30
Input
Sum.py / Sum.c
(1) 10 20
Literals
c = a + b
a=100
b=200
c=a+b
here identifier values are changing during excecition of the program and hence Identifers are called "Variables"
Distinct Names----->Identifiers---->Variables

==

Importance of Identifiers (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.

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

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 start 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. because 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 Variables 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

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

a) int

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

=>'int' is one of the pre-defined classes 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.

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.

Examples:

>>> a=0xAC

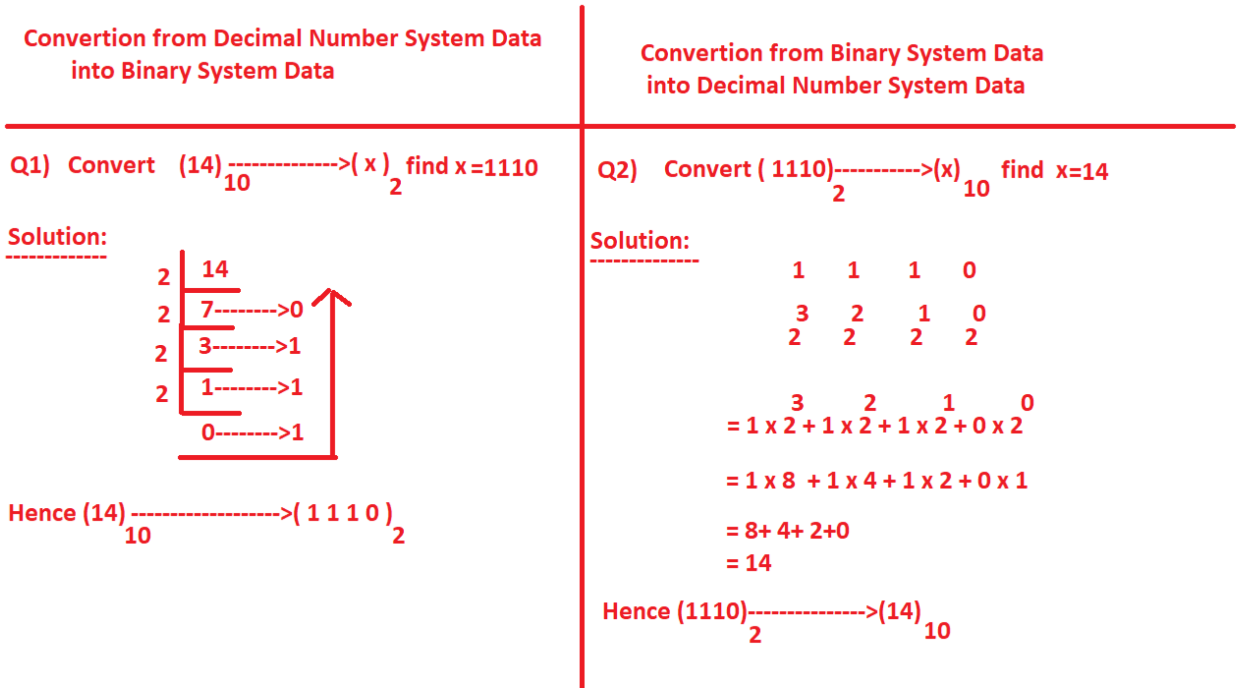
>>> print(a,type(a))---------172 <class 'int'>

>>> 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
```

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

1. Conversion from Decimal to Binary and Binary to Decimal



Convertion from Decimal Number System Data into Binary System Data

Q1) Convert $(14)_{10}$ --------------->$(x)_2$ find x =1110

Solution:

```
2 | 14
2 | 7--------->0
2 | 3--------->1
2 | 1--------->1
    0--------->1
```

Hence $(14)_{10}$ ------------------->$(1\,1\,1\,0)_2$

Convertion from Binary System Data into Decimal Number System Data

Q2) Convert $(1110)_2$ ----------->$(x)_{10}$ find x=14

Solution:

$$\begin{matrix} 1 & 1 & 1 & 0 \\ 2^3 & 2^2 & 2^1 & 2^0 \end{matrix}$$

$= 1 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0$

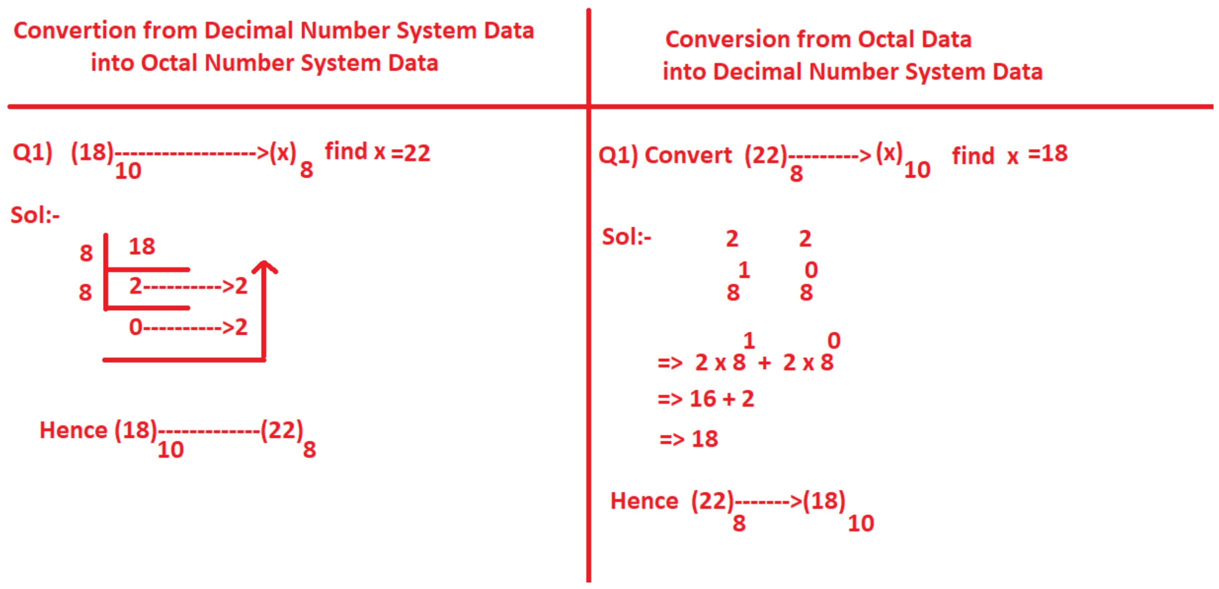
$= 1 \times 8 + 1 \times 4 + 1 \times 2 + 0 \times 1$

$= 8+ 4+ 2+0$

$= 14$

Hence $(1110)_2$--------------->$(14)_{10}$

2. Conversion from Decimal to Octal and Octal to Decimal



Convertion from Decimal Number System Data into Octal Number System Data

Q1) $(18)_{10}$------------------>$(x)_8$ find x =22

Sol:-

$$\begin{array}{r|l} 8 & 18 \\ \hline 8 & 2 \text{----------}> 2 \\ \hline & 0 \text{----------}> 2 \end{array}$$

Hence $(18)_{10}$-------------$(22)_8$

Conversion from Octal Data into Decimal Number System Data

Q1) Convert $(22)_8$---------> $(x)_{10}$ find x =18

Sol:- $2 \; 2$ $8^1 \; 8^0$

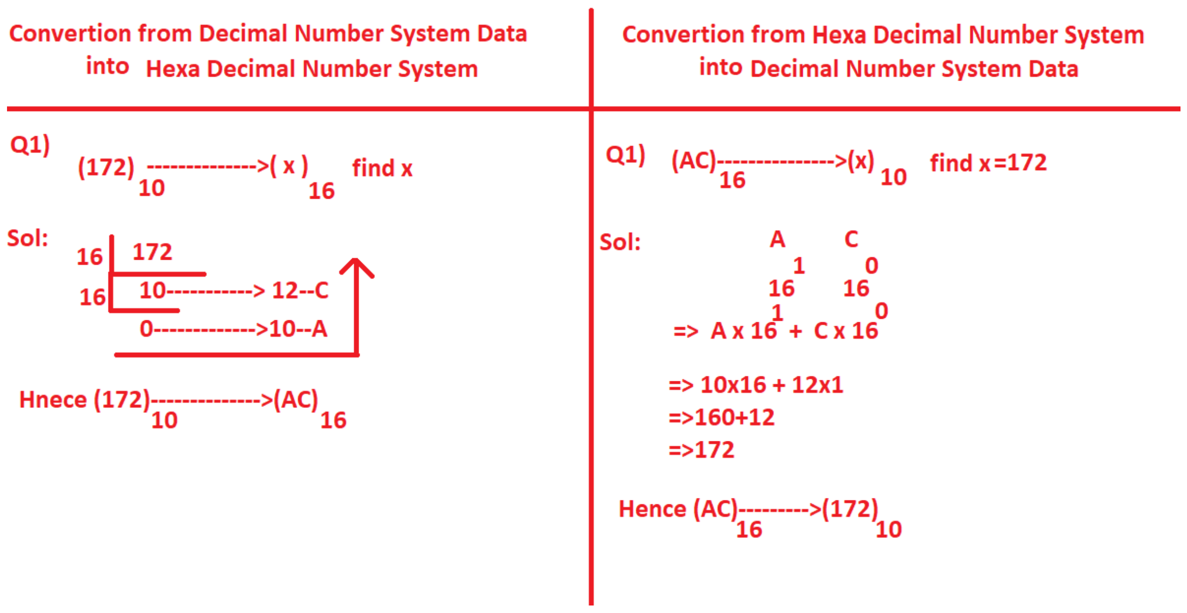
$\Rightarrow 2 \times 8^1 + 2 \times 8^0$

$\Rightarrow 16 + 2$

$\Rightarrow 18$

Hence $(22)_8$------->$(18)_{10}$

3. Conversion from Decimal to Hexa and Hexa to Decimal



Convertion from Decimal Number System Data into Hexa Decimal Number System

Q1) $(172)_{10}$ -------------->$(x)_{16}$ find x

Sol:

$$\begin{array}{r|l} 16 & 172 \\ \hline 16 & 10 \text{-----------}> 12\text{--C} \\ \hline & 0 \text{------------}> 10\text{--A} \end{array}$$

Hnece $(172)_{10}$-------------->$(AC)_{16}$

Convertion from Hexa Decimal Number System into Decimal Number System Data

Q1) $(AC)_{16}$--------------->$(x)_{10}$ find x=172

Sol: $A \; C$ $16^1 \; 16^0$

$\Rightarrow A \times 16^1 + C \times 16^0$

$\Rightarrow 10 \times 16 + 12 \times 1$

$\Rightarrow 160+12$

$\Rightarrow 172$

Hence $(AC)_{16}$--------->$(172)_{10}$

==

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)------------0b1001100111111101010111110000010001
```

----

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(0b10100000011110001111000011)
>>> 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(0b1111111111000000000111111111)
>>> 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'>
```

--

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

b) 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, 0.99 ....etc

-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:

```
>>> a=4e2
>>> print(a,type(a))----------400.0 <class 'float'>
>>> a=10e2
>>> print(a,type(a))----------1000.0 <class 'float'>
>>> a=10e-2
>>> print(a,type(a))-------0.1 <class 'float'>
>>> a=3.4e5
>>> print(a,type(a))--------340000.0 <class 'float'>
>>> a=0.0000000000000000000000000000000000000000000000002
>>> print(a,type(a))--->   2e-49 <class 'float'>
------------------------------------------
>>> a=0b1010.0b1010------SyntaxError: invalid decimal literal
>>> b=0o12.0o34------------SyntaxError: invalid decimal literal
>>> c=0xA.0xF-----------SyntaxError: invalid decimal literal
```

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

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

c) 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 is takes as 0

Examples:

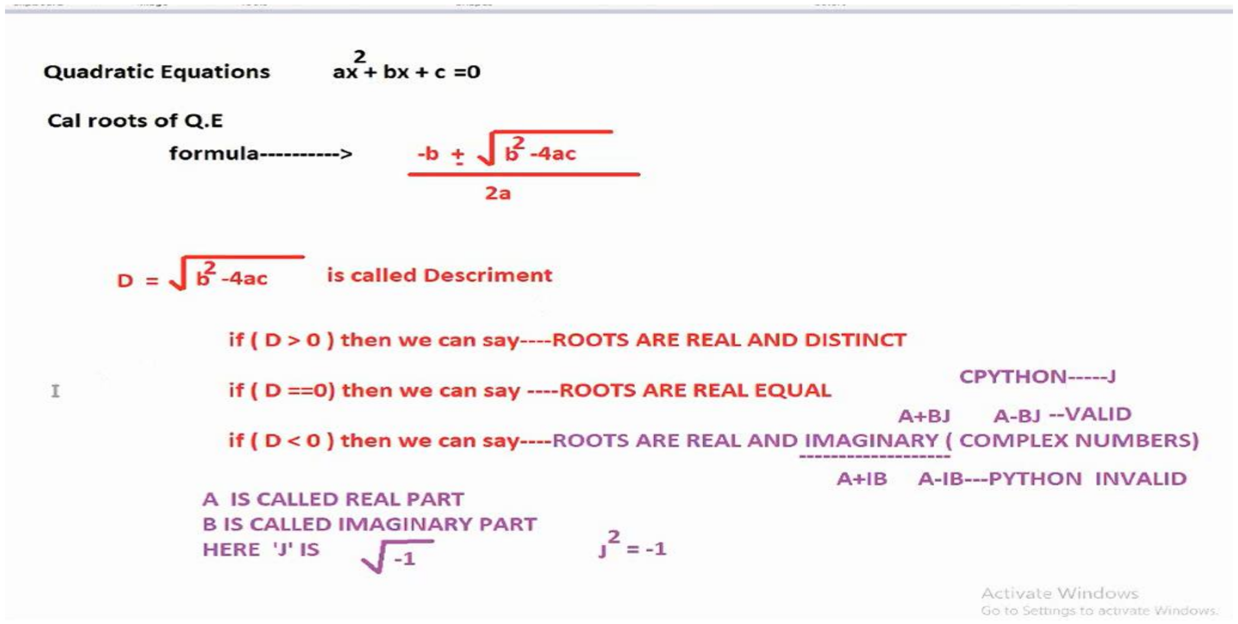
```
>>> a=True
>>> print(a,type(a))-------True <class 'bool'>
>>> b=False
>>> print(b,type(b))-------False <class 'bool'>
>>>
>>> print(True+True)-------------2
>>> print(True*True)------------1
>>> print(True+False)-----------1
>>> print(True*False)-----------0
>>> print(True+False*False)----1
>>> print(2*True+False)---2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

==

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

d) complex

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



Quadratic Equations $ax^2 + bx + c = 0$

Cal roots of Q.E

formula----------> $\frac{-b \pm \sqrt{b^2 -4ac}}{2a}$

D = $\sqrt{b^2 -4ac}$ is called Descriment

if (D > 0) then we can say----ROOTS ARE REAL AND DISTINCT

if (D ==0) then we can say ----ROOTS ARE REAL EQUAL

if (D < 0) then we can say----ROOTS ARE REAL AND IMAGINARY (COMPLEX NUMBERS)

CPYTHON-----J

A+BJ A-BJ --VALID

A+IB A-IB---PYTHON INVALID

A IS CALLED REAL PART

B IS CALLED IMAGINARY PART

HERE 'J' IS $\sqrt{-1}$ $J^2 = -1$

=>'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 below.

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-dedined 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 no 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

==

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

i) 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 classes 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/ store 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 otherworld's, 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="Gudido van Rossum
                SyntaxError: unterminated string literal (detected at line 1)
>>> addr1='Gudido 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

==

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

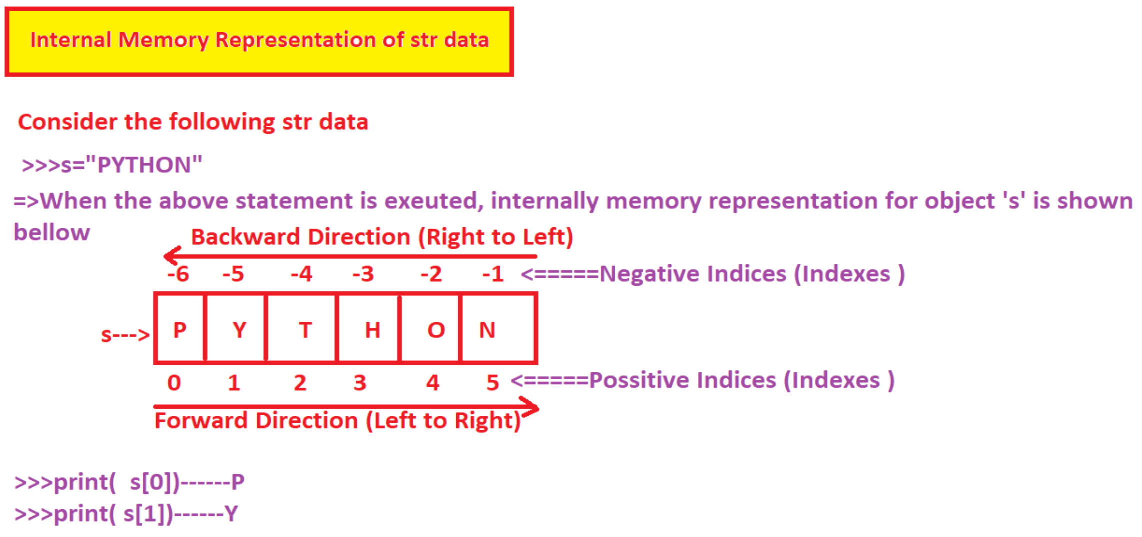
Operations on str data

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

=>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 exeuted, internally memory representation for object 's' is shown bellow

Backward Direction (Right to Left)
-6 -5 -4 -3 -2 -1 <=====Negative Indices (Indexes)
s--->
P Y T H O N
0 1 2 3 4 5 <=====Possitive Indices (Indexes)
Forward Direction (Left to Right)

```
>>>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 Positive 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
```

----

2) Slicing:

=>The process of obtaining the range of characters / sub string of main string is called Slicing.

=>The slicing operation contains the following Syntaxes:

=>Syntax-1:   strobj[Begin : End]

=>This syntax obtains range of characters from "Begin" Index to "End" Index-1 always provided **(Begin Index<End Index)** otherwise we never get any output (empty output)

=>Examples:

```
>>> s="PYTHON"
>>> s[2:4]------------------'TH'
>>> print(s[2:4])---------TH
>>> print(s[1:6])--------YTHON
>>> print(s[3:6])--------HON
>>> print(s[13:6])------ No Out put 14<6--False
>>> s[13:6]----------'  '  empty output
>>> s="PYTHON"
>>> print(s[-4:-2])-------------TH
>>> print(s[-6:-1])------------PYTHO
>>> print(s[-3:-1])-----------HO
>>> print(s[-1:-3])-------No output  because -1<-3 is False
>>> s[-1:-3]------------' '   empty output  -1<=3 is False
```

==

=>Syntax-2:   strobj[Begin :]

=>In this syntax, we have Begin Index and End Index is not specified

=>Since Programmer did't specify the end index and hence End Index considered by PVM as len(strobj)-1

=>This syntax gives range of characters from Begin Index to Last Character of the Str obj **(end**

index=(len(strobj)-1)

Examples:

```
>>> s="PYTHON"
>>> print(s[2:])-----------THON
>>> print(s[1:])----------YTHON
>>> print(s[4:])-------ON
>>> print(s[-2:])---------ON
>>> print(s[-6:])-------PYTHON
>>> print(s[0:])--------PYTHON
>>> print(s[-2:])-------ON
>>> print(s[-5:])-------YTHON
```

--

=>Syntax-3:   strobj[: End]

=>In this syntax, we have not specified Begin Index and End Index is specified

=>Since Programmer didn't specify the Begin index and hence Begin Index considered by PVM as Initial Index.

=>This syntax gives range of characters from Initial Index to End Index-1 of the Str obj

----

Examples:

```
>>> s="PYTHON"
>>> print(s[ : 5] )------------PYTHO
>>> print(s[ : 5])------------PYTHO
>>> print(s[ : 2])-----------PY
>>> print(s[ : 3])----------PYT
>>> print(s[ :-3])---------PYT
>>> print(s[ :-4])--------PY
```

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

--

=>Syntax-4: strobj[:]

=>In this syntax, we didn’t specify begin index and end index.

=>If we don't specify Begin Index then PVM takes Begin Index as Initial Index

=>If we don't specify End Index then PVM takes End Index as Ien(strobj)-1

=>This Syntax gives complete data of strobj.

Examples:

>>> s="PYTHON"

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

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

>>> s="Python Programming"

>>> print(s[5:7])----------n

>>> print(s[5:8])------n P

>>> print(s[:])----------Python Programming

>>> s="Lap Top"

>>> print(s[:])----------Lap Top

>>> print(s)----------Lap Top

>>> print(s[0:])---------Lap Top

>>> print(s[:6])-----Lap To

>>> print(s[:7])------Lap Top

=>ALL THE ABOVE SYNTAXES GETS THE DATA IN FORWARD DIRECTION ONLY BUT NOT IN REVERSE / BACKWARD DIRECTION.

Syntax:-

----

strobj[BEGIN:END: STEP]

=>here STEP is optional to specify and default value of step is 1 and looks in Forward direction.

=>Rule-1:= Here BEGIN , END and STEP can be Positive or Negative

----

=>Rule-2:= if the STEP value is POSSITIVE then we have to consider the characters / elements from BEGIN Index to END Index-1 in FORWARD Direction(Left to Right) provided **BEGIN Index < End Index** otherwise we never get any output

----

=>Rule-3:= if the STEP value is NEGATIVE then we have to consider the characters / elements from BEGIN Index to END Index+1 in BACKWARD Direction(Right to Left) provided **BEGIN Index > End Index** otherwise we never get any output

----

=>**Rule-4:=** In FORWARD Direction, if the end value is 0 then we never get any output.

=>**Rule-5**:= In BACKWARD Direction, if the end value is -1 then we never get any output.

==

Examples:

>>> s="PYTHON"

>>> print(s[0:6:2])----------PTO

>>> print(s[0:6:3])---------PH

>>> print(s[2:6:3])---------TN

>>> print(s[-6:-3:2])-------PT

>>> print(s[:4:2])--------PT

>>> print(s[3: :2])----------HN

>>> print(s[: : 2])--------PTO

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

>>> s="PYTHON"

>>> print(s[2:6:-1])------- No output

>>> print(s[6:2:-1])--------NOH

>>> print(s[5:0:-2])-------NHY

>>> print(s[::-2])--------NHY

```
>>> print(s[::-1])----------NOHTYP
>>> print(s[::1])----------PYTHON
>>> s="KVR"
>>> print(s[::-1])-------RVK
>>> s="PYTHON PROGRAMMING"
>>> print(s)--------PYTHON PROGRAMMING
>>> print(s[::1])---------PYTHON PROGRAMMING
>>> print(s[::-1])-------GNIMMARGORP NOHTYP
>>> s="MADAM"
>>> print(s[::1])--------MADAM
>>> print(s[::-1])-------MADAM
>>> s="MALAYALAM"
>>> print(s[::1])--------MALAYALAM
>>> print(s[::-11])--------M
>>> s="LIRIL"
>>> print(s[::1])--------LIRIL
>>> print(s[::-1])---------LIRIL
>>> s="NAYAN"
>>> print(s[::1])------NAYAN
>>> print(s[::-1])----------NAYAN
>>> print(s[:0:1])---No Output----Rule-4
>>> print(s[:-1:-1])-----No Output----Rule-5
```

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

Mutable and Immutable concept

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

Mutable:

=>A Mutable Object is one which allows us to perform the changes / updations at the same Address.

Examples:- **list set dict**

Immutable:

=>An Immutable Object is one which satisfies the following Properties.

a) It NEVER allows us to perform the the changes / updations at the same Address.(If at all any changes / updation are allowed then modified values are placed in different address)

b) It NEVER allows us to perform ITEM ASSIGNMENT

Examples:-

int, float, str, tuple, set...etc

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

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

b) bytes

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

=>'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 Positive Integer Values ranges from 0 to 256. i.e It stores Numerical Positive 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 because '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==================================
```

===

bytearray

===

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

=>The purpose of bytearray 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.

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

Syntax: varname=bytearray(object)

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

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

Note:- The Functionality of bytearray is exactly similar to bytes data type but an object of bytearray 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=bytearray(l)
>>> print(ba,type(ba),id(ba))---bytearray(b'\n\x14\x1e-CY') <class 'bytearray'>
                                                      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 intervalue 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)

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

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
>>>
```

==

==

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 specfied 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 Occurence 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 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
```

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

Types of Copy Techniques in Python

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

=>In Python Programming, we have two types of Copy Techniques. They are

a) Shallow Copy

b) Deep Copy

a) Shallow Copy:

=>In Shallow Copy

a) Initial Content of Both the objects are Same.

b) Memory Address(id's) of Both the Objects are different.

c) The Modifications are Independent (Whatever the changes we do one object they are not reflecting on other object).

=>To Implement Shallow Copy, we use copy()

=>Syntax:- object2=object1.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
```

----

b) Deep Copy:

=>In Deep Copy

a) Initial Content of Both the objects are Same.

b) Memory Address (id's)of Both the Objects are Same.

c) The Modifications are dependent (Whatever the changes we do one object they are reflecting on another object).

=>To Implement Deep Copy, we use assignment operator (=)

=>Syntax:-   object2=object1

Examnples:

```
>>> l1=[10,"Rossum"]
>>> print(l1,id(l1))-----------[10, 'Rossum'] 1957574950720
>>> l2=l1 # Deep Copy
>>> print(l2,id(l2))----------[10, 'Rossum'] 1957574950720
>>> l1.append("Python")
>>> print(l1,id(l1))--------[10, 'Rossum', 'Python'] 1957574950720
>>> print(l2,id(l2))-----------[10, 'Rossum', 'Python'] 1957574950720
>>> l2.append("DSC")
>>> print(l1,id(l1))---------[10, 'Rossum', 'Python', 'DSC'] 1957574950720
>>> print(l2,id(l2))------[10, 'Rossum', 'Python', 'DSC'] 1957574950720
```

==

10) reverse() :

=>This Function is used for obtaining reverse of list object(back to front and front to back)

=>Syntax: listobj.reverse()

Examples:

```
>>> lst=[10,"Rossum",55.55,"Python","Hyd"]
>>> print(lst)-----[10, 'Rossum', 55.55, 'Python', 'Hyd']
>>> lst.reverse()
>>> print(lst)-----['Hyd', 'Python', 55.55, 'Rossum', 10]
>>> lst.reverse()
>>> print(lst)-----[10, 'Rossum', 55.55, 'Python', 'Hyd']
```

----

11) sort() :

=>This function is used for Sorting the given data in the form of AScending Order.

=>Syntax: listobj.sort()

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

=>Here reverse=False represents sort the given in Ascending Order

=>Here reverse=True represents sort the given in Decending Order

Note: writing listobj.sort(reverse=False) is as good as listobj.sort()

Examples:

```
   >>> lst=[10,3,45,23,-45,8,9,1,67,3,0]
>>> print(lst)---------[10, 3, 45, 23, -45, 8, 9, 1, 67, 3, 0]
>>> lst.sort()
>>> print(lst)----------[-45, 0, 1, 3, 3, 8, 9, 10, 23, 45, 67]
>>> lst.reverse()
>>> print(lst)-------------[67, 45, 23, 10, 9, 8, 3, 3, 1, 0, -45]
>>>
>>> lst=["kiwi","banana","guava","apple","sberry"]
>>> print(lst)----------['kiwi', 'banana', 'guava', 'apple', 'sberry']
>>> lst.sort()
>>> print(lst)---------['apple', 'banana', 'guava', 'kiwi', 'sberry']
>>> lst.reverse()
>>> print(lst)----------['sberry', 'kiwi', 'guava', 'banana', 'apple']
---------------------------------------------------------------------------
>>> lst=[10,3,45,23,-45,8,9,1,67,3,0]
>>> print(lst)-----------[10, 3, 45, 23, -45, 8, 9, 1, 67, 3, 0]
>>> lst.sort(reverse=True)
>>> print(lst)-----------[67, 45, 23, 10, 9, 8, 3, 3, 1, 0, -45]
>>> lst=[10,3,45,23,-45,8,9,1,67,3,0]
>>> print(lst)----------[10, 3, 45, 23, -45, 8, 9, 1, 67, 3, 0]
>>> lst.sort(reverse=False)
>>> print(lst)---------[-45, 0, 1, 3, 3, 8, 9, 10, 23, 45, 67]
```

----

12) extend()

=>This function is used for extending the functionality of one list object to another list object.

=>Syntax:- listobj1.extend(listobj2)

=>Here the elements of listobj2 is added at the end of listobj1

Examples:

```
>>> lst1=[10,20,30,40]
>>> lst2=["Python","Java","CPP"]
>>> lst1.extend(lst2)
>>> print(lst1)----------[10, 20, 30, 40, 'Python', 'Java', 'CPP']
>>> lst1=[10,20,30,40]
>>> lst2=["Python","Java","CPP"]
>>> lst3=["DS","Django","Numpy","Pandas"]
>>> print(lst1)---------[10, 20, 30, 40]
>>> print(lst2)--------['Python', 'Java', 'CPP']

>>> print(lst3)--------['DS', 'Django', 'Numpy', 'Pandas']

>>> lst1.extend(lst2,lst3)--TypeError: list.extend() takes exactly one

>>> lst1.extend(lst2)

>>> print(lst1)--------[10, 20, 30, 40, 'Python', 'Java', 'CPP']

>>> lst1.extend(lst3)

>>> print(lst1)--[10, 20, 30, 40, 'Python', 'Java', 'CPP', 'DS', 'Django', 'Numpy',
                                          'Pandas']
```

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

>>> lst2=["Python","Java","CPP"]

>>> lst3=["DS","Django","Numpy","Pandas"]

>>> lst1=lst1+lst2+lst3

>>> print(lst1)--[10, 20, 30, 40, 'Python', 'Java', 'CPP', 'DS', 'Django', 'Numpy',
                          'Pandas']
```

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

Inner list (or) nested list

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

My requirement

----

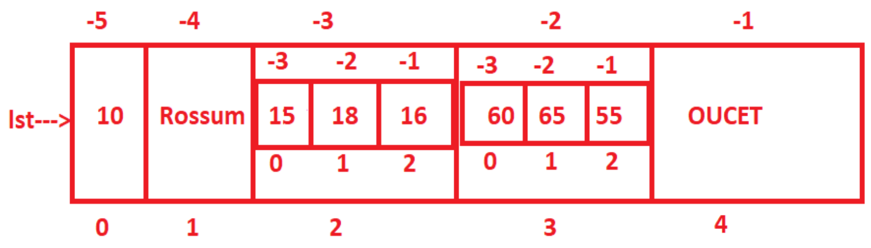
i want to store

stno , name , internal marks(sub1,sub2,sub3), Ext Marks(sub1,sub2,sub3),

collegne name

Memory Management of Inner list

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



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

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

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

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

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]]
```

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

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

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 oftuple , 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()

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

===

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 otherwards, 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 because set object never maintains 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'>
```

2190683253088

```
>>> 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 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}
                                          <class 'set'> 2190683253088
>>> 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'>
                                                                  2190683251968
>>> s2=s1  # Deep Copy
>>> print(s2,type(s2),id(s2))--------{'Python', 10, 'Rossum', 45.67} <class 'set'>
                                                  2190683251968
>>> s1.add("Django")
>>> print(s1,type(s1),id(s1))---{'Python', 'Django', 'Rossum', 10, 45.67} <class 'set'>
                                                  2190683251968
>>> print(s2,type(s2),id(s2))----{'Python', 'Django', 'Rossum', 10, 45.67} <class 'set'>
                                                  2190683251968
```

----

4) remove() :

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

=>If the element does not exist 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 arbitrary 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

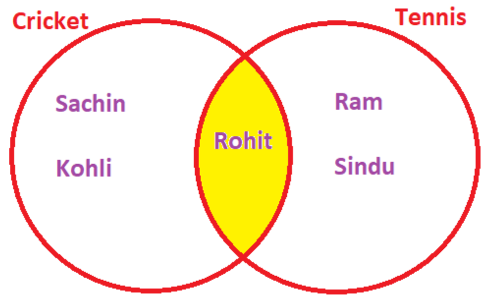
cricket players={"Sachin","Kohli","Rohit"}

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

Use Case: --Assigment

Let consider the following problem
cricket players={"Sachin","Kohli","Rohit"}
Tennis players={"Rohit","Ram","Sindu"}

Q1) Find all the payers 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()



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

==

Solving:

Q1) Find all the payers 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 payers 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'}
```

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

===

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 otherwards, 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 because 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 because an object of frozenset is purely Immutable.

a) add()

b) clear()

c) remove()

d) pop()

f) discard()

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

==

Dict Category Data Types (Collections Data Type)

==

=>we have one data type in Dict Category and whose name is 'dict'.

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

=>The purpose of dict data type is that "To store the data in the form (Key,Value)".

=>In (Key,Value) , the values of Key represents Unique and values of Value

represents Unique / Duplicates.

=>The (Key,value) must be organized / stored within curly braces { } .

=>An object of dict maintains Inserion Order

=>An object of dict never supports Indexing slicing

=>An object of dict belongs to mutable. In otherwords the values of Key is immutable where values of Value is mutable. The Values of Value can be updated based on Value of Key.

=>We can create two types of dict objects.

a) empty dict

b) non-empty dict

----

a) empty dict:

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

=>Syntax:-   dictobjname = { }

(or)

dictobjname = dict()

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

----

dictobjname[Key1]=Val1

dictobjname[Key2]=Val2

----

dictobjname[Key-n]=Val-n

=>Here the values of Key1 ,Key2...Key-n can be either Numeric (int) or Str

--

b) non-empty dict:

=>Empty dict is one which contains (Key,Value) and whose length is > 0.

=>Syntax:-   dictobjname = {Key1:Val1, Key2:Val2,....Key-n:Val-n }

--

Examples:

```
>>> d1={10:"RS",20:"JG",30:"RS",40:"DR"}
>>> print(d1,type(d1),id(d1))---{10: 'RS', 20: 'JG', 30: 'RS', 40: 'DR'} <class 'dict'>
                                              1520128442688
>>> d1[10]----------'RS'
>>> d1[20]----------'JG'
>>> d1[30]---------'RS'
>>> d1[40]-------'DR'
>>> d1[0]------KeyError: 0
>>> d1["RS"]-----KeyError: 'RS'
>>>
>>> d1[10]="Kavya"
>>> print(d1,type(d1),id(d1))----{10: 'Kavya', 20: 'JG', 30: 'RS', 40: 'DR'} <class 'dict'>
                                              1520128442688
>>> d1[20]="Priyanka"
>>> print(d1,type(d1),id(d1))---{10: 'Kavya', 20: 'Priyanka', 30: 'RS', 40: 'DR'}
                                                       <class 'dict'> 1520128442688
```

```
>>> d2={}
>>> print(d2,type(d2),id(d2))----------{} <class 'dict'> 1520128442944
>>> len(d2)----------0
>>> d3=dict()
>>> print(d3,type(d3),id(d3))-------{} <class 'dict'> 1520128756480
>>> len(d3)----0
>>> d2["Python"]=4000.00
>>> d2["Data Sci"]=15000.00
>>> d2["Django"]=5000
>>> print(d2,type(d2),id(d2))---{'Python': 4000.0, 'Data Sci': 15000.0, 'Django': 5000}
                                           <class 'dict'> 1520128442944
>>> len(d2)--------3
>>> d2["Java"]=2500
>>> print(d2,type(d2),id(d2))---{'Python': 4000.0, 'Data Sci': 15000.0, 'Django': 5000,
                                           'Java': 2500} <class 'dict'> 1520128442944
>>> d2["Python"]------4000.0
>>>
>>> d3=dict()
>>> print(d3,type(d3),id(d3))--------{} <class 'dict'> 1520128756928
>>> d3[10]="Apple"
>>> d3[20]="Mango"
>>> d3[30]="Banana"
>>> d3[40]="Kiwi"
>>> print(d3,type(d3),id(d3))---{10: 'Apple', 20: 'Mango', 30: 'Banana', 40: 'Kiwi'}
                                           <class 'dict'> 1520128756928


>>> d4=dict()
>>> print(d4,type(d4),id(d4))--------{} <class 'dict'> 1520128757120
>>> d4[10]="Apple"
>>> d4["Kiwi"]=30
>>> d4[20]=30
>>> d4[40]=30.99
>>> print(d4,type(d4),id(d4))----{10: 'Apple', 'Kiwi': 30, 20: 30, 40: 30.99}
                                    <class 'dict'> 1520128757120

>>> d1={10:"Apple",20:'Mango'}
>>> print(d1,type(d1),id(d1))----{10: 'Apple', 20: 'Mango'} <class 'dict'>
                              1520128756480
>>> d1[10]="Guava"
>>> print(d1,type(d1),id(d1))---{10: 'Guava', 20: 'Mango'} <class 'dict'>
                                    1520128756480
>>> d1[30]="Banana"
>>> print(d1,type(d1),id(d1))----{10: 'Guava', 20: 'Mango', 30: 'Banana'}
                                    <class 'dict'> 1520128756480
>>> d1={10:"Apple",20:'Mango'}
>>> print(d1,type(d1),id(d1))----{10: 'Apple', 20: 'Mango'} <class 'dict'>
                              1520128442688
>>> len(d1)------2
```

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

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

Pre-defined Functions in dict

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

=>In dict, we have the following pre-defined functions. They are

----

a) clear():

=>This Function is used for removing all the entries / elements of dict object.

=>Syntax: dictobj.clear()

Examples:

```
>>> d1={10:"Python",20:"Django",30:"Data Sci",40:"ML",50:"DL"}
>>> print(d1,id(d1))----{10: 'Python', 20: 'Django', 30: 'Data Sci', 40: 'ML', 50: 'DL'} 1520128756480
>>> len(d1)-----5
>>> d1.clear()
>>> print(d1,id(d1))---{} 1520128756480
>>> len(d1)---0
```

--

b) copy():

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

=> Syntax: dictobj2=dictobj1.copy()

Examples:

```
>>> d1={10:"Python",20:"Django",30:"Data Sci",40:"ML",50:"DL"}
>>> print(d1,id(d1))
{10: 'Python', 20: 'Django', 30: 'Data Sci', 40: 'ML', 50: 'DL'} 1520128442688
>>> d2=d1.copy()
>>> print(d2,id(d2))
{10: 'Python', 20: 'Django', 30: 'Data Sci', 40: 'ML', 50: 'DL'} 1520128756480
>>> d2[40]="Tensor-Flow"
>>> print(d2,id(d2))
{10: 'Python', 20: 'Django', 30: 'Data Sci', 40: 'Tensor-Flow', 50: 'DL'} 1520128756480
```

```
>>> print(d1,id(d1))
{10: 'Python', 20: 'Django', 30: 'Data Sci', 40: 'ML', 50: 'DL'} 1520128442688
```

---

c) pop():

Syntax:-   dictobj.pop(Key)

=>This Function is used for removing the entry of dict object by passing value of Key.

=>If the value of Key does not exists then we get KeyError.

=>Example:-

```
>>> d1={10:"Python",20:"Django",30:"Data Sci",40:"ML",50:"DL"}
>>> print(d1,type(d1))---{10: 'Python', 20: 'Django', 30: 'Data Sci', 40: 'ML', 50: 'DL'}
                                    <class 'dict'>
>>> d1.pop(20)-------'Django'
>>> print(d1,type(d1))---{10: 'Python', 30: 'Data Sci', 40: 'ML', 50: 'DL'} <class 'dict'>
>>> d1.pop(30)-----'Data Sci'
>>> print(d1,type(d1))-------{10: 'Python', 40: 'ML', 50: 'DL'} <class 'dict'>
>>> d1.pop(40)-----'ML'
>>> print(d1,type(d1))------{10: 'Python', 50: 'DL'} <class 'dict'>
>>> d1.pop(400)----KeyError: 400
```

---

d) popitem():

Syntax:-   dictobj.popitem()

=>This Function is used for removing last entry (Key,Value) of dict object.

=>If we call this function empty dict object then we get KeyError.

Examples:

```
----------------
>>> d1={10:"Python",20:"Django",30:"Data Sci",40:"ML",50:"DL"}
>>> print(d1,type(d1))---{10: 'Python', 20: 'Django', 30: 'Data Sci', 40: 'ML', 50: 'DL'}
                              <class 'dict'>
>>> d1.popitem()---------(50, 'DL')
>>> print(d1,type(d1))----{10: 'Python', 20: 'Django', 30: 'Data Sci', 40: 'ML'}
                                                            <class 'dict'>
>>> d1.popitem()-------(40, 'ML')
>>> print(d1,type(d1))-------{10: 'Python', 20: 'Django', 30: 'Data Sci'} <class 'dict'>
>>> d1.popitem()-----(30, 'Data Sci')
>>> print(d1,type(d1))------{10: 'Python', 20: 'Django'} <class 'dict'>
>>> d1.popitem()-----(20, 'Django')
>>> print(d1,type(d1))-----{10: 'Python'} <class 'dict'>
>>> d1.popitem()------(10, 'Python')
>>> print(d1,type(d1))------{} <class 'dict'>
>>> d1.popitem()------KeyError: 'popitem():
-------------------------------------------------------------------------------
```

e) get():

=>This function is used for obtaining the value of value by passing value of key.

=>If the value of key of does not exists then we get None

=>Syntax:-	varname=dictobj.get(Key)

Examples:

```
>>> d1={10:"Python",20:"Django",30:"Data Sci"}
>>> val=d1.get(10)
>>> print(val)----------Python
>>> val=d1.get(20
>>> print(val)--------Django
>>> val=d1.get(30)
>>> print(val)-------Data Sci
>>> val=d1.get(100)
>>> print(val)-------None

>>> d={"TS":"HYD","KAR":"BANG","TAMIL":"CHE","MH":"MUM"}
>>> print(d)---{'TS': 'HYD', 'KAR': 'BANG', 'TAMIL': 'CHE', 'MH': 'MUM'}
>>> cap=d.get("TS")
>>> print(cap)-----HYD
>>> cap=d.get("TAMIL")
>>> print(cap)------CHE
>>> cap=d.get("GOA")
>>> print(cap)------None
>>> print(d["GOA"])---------KeyError: 'GOA'
```

--

f) keys():

--

=>This Function is used for obtaining set of Values of Keys

=>Syntax: Varname=dictobj.keys()

Examples:

```
>>> d={"TS":"HYD","KAR":"BANG","TAMIL":"CHE","MH":"MUM"}
>>> states=d.keys()
>>> print(states)---------dict_keys(['TS', 'KAR', 'TAMIL', 'MH'])
>>> d1={10:"Python",20:"Django",30:"Data Sci"}
>>> crscodes=d1.keys()
>>> print(crscodes)---------dict_keys([10, 20, 30])
>>> for x in d:
...     print(x)
                    ...
                    TS
                    KAR
                    TAMIL
                    MH
>>> for x in d.keys():
...     print(x)
                    ...
                    TS
                    KAR
                    TAMIL
                    MH
>>> for kvr in d1:
...     print(kvr)
                    ...
                    10
                    20
                    30
>>> for kvr in d1.keys():
...     print(kvr)
                    ...
                    10
                    20
                    30
```

--

g) values():

--

=>This Function is used for obtaining set of Values of Value

=>Syntax: Varname=dictobj.values()

```
>>> d={"TS":"HYD","KAR":"BANG","TAMIL":"CHE","MH":"MUM"}
>>> print(d)
{'TS': 'HYD', 'KAR': 'BANG', 'TAMIL': 'CHE', 'MH': 'MUM'}
>>> vals=d.values()
>>> print(vals)
dict_values(['HYD', 'BANG', 'CHE', 'MUM'])
>>>
>>>
>>> for caps in vals:
...     print(caps)
                          ...
                          HYD
                          BANG
                          CHE
                          MUM
>>> for caps in d.values():
...     print(caps)
                          ...
                          HYD
                          BANG
                          CHE
                          MUM
```

--

h) items():

=>This function is used for obtaining (Key,value) from dict object.

=>Syntax: varname=dictobj.items()

Examples:

```
>>> d={"TS":"HYD","KAR":"BANG","TAMIL":"CHE","MH":"MUM"}
>>> print(d)----{'TS': 'HYD', 'KAR': 'BANG', 'TAMIL': 'CHE', 'MH': 'MUM'}
>>>
>>> keysvals=d.items()
>>> print(keysvals)--dict_items([('TS', 'HYD'), ('KAR', 'BANG'), ('TAMIL', 'CHE'),
                                                   ('MH', 'MUM')])
>>>
>>> for kv in keysvals:
...     print(kv)
                         ...
                         ('TS', 'HYD')
                         ('KAR', 'BANG')
                         ('TAMIL', 'CHE')
                         ('MH', 'MUM')
>>> for kv in d.items():
...     print(kv)
...
                         ('TS', 'HYD')
                         ('KAR', 'BANG')
                         ('TAMIL', 'CHE')
                         ('MH', 'MUM')
>>>
>>> for k,v in d.items():
```

```
...     print(k,"---->",v)
                                        ...
                                        TS ----> HYD
                                        KAR ----> BANG
                                        TAMIL ----> CHE
                                        MH ----> MUM
>>> for state,cap in d.items():
...     print(state,"===>",cap)
                                        ...
                                        TS ===> HYD
                                        KAR ===> BANG
                                        TAMIL ===> CHE
                                        MH ===> MUM
```

--

i) update():

=>This Function is used for updating / adding (Key,value) of One dict object with another dict object.

Syntax:-   dictobj1.update(dictobj2)

=>Here dictobj1 values are updated with dictobj2 values.

Examples:

```
>>> d1={10:3.4,20:4.5,30:6.7,40:3.4}
>>> print(d1)-------{10: 3.4, 20: 4.5, 30: 6.7, 40: 3.4}
>>> d2={100:1.2,200:2.2}
>>> print(d2)----------{100: 1.2, 200: 2.2}
>>>
```

```
>>> d1.update(d2)

>>> print(d1)-------{10: 3.4, 20: 4.5, 30: 6.7, 40: 3.4, 100: 1.2, 200: 2.2}

>>> d3={10:12.3,200:13.4}

>>> print(d3)------{10: 12.3, 200: 13.4}

>>> d1.update(d3)

>>> print(d1)-------{10: 12.3, 20: 4.5, 30: 6.7, 40: 3.4, 100: 1.2, 200: 13.4}

>>>

>>> d4={20:15.4,300:6.7}

>>> print(d4)---{20: 15.4, 300: 6.7}

>>> d1.update(d4)

>>> print(d1)---{10: 12.3, 20: 15.4, 30: 6.7, 40: 3.4, 100: 1.2, 200: 13.4, 300: 6.7}

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

>>> d1={1:["Python","ML","DL","AI"], 2:["C","CPP","8086"],3:["JAVA","SPRING"] }

>>> print(d1)

        {1: ['Python', 'ML', 'DL', 'AI'], 2: ['C', 'CPP', '8086'], 3: ['JAVA', 'SPRING']}

>>>

>>>

>>> for k in d1.keys():

...     print(k)

                        ...

                        1

                        2

                        3

>>> for v in d1.values():

...     print(v)

                        ...

                        ['Python', 'ML', 'DL', 'AI']

                        ['C', 'CPP', '8086']
```

```
                              ['JAVA', 'SPRING']
>>> for k,v in d1.items():
...     print(k,"---->",v)
                                        ...
                                        1 ----> ['Python', 'ML', 'DL', 'AI']
                                        2 ----> ['C', 'CPP', '8086']
                                        3 ----> ['JAVA', 'SPRING']
```

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

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

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 because it contains single value called None.

=>The value None is not a False, space , empty . Hence None isself 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
---------------------------------------X----------------------------------
```

==

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)

```
>>> 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)

--

```
>>> a=12.34
>>> print(a,type(a))---------12.34 <class 'float'>
>>> b=bool(a)
>>> print(b, type(b))--------True <class 'bool'>
>>> a=0.0
>>> print(a,type(a))--------0.0 <class 'float'>
>>> b=bool(a)
>>> print(b, type(b))-------False <class 'bool'>
>>> a=0.00000000000000000000000000000000000000001
>>> 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
```

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

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

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 because 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 coming 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

----

Program:-

```
#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("-----------------------------")
```

Program:-

```
#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 Otherwords, 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

Examples:

>>> 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

----

Examples:

>>> 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 displying 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 the 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. Programmatically, 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))
```

--

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))
```

…….

Program-1:-

```
#this program cal area of circle by accepting radious
r=float(input("Enter Radious:"))
ac=3.14*r**2
```

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

Program-2:-

```
#Circleex1.py
r=float(input("Enter Radious:"))
ac=3.14*r*r
print("Area of Circle={}".format(ac))
```

Program-3:-

```
#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)))
```

Program-4:-

```
#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)))
```

Program-5:-

```
#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))
```

Program-6:-

```
#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))
```

Program-7:-

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

Program-8:-

```
#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))
```

Program-9:-

```
#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))
```

Program-10:-

```
#This program reads two values and multiply them
#mulex10.py
```

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

==

Operators in Python

==

Index:

==

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.	-	Substraction	print(a-b)----------->7
3.	*	Multiplication	print(a*b)---------> 30
4.	/	Division	print(a/b)---------> 3.3333333333335
5.	//	Floor Division	print(a//b)--------> 3
6.	%	Modulo Division	print(a%b)--------> 1
7.	**	Exponentiation	print(a**b)------->1000

Program-1:-

```
#This Program demonstartes 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("{}**{}={}".format(a,b,a**b))
print("-"*50)
```

Program-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))
```

Program-3:-

```
#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(%f)=%0.2f" %(n,res))
```

==

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 Assigment 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
```

---X---

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

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	print(a>=b)----------False
		or equal to	print(a>=c)----------True
6.	<=	Less than	print(a<= -2)----------False
		or equal to	print(a<=c)-----------True

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

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".

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

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
```

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

Program-1:-

```
#This program demonstartes 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)
```

Program-2:-

```
#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("Swapoped Value of a:{}".format(a))
print("Swapoped Value of b:{}".format(b))
print("-"*50)
```

==

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 (^)

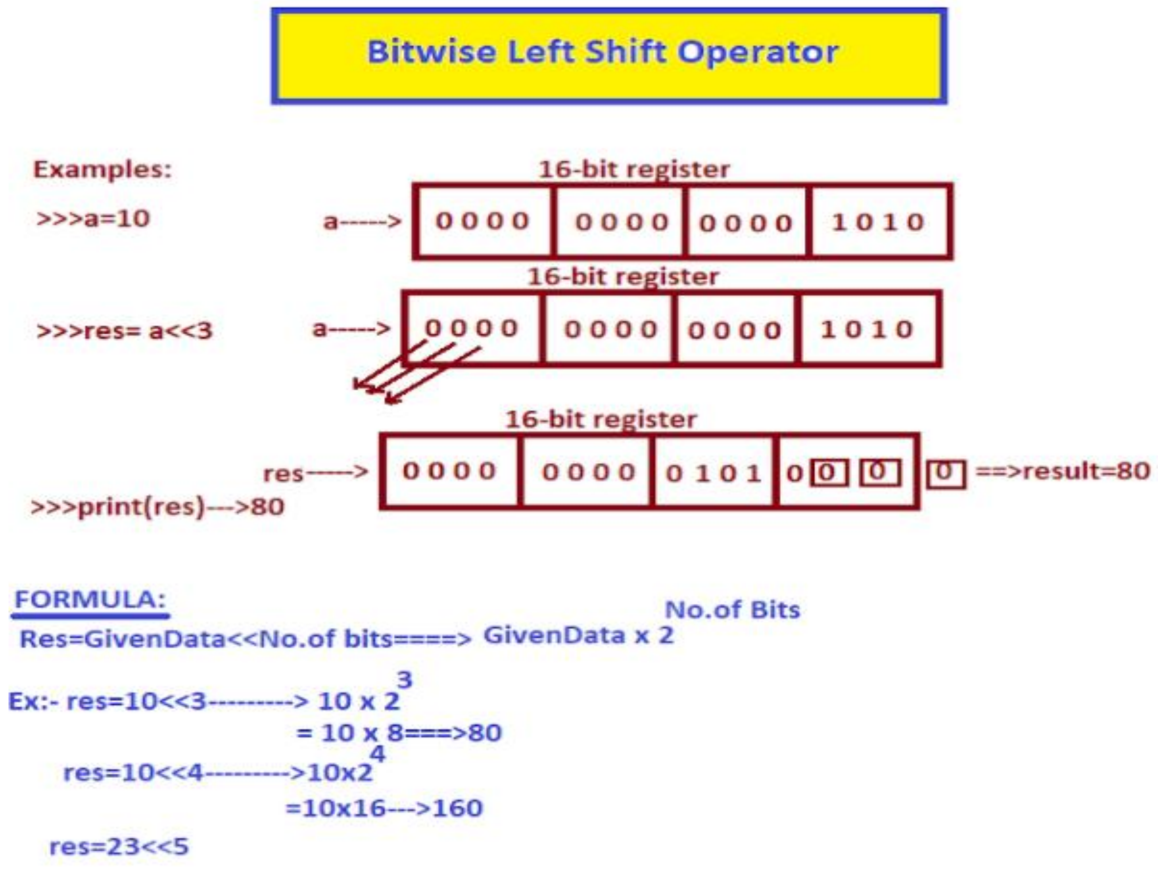
---

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)



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
```

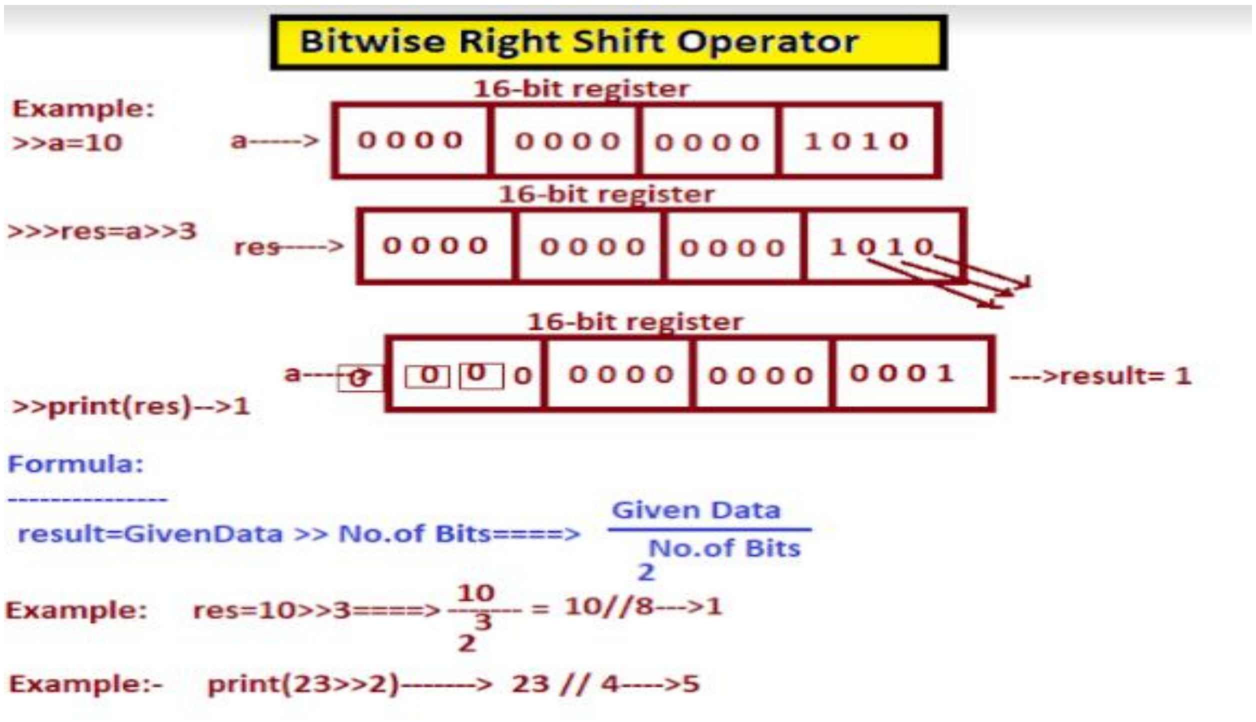
==

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 specfied number of bits)



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+1)
                                                   - ( 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

Eamples:

```
>>>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}
```

==============================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" contans 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 rerturns False.

----

b) is not

Syntax:-   varname= obj1 is not obj2

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

===

Index:-

a) Break
b) Continue
c) Pass

➔programming Examples

➔Nested loops Concepts

➔programming Examples

=>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 classfied into 3 types. They are

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

==

1. Conditional (or) Selection (or) Branching Statements

==

=>The purpose of Conditional / Selection / Branching Statements is that "To perform Certain operation Only once depends on Condition evaluation".

=>In Otherwords perform Certain X-operation when condition is True or Peform Y-Operation when Condtion is False only once.

=>In Python Programming, 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:

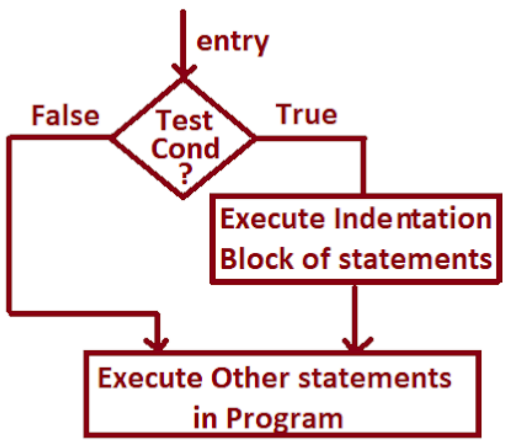
Indetation Symbol

```
if ( Test Cond ) :
    Statement-1
    Statement-2      <--Indentation
    --------------      Block
    Statement-4
Other statements in Program
```

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 Program.
=>If the test condition is False then PVM executes only Other statements in Program without executing Indentation block of statements.

Flow Chart for simple if statement



entry
False
Test Cond ?
True
Execute Indentation Block of statements
Execute Other statements in Program

Program 1:-

```
#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")
```

Program 2:-

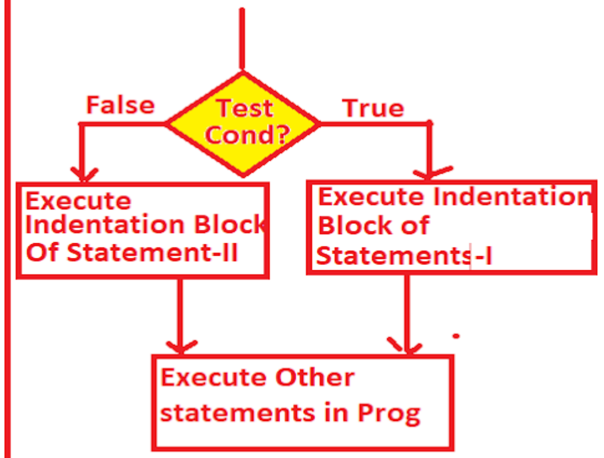
```
#posnegzero.py
n=int(input("Enter a number:"))    #  n--   10
if(n>0):
        print("{} is Possitive".format(n))
if(n<0):
        print("{} is Negative".format(n))
if(n==0):
        print("{} is Zero".format(n))
print("Program completed")
```

2. if..else statement

Syntax:

```
        if ( Test Cond ) :            <- Indentation Symbols
        |-----Statement-1  ]
        |-----Statement-2  ]  Indentation
        |----- ----------  ]  Block-I
        |-----Statement-n  ]
        else:
        |-----Statement-11 ]  Indentation
        |-----Statement-22 ]  Block-II
        |-----Statement-nn ]
        |-------------------------
        Other Statements in Prog
        -------------------------
```

Flow Chart if..else statement


False
Test Cond?
True
Execute Indentation Block Of Statement-II
Execute Indentation Block of Statements-I
Execute Other statements in Prog

Explanation of if..else statement

=>Here 'if' and 'else' are the Key words

=>If the test cond is True then PVM executes Indentation Block of statements-I and later excutes Other statements in Program (without executing Indentation Block of statement-II)

=>If the test cond is False then PVM executes Indentation Block of statements-II and later excutes Other statements in Program (without executing Indentation Block of statement-I)

=>Hence at any point of time PVM executes either Indentation Block of statements-I or Indentation Block of statement-II and later excutes Other statements in Program

Nested if..else statement i have 5 Operation with 5 decisions

Logic-1 --with simple if..

```
if (test cond1)
        do  x-op
if( test cond2):
       do  y-op
if( test cond3):
       do  Z-op
if( test cond4):
       do  k-op
if( test cond5):
       do  L-op
```

Logic-2 -----with if..else

nested if ..else or if..else ladder

```
if (Test Cond-1):
|     do  x-operation
else:
    if (Test Cond-2):
    |     do  y-operation
    else:
        if (Test Cond-3):
        |      do  z-operation
        else:
            if (Test Cond-4):
            |     do  K-operation
            else:
                  do L-Operation
```

Program-1:
#This program accept a digit and print its name
#digitex1.py

```
d=int(input("Enter a digit:")) # 0 1 2 3 4 5 6  7  8  9
if(d==0):
        print("{} is ZERO".format(d))
if(d==1):
        print("{} is ONE".format(d))
if(d==2):
        print("{} is TWO".format(d))
if(d==3):
        print("{} is THREE".format(d))
if(d==4):
        print("{} is FOUR".format(d))
if(d==6):
        print("{} is SIX".format(d))
if(d==5):
        print("{} is FIVE".format(d))
if(d==7):
        print("{} is SEVEV".format(d))
if(d==8):
        print("{} is EIGHT".format(d))
if(d==9):
        print("{} is NINE".format(d))
if d not in [0,1,2,3,4,5,6,7,8,9]:
        print("{} a NUMBER:".format(d))
```

Program-2:
#emppayslip.py

```
eno=int(input("Enter Employee Number:"))
ename=input("Enter Employee Name:")
basicsal=float(input("Enter Employee Basic Salary:"))
if(basicsal>=10000):
        da= (20/100)*basicsal
        ta=(15/100)*basicsal
        hra=(12/100)*basicsal
        cca=(2/100)*basicsal
        ma=(2/100)*basicsal
        gpf=(2/100)*basicsal #  deduction
        lic=(1/100)*basicsal   # deduction
else:
        da= (25/100)*basicsal
        ta=(20/100)*basicsal
        hra=(16/100)*basicsal
        cca=(3/100)*basicsal
        ma=(2/100)*basicsal
        gpf=(2/100)*basicsal #  deduction
        lic=(1/100)*basicsal   # deduction
netsal=(basicsal+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("\tEmployee Name:{}".format(ename))
print("\tEmployee Basic salary:{}".format(basicsal))
print("\tEmployee DA:{}".format(da))
print("\tEmployee TA:{}".format(ta))
print("\tEmployee HRA:{}".format(hra))
print("\tEmployee 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)
```

Program-3:

```
#This program accepts a number and decides weather it is even or odd
#evenodd.py
n=int(input("Enter a number:"))
if (n%2==0):
       print("{} is EVEN".format(n))
else:
       print("{} is ODD".format(n))

print("Program execution completed:")
```

Program-4:

```
#This program accepts a number and decides weather it is +ve or -ve or
zero
#posnegzero1.py
n=int(input("Enter a number:"))
if(n==0):
       print("{} is ZERO".format(n))
else:
       if(n>0):
              print("{} is +VE".format(n))
       else:
              print("{} is -VE".format(n))
```

3. if..elif..else statement:

Syntax:

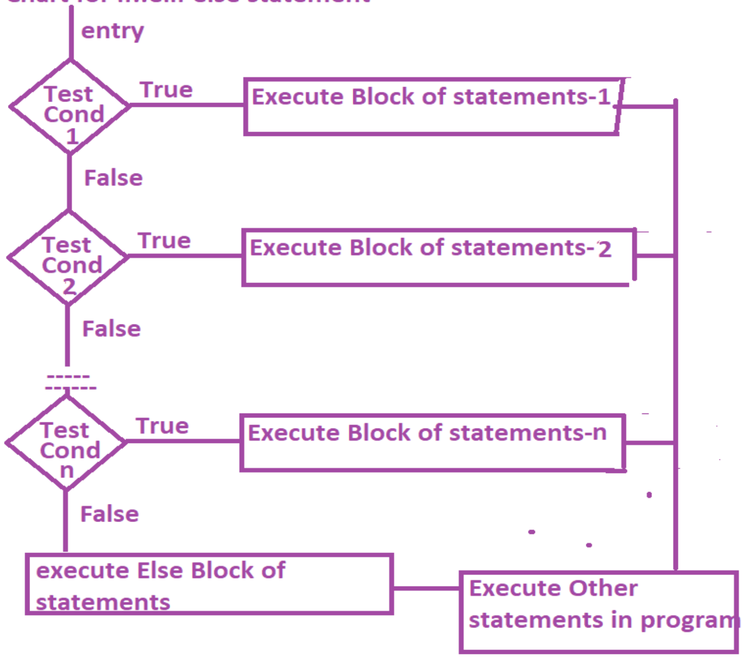
----

if (test cond1):
Block of statements-I
elif (test cond-2):
Block of statements-2
elif (test cond-3):
Block of statements-3

----
elif (test cond-n):
Block of statements-n
else:
Else Block of statements

Other statements in Program

Flow Chart for if..elif else statement


entry
Test Cond 1
True
Execute Block of statements-1
False
Test Cond 2
True
Execute Block of statements-2
False
Test Cond n
True
Execute Block of statements-n
False
execute Else Block of statements
Execute Other statements in program

Explanation for if..elif..else statement

=>here if the test cond1 is true then PVM executes Block of statements-1 and also executes Other statements in Program
=>if the Test cond1 is False then evaluate test cond2 and if it is true then PVM execute Block of statement-2 and also execute Other statements in Program
=>if the Test cond2 is False then evaluate test cond3 and if it is true then PVM execute Block of statement-3 and also execute Other statements in Program.
=>This process will performed until all Test condtions are evaluated.
=>If all test conditions are false then execute Else block of statements which are written under else block.
=>writing else block is optional.

Program-1:

```
#Program for accepting 3 integers and find biggest among them
#bigex1.py
a=int(input("Enter First Number:"))   # 10
b=int(input("Enter Second Number:"))#b=13
c=int(input("Enter Third Number:"))# c=2
#find big
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,c))
elif(a==b) and (c==b) and (c==a):
       print("ALL Values are Equal:")
```

Program-2:

```
#This program accept a digit and print its name
#digitex2.py
d=int(input("Enter a digit:")) # 0 1 2 3 4 5 6  7  8  9
if(d==0):
       print("{} is ZERO".format(d))
elif(d==1):
       print("{} is ONE".format(d))
elif(d==2):
       print("{} is TWO".format(d))
elif(d==3):
       print("{} is THREE".format(d))
elif(d==4):
       print("{} is FOUR".format(d))
elif(d==6):
       print("{} is SIX".format(d))
elif(d==5):
       print("{} is FIVE".format(d))
elif(d==7):
       print("{} is SEVEV".format(d))
elif(d==8):
       print("{} is EIGHT".format(d))
elif(d==9):
       print("{} is NINE".format(d))
elif d  in [-1,-2,-3,-4,-5,-6,-7,-8,-9]:
       print("{}  is -VE  Digit :".format(d))
elif d not in [0,1,2,3,4,5,6,7,8,9]:
       print("{}  NUMBER:".format(d))
```

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

4. match case statement

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

=>match…case is one of the new features in Python 3.10 onwards

=>match case statement is always recommended to deal with pre-defined / designed decisions / Operations.

=>Syntax:-

```
match ( Choice Expression ):
        case Lable1:
                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....lable-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

Program-1:

```
#Menu driven application for arithmetic operations by using match case
statement
#aop.py
import sys
print("-"*50)
print("\tArithmetic Operations")
print("-"*50)
print("\t1. Addition:")
print("\t2. Substraction:")
print("\t3. Multiplication:")
print("\t4. Division:")
print("\t5. Modulo Division:")
print("\t6. Exponetiation:")
print("\t7. Exit:")
print("-"*50)
ch=int(input("Enter Ur Choice:"))
match(ch):
       case 1:
              a,b=float(input("Enter First Value for
Addition:")),float(input("Enter Second Value for Addition:"))
              print("sum({},{})={}".format(a,b,a+b))
       case 2:
              a=float(input("Enter First Value for Substraction:"))
              b=float(input("Enter Second Value for Substraction:"))
              print("sub({},{})={}".format(a,b,a-b))
       case 3:
              a=float(input("Enter First Value for Multiplication:"))
              b=float(input("Enter Second Value for Multiplication:"))
              print("mul({},{})={}".format(a,b,a*b))
       case 4:
              a=float(input("Enter First Value for Division:"))
              b=float(input("Enter Second Value for Division:"))
              print("div({},{})={}".format(a,b,a/b))
              print("floor div({},{})={}".format(a,b,a//b))
       case 5:
              a=float(input("Enter First Value for Modulo Division:"))
              b=float(input("Enter Second Value for Modulo Division:"))
              print("mod({},{})={}".format(a,b,a%b))
       case 6:
              a=float(input("Enter Base:"))
              b=float(input("Enter Power:"))
              print("exp({},{})={}".format(a,b,a**b))
       case 7:
              print("\nThanks for Using Program!")
              sys.exit()
       case _:
              print("Ur Choice of selection is wrong!")
```

Program-2:

```
#matchcaseex.py
wkn=input("Enter the week name:")
```

```
match(wkn.lower()):
        case "monday":
                print("{} is working day:".format(wkn))
        case "tuesday":
                print("{} is working day:".format(wkn))
        case "wednessday":
                print("{} is working day:".format(wkn))
        case "thursday":
                print("{} is working day:".format(wkn))
        case "friday":
                print("{} is working day:".format(wkn))
        case "saturday":
                print("{} is Week End:".format(wkn))
        case "sunday":
                print("{} is Holiday day:".format(wkn))
        case _:
                print("{} is not week day:".format(wkn))
```

Program-3:

```
#matchcaseex1.py
wkn=input("Enter the week name:")
match(wkn.lower()):
        case "monday" |"tuesday"| "wednessday"|"thursday"| "friday":
                print("{} is working day:".format(wkn))
        case "saturday":
                print("{} is Week End:".format(wkn))
        case "sunday":
                print("{} is Holiday day:".format(wkn))
        case _:
                print("{} is not week day:".format(wkn))
```

==

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

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

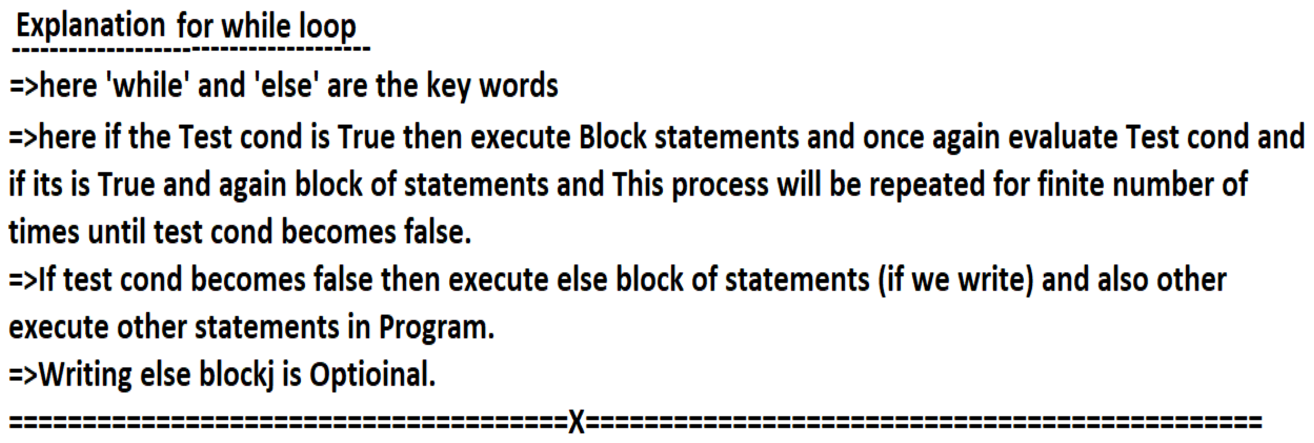
=>At time of dealing with loop of Python, we must ensure that there must exists 3 points.

1)Initialization Part

2)Conditional Part

3)Updation Part (Increment or Decrement)

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



Explanation for while loop

=>here 'while' and 'else' are the key words
=>here if the Test cond is True then execute Block statements and once again evaluate Test cond and if its is True and again block of statements and This process will be repeated for finite number of times until test cond becomes false.
=>If test cond becomes false then execute else block of statements (if we write) and also other execute other statements in Program.
=>Writing else blockj is Optioinal.
=================================X===

1) while (or) while ...else

Syntax-1

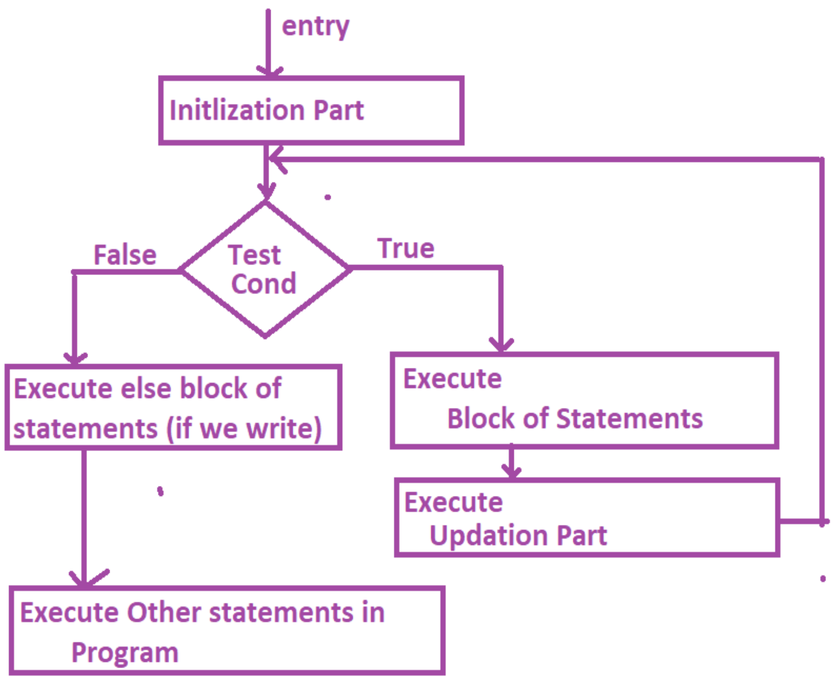
```
while (test cond):
        statement-1
        statement-2
        ----------------
        statement-n
-------------------------
Other statements in Program
-------------------------
```

(OR)

Syntax-2

```
while (test cond):
        statement-1
        statement-2
        ----------------
        statement-n
else:
        else block of statements
-------------------------
Other statements in Program
-------------------------
```

Flow Chart for while..else statement



entry
Initlization Part
Test Cond
False
True
Execute else block of statements (if we write)
Execute Block of Statements
Execute Updation Part
Execute Other statements in Program

Program-1

```
#This program generates multiplication table for a given +ve number
#multable.py
n=int(input("Ener a number:"))
if(n<=0):
        print("{} is invalid input:".format(n))
else:
        i=1
        print("*"*50)
        print("\tMul table for {}".format(n))
        print("*"*50)
        while(i<=10):
                print("\t{} x {}={}".format(n,i,n*i))
                i=i+1
        else:
                print("*"*50)
```

Program-2

```
# This program generates 1 to n numbers where 'n' is +ve
#NumGenex1.py
n=int(input("Enter How many numbers number u want generate:"))
if(n<=0):
        print("{} is invalid input".format(n))
```

```
else:
	i=1 #Initlization part
	print("-"*50)
	print("Range of Values within:{}".format(n))
	print("-"*50)
	while(i<=n): # cond part
		print("Val of i=",i)
		i=i+1 #updation part
	else:
		print("-"*50)
		print("\nI am from else part of while loop")
	print("\ni am from other part of program")
```

Program-3

```
#This program accepts a number and find its digits sum
#sumdigits.py
n=int(input("Enter a number:"))
if(n<=0):
	print("{} is inavlid input".format(n))
else:
	print("-"*50)
	print("Given Number={}".format(n))
	print("-"*50)
	s=0
	while(n>0):
		d=n%10
		s=s+d
		n=n//10
	else:
		print("-"*50)
		print("Sum of digits={}".format(s))
		print("-"*50)
```

==

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)) unitl all elements are completed in Iterable_object.

=>After all elements of Iterable_Object of for loop completed and it indicates condtion 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-Iterbale_objects repetation, It is recommended for using " while Loop"

==

Program-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=" ")
```

Program-2

```
#Forloopex2.py
a={10,23.45,100.5,2+3j}
for i in a:
        print(i)
```

Program-3

```
#Multableex2.py
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)
```

Program-4

```
#numgenex2.py
n=int(input("Enter a number:"))
if(n<=0):
        print("{} is invalid input:".format(n))
```

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

Program-5

```
#NatNumSum.py
n=int(input("Enter a number for finding sum of First Numbers , squares and
cubes:"))
if(n<=0):
        print("{} is invalid input:".format(n))
else:
        s,ss,cs=0,0,0
        print("-"*50)
        print("\tNat Nums\tSquares\t\tCubes")
        print("-"*50)
        for val in range(1,n+1):
                print("\t{}\t\t{}\t\t{}".format(val, val**2,val**3))
                s=s+val
                ss=ss+val**2
                cs=cs+val**3
        else:
                print("-"*50)
                print("\t{}\t\t{}\t\t{}".format(s,ss,cs))
                print("-"*50)
```

Program-6

```
#Program accepting list of values and find sum and avg--Approach-1.
#sumavg.py
n=int(input("Enter How many elements sum u want to find:")) # 5
if(n<=0):
        print("{} is Invalid Input:".format(n))
else:
        lst=list()    #  [ ]
        for i in range(1,n+1):
                val=float(input("Enter {} Value:".format(i)) )
                lst.append(val)
        else:
                print("-"*40)
                print("Content of list={}".format(lst))
                print("-"*40)
                s=0
                for val in lst:
                        print("\t{}".format(val))
                        s=s+val
                else:
                        print("-"*50)
                        print("\tsum={}".format(s))
                        print("\tAvg={}".format(s/len(lst)))
                        print("-"*50)
```

Program-7

#This Program accept number of Integer values and sort then in both ascending and descending orders

```
#sortnumbers.py
n=int(input("Enter How Many number u want sort:"))
if(n<=0):
        print("{} is invalid input:".format(n))
else:
        lst=list() # empty list
        print("Enter {} Values:".format(n))
        for i in range(1,n+1):
                val=int(input())
                lst.append(val)
        else:
                print("="*50)
                print("Original List of elements:")
                print("="*50)
                for val in lst:
                        print("{}".format(val),end=" ")
                else:
                        print()
                        print("="*50)
                        lst.sort() # lst data sorts in ascending order
                        print("List of elements in Ascending Order:")
                        for val in lst:
                                print("{}".format(val),end=" ")
                        else:
                                print()
                                print("="*50)
                                lst.sort(reverse=True)
                                print("List of elements in Decending Order:")
                                for val in lst:
                                        print("{}".format(val),end=" ")
                                else:
                                        print()
                                        print("="*50)
```

Program-8

#Program accepting list of values and find sum and avg--Approach-1.

```
#avgsum.py
n=int(input("Enter How many elements sum u want to find:")) # 5
if(n<=0):
        print("{} is Invalid Input:".format(n))
```

```
else:
        i=1
        s=0
        while(i<=n):
                val=float(input("Enter {} Value:".format(i)))
                s=s+val
                i=i+1
        else:
                print("Sum={}".format(s))
                print("Average={}".format(s/n))
```

Program-9

```
#Program accepting list of values and find sum and avg--Approach-1.
#avgsum1.py
n=int(input("Enter How many elements sum u want to find:")) # 5
if(n<=0):
        print("{} is Invalid Input:".format(n))
else:
        i=1
        s=0
        print("Enter {} Values:".format(n))
        while(i<=n):
                val=float(input())
                s=s+val
                i=i+1
        else:
                print("Sum={}".format(s))
                print("Average={}".format(s/n))
```

Program-10

```
#Program accepting list of values and find sum and avg--Approach-1.
#sumavg1.py
print("Enter Number of Values separated by space:")
lst=[float(val) for val in input().split()]  # List Comprehension
print("Content of list={}".format(lst))
s=0
for val in lst:
        print("\t{}".format(val))
        s=s+val
else:
        print("-"*50)
        print("\tsum={}".format(s))
        print("\tAvg={}".format(s/len(lst)))
        print("-"*50)
```

Program-11

```
#Program for accepting any number and obtains its reversed format
# Hint  : 1234------------> 4321
#palindrome.py
n=int(input("Enter a number:"))  # 1234
if(n<=0):
        print("{} is invalid input".format(n))
else:
        rn=0
        tn=n #preserver 'n' value into temp number
        while(n>0):
                d=n%10
                rn=rn*10+d
                n=n//10
        else:
                if(rn==tn):
                        print("Given Number is Palindrome:")
                else:
                        print("Given Number is Not Palindrome:")
```

Program-12

```
#Program for accepting any number and obtains its reversed format
# Hint  : 1234------------> 4321
#reverse.py
n=int(input("Enter a number:"))  # 1234
if(n<=0):
        print("{} is invalid input".format(n))
else:
        rn=0
        while(n>0):
                d=n%10
                rn=rn*10+d
                n=n//10
        else:
                print("Orginal Number={}".format(n))
                print("Reversed Number={}".format(rn))
```

==

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

==

Continue statement

==

=>continue is one of the keywords

=>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
--------------------------------------------------------
--------------------------------------------------------
```

Program-1

```
#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")
```

Program-2

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

Program-3

```
#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))
```

Program-4

```
#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))
```

Program-5

```
#This program accept a number and decide whether 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))
```

Program-6

```
#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))
```

Program-7

```
#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))
```

Program-8

```
#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
```

Program-9

```
#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("Possitive 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)
```

Program-10

```
#This Program accept a line of text and display the Vowels
#continueex4py
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=======================================

Program-1

```
#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
----------------------------------------
"""
```

Program-2

```
#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"""
```

Program-3

```
#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
----------------------------------------"""
```

Program-4

```
#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"""
```

Program-5

```
#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)
```

Program-6

```
#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)
```

Program-7

```
#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(n)
		else:
			print("Given List={}".format(lst))
			print("Prime Numbers List={}".format(primelist))
			print("Non-Prime Numbers 
List={}".format(nonprimelist))
```

Program-8

```
#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))
```

Special Program on Loop:-

Program-1

```
#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:")
```

Program-2

```
#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))
```

Program-3

```
#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))
```

Program-4

```
#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"

#Display the Marks Memo
print("="*70)
print("\tS t u d e n t  M a r k s  R e p o r t")
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)
```

==

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 Performnace is Degraded

=>Redundency (Duplication) of Code is More

=>Example: GW-BASIC

---

2. Structured Programming Languages

=>Structured Programming Languages contains 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 (Improved)

=>Redundancy (Duplication) of Code is Minimized

=>Example: C,CPP,JAVA, PYTHON...etc

==

Functions in Python

==

=>The purpose of Functions is that "To perform certain Operation and prvides 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)

=>Redundancy (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 perticular 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.

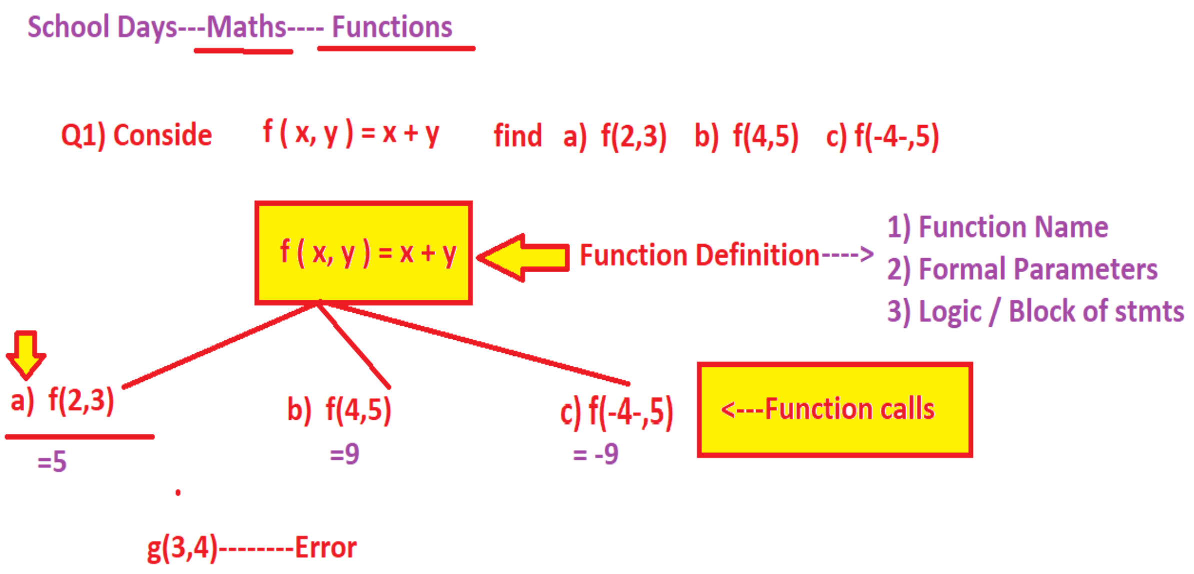
Phases in Functions:

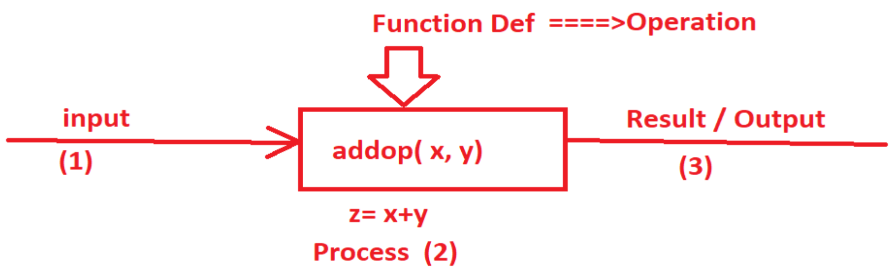
=>In Functions, there must exists 3 phases.

a) Every Function Must Take INPUT

b) Every Function Must PROCESS the input

c) Every Function must give OUTPUT / RESULT





We are all developing a TASK --GW-BASIC
"ADDITION of TWO Numbers---350---Students--team members and KVR is a teacher

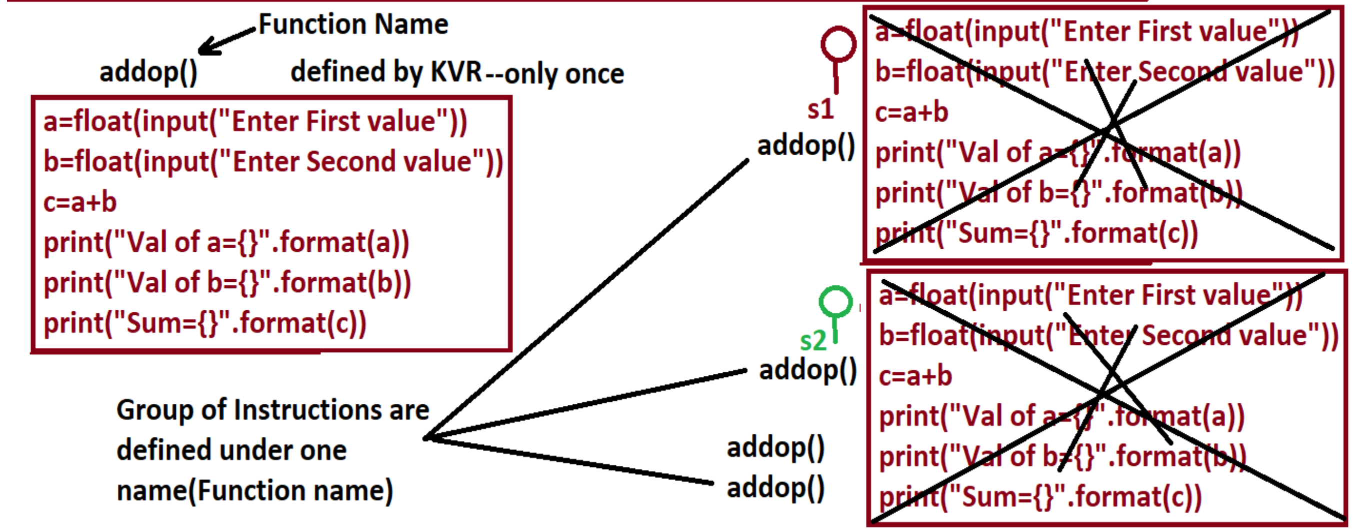
In this Programming
--350 Students has to
defined 350 times 6 lines
of code

=>Application Development time is More
=>Application Memory Space is More
=>Application Execution time is More
=>Application Performnace is Degraded
=>Redundency (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))
```

We are all developing a TASK --~~GW-BASIC~~ C,CPP.JAVA,.NET, PYTHON --Structured PLs
"ADDITION of TWO Numbers---350---Students--team members and KVR is a teacher

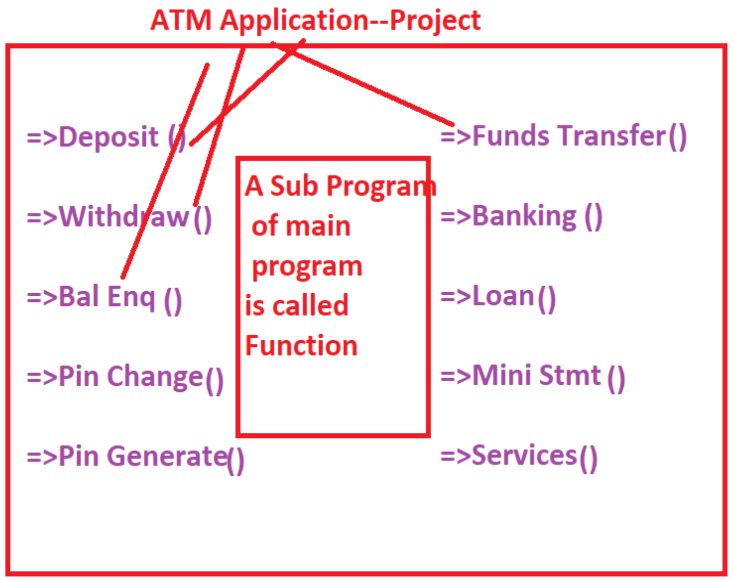


Function Name
addop()
defined by KVR--only once
Group of Instructions are defined under one name(Function name)
s1
addop()
s2
addop()
addop()
addop()

```
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))
```

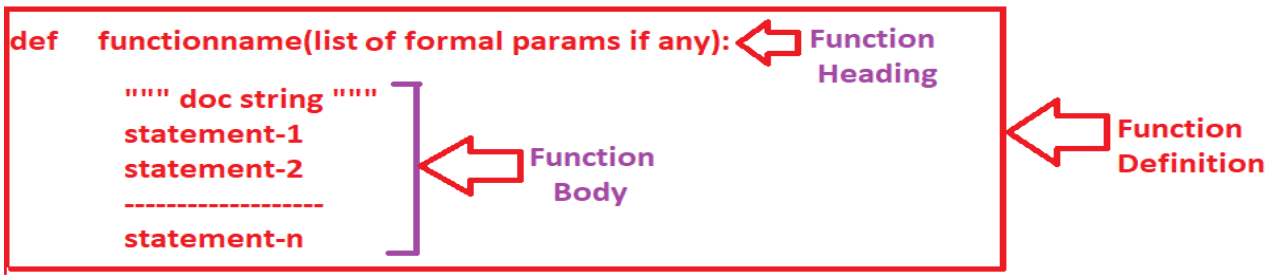
~~a=float(input("Enter First value"))~~
~~b=float(input("Enter Second value"))~~
~~c=a+b~~
~~print("Val of a={}".format(a))~~
~~print("Val of b={}".format(b))~~
~~print("Sum={}".format(c))~~

~~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))~~



Def: A part of main program is called Function.

Syntax for defining the Function:



```
def    functionname(list of formal params if any):
           """ doc string """
           statement-1
           statement-2
           -------------------
           statement-n
```

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 correspondig Function Definition only but not possible to access in some function definitions.

Program-1:

```
#This function computes sum of two numbers by using functions.
#Approach1.py
#Input<----Taking From Function Call
#Process<----Inside of Function Body
#Result/Output<----Function Call
def  addop(a,b):
        c=a+b
        return c

#main program
x=float(input("Enter First value:"))
y=float(input("Enter Second value:"))
result=addop(x,y)
print("sum=",result)
```

Program-2:

```
#This function computes sum of two numbers by using functions.
#Approach2.py
#Input<----Taking Inside of  Function Body
#Process<----Inside of Function Body
#Result/Output<----Inside of Function Body
def   addop():
        x=float(input("Enter First value:"))
        y=float(input("Enter Second value:"))   # Input
        z=x+y      # Process
        print("Sum({},{})={}".format(x,y,z)) # output
```

```
#main program
addop()
```

Program-3:

```
#This function computes sum of two numbers by using functions.
#Approach3.py
#Input<----Taking from  Function calls
#Process<----Inside of Function Body
#Result/Output<----Inside of Function Body
def  addop(x,y):
       z=x+y
       print("sum(%0.2f,%0.2f)=%0.2f" %(x,y,z))

#main program
a,b=float(input("Enter First Value:")),float(input("Enter Second Value:"))
addop(a,b)
```

Program-4:

```
#This function computes sum of two numbers by using functions.
#Approach4.py
#Input<----Taking Inside of  Function body
#Process<----Inside of Function Body
#Result/Output<----Gives result Function Call
def  addop():
       k=float(input("Enter First Value:"))
       v=float(input("Enter Second Value:"))
       r=k+v
       return r

#main program
result=addop()
print("sum={}".format(result))
```

Program-5:

```
#This function computes sum of two numbers by using functions.
#Approach41.py
#Input<----Taking Inside of  Function body
#Process<----Inside of Function Body
#Result/Output<----Gives result Function Call
def  addop():
       k=float(input("Enter First Value:"))
       v=float(input("Enter Second Value:"))
       r=k+v
       return k,v,r # In python programming, return stmt can return either
one value or multiple values

#main program
k1,v1,r=addop()  # Function call with multi line assignment
print("sum({},{})={}".format(k1,v1,r))
print("==========OR===========")
result=addop()  # Function call with single line assignment
print("type of result var=",type(result)) # type of result var= <class
'tuple'>
print("sum({},{})={}".format(result[0],result[1],result[2]))
print("===========OR============")
print("sum({},{})={}".format(result[-3],result[-2],result[-1]))
```

Program-6:

#This function computes sum of two numbers by using functions.
#FunEx1.py

```
def  addop(a,b):
	"""This program cal sum of two numbers"""
	c=a+b # here 'c' is called Local variable
	return c  # here return statement used to return the result from
function body.

#main program
c=addop(10,20)  # Function call
print("sum=",c)
print("type of addop=", type(addop))
print("doc string of addop()=",   addop.__doc__)
```

Program-7:

#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-8:

#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 Decending 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 delimeter (space, comma, colon..etc)

=>List comprehension is the most effective way for reading the data for list instead tradtional reading the data.

=>**Syntax:- listobj=[expression for varname in Iterable_object]**

=>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]

Program-1:

```
#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)
```

Program-2:

```
#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)
```

Program-3:

```
#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 Decending 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)
```

Program-4:

```
#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 Decending 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 Parmeters / 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".

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

==

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.

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

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

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).

Program-1

#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)
```

Program-2

#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)
```

Program-3

#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)
```

Program-4

#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)
```

Program-5

```
#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

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

==

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 Impalement, Variable length Parameters concept, we must define single Function Definition and takes a formal Parameter preceded with a symbol called asterisk (* param) and the formal parameter with asterisk 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 because Variable number of values are treated as Posstional Argument Value(s)

Program-1

```
#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)
```

Program-2

```
#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)
```

Program-3

```
#kwdparamex3.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)
```

Program-4

```
#nonvarlenargex1.py----
#This program will not execute as it is shown bellow , because PVM
remebers 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
```

Program-5

```
#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
```

Program-6

```
#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
```

Program-7

```
#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)
```

Program-8

```
#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")
```

Program-9

```
#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="Dajngo",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 multiple function definitions. 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 asterisk (** param) and the formal parameter with double asterisk 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)

Program-1:

```
#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()
```

Program-2:

```
#This program computes total martks 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  M a r k s  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\t{}".format(sub,marks))
                tm=tm+marks
        else:
                print("-"*50)
                print("\tTotal 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, whish are defined before all the function calls.
=>Local Variables are those, which are used for storing Temporary results in the Function Body.
=>Local Variables are those, whish are defined within the Function Body
=>Syntax:-

```
                 FileName.py
                 ------------------
                 var1=val1
                 var2=val2   # here var1,var2...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
                                        ------------------------------
==============================X==================================
```

Program-1:

```
#globalvarex1.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 because 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 because 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 because they are local
in other functions
#main program
learnDataSci()
learnAI()
learnML()
```

Program-2:

```
#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 because 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 because 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 because they are local
in other functions
#main program
lang="PYTHON"  # here 'lang' is called Global Variable
learnDataSci()
learnAI()
learnML()
```

Program-3:

```
#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 because 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 because 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 because they are local
in other functions
#main program
learnDataSci()
learnAI()
learnML()
```

Program-4:

```
#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 because 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 because 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 because 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 defintion then global variable names must be preceded with 'global' keyword otherwise we get "UnboundLocalError: local variable names referenced before assignment"

Syntax:

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

```
def   fun2():
       ------------------------
        global var1,var2...var-n
     # Modify var1,var2....var-n
    --------------------------
```

Program-1:

```
#globalkwdex1.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()
```

Program-2:

```
#globalkwdex2.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
```

Program-3:

```
#globalkwdex3.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
```

===

global and local variables and globals()

===

=>When we come across same global Variable names and Local Variable Names in same function definition then PVM gives preference for local variables but not for global variables.

=>In this context, to extract / retrieve the global variables names along with local variables, we must use globals() and it returns an object of <class,'dict'> and this dict object stores all global variable Names as Keys and global variable values as values of value.

=>Syntax:-

```
var1=val1
var2=val2
--------------
var-n=val-n  # var1, var2...var-n are called global Variables
def   functionname():
     ------------------------
     var1=val11
```

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

Program-1:

```
#globalsfunex1.py
a=10
b=20
c=30
d=40   # here a,b,c,d are called global variables
def  operation():
       a=1
       b=2  # here 'a' and 'b' are called Local Variable
       print("val of a(global variable) in operation()=",
globals().get('a'))
       print("val of b(global variable) in operation()=", globals()['b'] )
       print("val of a(Local Varaible) operation()=",a)
       print("val of b(Local Varaible) operation=",b)
       print("val of c (Global Variable) operation()=",c)
       print("val of d (Global variable) operation()=",d)

#main program
operation()
print("\nval of a main program=",a)
print("val of b main program=",b)
print("val of c main program=",c)
print("val of d main program=",d)
```

Program-2:

```
#globalsfunex2.py
a=10
b=20
c=30
d=40
def   operation():
       kvr=globals()   #  kvr  object---dict { Extrenal Global Varibles,
'a':10, 'b':20,'c':30,'d':40}
       print("type of kvr=",type(kvr))
       print("-------------------------------------")
       for k,v in kvr.items():
              print("{}----->{}".format(k,v))
       else:
              print("-------------------------------------")


#main program
operation()


"""
E:\KVR-PYTHON-9AM\FUNCTIONS>py globalsfunex2.py
type of kvr= <class 'dict'>
-------------------------------------
__name__----->__main__
__doc__----->None
__package__----->None
__loader__-----><_frozen_importlib_external.SourceFileLoader object at
0x000001F1151646A0>
__spec__----->None
__annotations__----->{}
__builtins__-----><module 'builtins' (built-in)>
__file__----->E:\KVR-PYTHON-9AM\FUNCTIONS\globalsfunex2.py
__cached__----->None
a----->10
b----->20
c----->30
d----->40
operation-----><function operation at 0x000001F114C93E20>
------------------------------------- """
```

Program-3:

```
#globalsfunex3.py
a=10
b=20
c=30
d=40
def   operation():
       d=globals()
       print("--------------------------------------------------------")
       print("Out Function / Program Global variable Values:")
       print("--------------------------------------------------------")
       print("\tValue a =",d['a'] )
       print("\tValue b =",d['b'] )
       print("\tValue c =",d['c'] )
       print("\tValue d =",d['d'] )
       print("----------------OR-------------------------------------")
       print("\tValue a =",d.get('a') )
```

```
        print("\tValue b =",d.get('b') )
        print("\tValue c =",d.get('c') )
        print("\tValue d =",d.get('d') )
        print("----------------OR-------------------------------------")
        print("\tValue a =",globals().get('a'))
        print("\tValue b =",globals().get('b') )
        print("\tValue c =",globals().get('c') )
        print("\tValue d =",globals().get('d') )
        print("-----------------OR-------------------------------------")
        print("\tValue a =",globals()['a'] )
        print("\tValue b =",globals()['b'] )
        print("\tValue c =",globals()['c'] )
        print("\tValue d =",globals()['d'] )


#main program
operation()
```

Program-4:

```
#globalsfunex4.py
a=10
b=20
c=30
d=40   # here a,b,c,d are called global variables
def  operation():
        global c,d;
        c=c+1  #  c= 31
        d=d+1 # 41
        a=1
        b=2  # here 'a' and 'b' are called Local Variable
        res=a+b+c+d+globals().get('a')+globals()['b']
        print("result=",res)

#main program
operation()
```

==

==

Anonymous or Lambda Functions

==

=>Anonymous Functions are those which does not contain any name explicitly.

=>The purpose of Anonymous Functions is that "To Peform 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:

Part	Description
varname	The variable name to which the lambda function is assigned. This variable becomes a callable function.
lambda	The keyword used to define an anonymous (lambda) function.
params-list	The input variables for the function.
expression	A single executable statement that computes and returns a result.

=>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.

==

Program-1:

```
#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)
```

Program-2:

#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")
```

Program-3:

#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")
```

Program-4:

#Write a python program which will calculate all types of academic operations (Use menu driven application along with anonymous function).

```
#aopmenuop2.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
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 Multiplication:"))
                        x2=float(input("Enter Second Value 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 Floor Division:"))
                        x2=float(input("Enter Second Value Floor Division:"))
                        res=floordivop(x1,x2)
                        print("FloorDiv({},{})={}".format(x1,x2,res))
                case 6:
                        x1=float(input("Enter First Value Modulo Division:"))
                        x2=float(input("Enter Second Value 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-5:

#Write a python program which will calculate all types of academic operations (Use menu driven application along with anonymous function).

```
#aopmenuop3.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  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  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 is wrong!--try again")
```

Program-6:

```
#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-7:

```
#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-8:

#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-9:

#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 Ityerable 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 condtion, 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.

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

Program-1:

#This program takes list of values and extract only +ve values

```
#FilterEx1.py
def   positve(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(positve,lst)
print("type of obj=",type(obj1)) #<class, 'filter'>
pslist=list(obj1)
print("Original Elements=",lst)
print("Possitive Elements=",pslist)
obj2=filter(negative,lst)
ngtpl=tuple(obj2)
print("Negative Elements=",ngtpl)
```

Program-2:

#This program takes list of values and extract only +ve values

```
#FilterEx2.py
def   positve(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(positve,lst)
print("type of obj=",type(obj1)) #<class, 'filter'>
pslist=list(obj1)
print("Original Elements=",lst)
print("Possitive Elements=",pslist)
obj2=filter(negative,lst)
ngtpl=tuple(obj2)
print("Negative Elements=",ngtpl)
```

Program-3:

```
#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("---------------------------------------")
```

Program-4:

```
#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("---------------------------------------")
```

Program-5:

```
#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,nglist

#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("Possitive Elements:",psl)
print("Negetive Elements:",ngl)
```

Program-6:

```
#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)
```

Program-7:

```
#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("--------------------------------------------------")
```

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

==

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 other words , 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 .

---X--

Program-1:

```
#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)))
sqrlist=list(mapobj)
print("Original Elements:{}".format(oldlst))
print("New  Elements:{}".format(sqrlist))
```

Program-2:

```
#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()]
sqrlist=tuple(map(lambda val:val**2 , oldlst))
print("Original Elements:{}".format(oldlst))
print("New  Elements:{}".format(sqrlist))
```

Program-3:

```
#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))
```

Program-4:

```
#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)
```

Program-5:

#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\t{}".format(on,round(sn,3)))
print("-"*50)
```

Program-6:

#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(oldlst,sqrootlist,sqlist):
        print("\t{}\t{}\t\t{}".format(on,round(sqrtno,2),sqrno))
print("-"*50)
```

Program-7:

#This program accepts list of Old Salaries of employees and gets new salary list with 15% hike.

```
#mapex7.py
print("Enter list of old slaries  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)
```

Program-8:

#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)
```

Program-9:

#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 who's 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("\tSal More Than 5000\t Hike Salaries")
print("="*50)
for sal,hsal in zip(salmore5000,hikesalmorethan5000):
        print("\t\t{}\t\t{}".format(sal,hsal))
print("="*50)
```

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

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

reduce()

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

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

=>Syntax:-

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

=>here varname is an object of int, float,bool,complex,str only

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

Internal Flow of reduce()

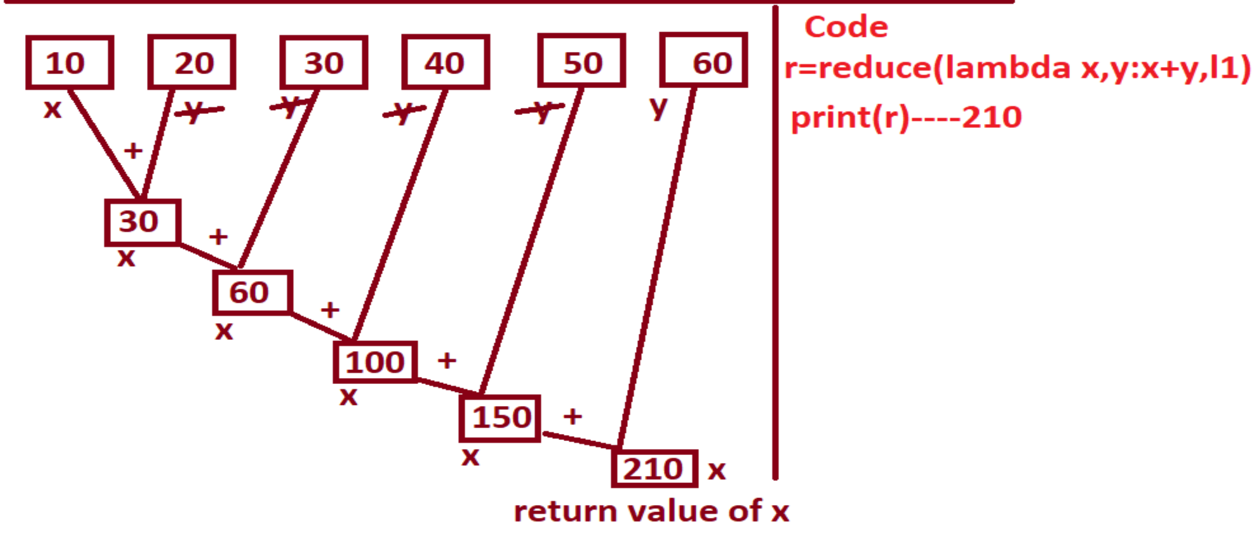
Step-1:- reduce() selects two First values of Iterable object and place them First var and Second var .

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

Let l1=[10,20,30,40,50,60] and Find Sum of list of elements.



Program-1:

#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)
```

Program-2:

#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))
```

Program-3:

#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))
```

Program-4:

#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))
```

==

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 mathsinfo ......etc

===

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-1:

```
#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))
```

Program-2:

```
#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-3:

```
# Program for displaying the calendar for year and month
#calendarex1.py
import calendar as c
print(c.month(2022,4))
```

Program-4:

```
# Program for displaying the calendar for year
#calendarex2.py
import calendar
print(calendar.calendar(2022))
```

Program-5:

```
#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
```

Program-6:

```
#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
```

Program-7:

```
#SE2.py---file name
from mathsinfo import *
print("val of pi=",pi)
print("val of e=",e)
print("val of k=",k)
```

Program-8:

```
#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 aliasname......module   name-n as alias name

----

=>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 "form" "import" are the key words

=>The purpose of from.... import statement is that " To refer or access the variable names, function names and class names in current program"

=> 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 "from ....import statement" , we must access variable names, Function names and class names Directly without using Module Names or alias names.

Variable Name

Function Name

Class Name

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

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

Program-1:
```
# 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-2:
```
# Program for displaying the calendar for year
#calendarex2.py
import calendar
print(calendar.calendar(2022))
```
Program-3:
```
# 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)
```

Program-4:
```
#city1.py---file name and treated as module name
def  hello(s):
       print("Hi {}, Good Morning:".format(s))
```

Program-5:
```
#city2.py--filr name--acts as module name
def  hello(s):
       print("Hello {}, Good Evening".format(s))
```

Program-6:
```
#citydemo.py--main program
from city2 import hello as e
from city1 import hello as m
```

```
m("Rossum")
e("Ritche")
```

Program-7:

```
#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)
```

Program-8:

```
#multabledemo.py---main program
from multable import table
table()
```

Program-9:

```
#This is a main program, where i am integrating all functions of different
modules
from aopmenu import aopmenu
from aopoperations import addop,subop,mulop,divop,expop,modop
import sys
while(True):
    aopmenu()
    ch=int(input("Enter ur Choice:"))
    if(ch==1):
        addop()
    elif(ch==2):
        subop()
    elif(ch==3):
        mulop()
    elif(ch==4):
        divop()
    elif(ch==5):
        modop()
    elif(ch==6):
        expop()
    elif(ch==7):
        print("Thanks for using this program")
        sys.exit()
    else:
        print("Ur selection of operation is wrong--try again")
```

Program-10:

```
def aopmenu():
    print("="*50)
    print("Arithmetic Operations")
    print("="*50)
    print("\t1.Addition")
    print("\t2.Substraction")
    print("\t3.Multiplication")
    print("\t4.Division")
    print("\t5.Modulo Div")
    print("\t6.Exponentiation")
    print("\t7.Exit")
    print("="*50)
```

Program-11:

```
def addop():
    a=float(input("Enter First Value for addition:"))
    b = float(input("Enter Second Value for addition:"))
    print("sum({},{})={}".format(a,b,a+b))
def subop():
    a = float(input("Enter First Value for Substraction:"))
    b = float(input("Enter Second Value for Substraction:"))
    print("sub({},{})={}".format(a, b, a - b))
def mulop():
    k = float(input("Enter First Value for Multiplication:"))
    v = float(input("Enter Second Value for Multiplication:"))
    print("mul({},{})={}".format(k, v, k * v))
def divop():
    k = float(input("Enter First Value for Division:"))
    v = float(input("Enter Second Value for Division:"))
    print("Div({},{})={}".format(k, v, k / v))
    print("Floor Div({},{})={}".format(k, v, k // v))
def expop():
    k = float(input("Enter Base:"))
    v = float(input("Enter Power:"))
    print("pow({},{})={}".format(k, v, k ** v))
def modop():
    k = float(input("Enter First Value for Modulo Division:"))
    v = float(input("Enter Second Value for Modulo Division:"))
    print("Mod({},{})={}".format(k, v, k %v))
```

==

reloading a modules in Python

==

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

=>Syntax:- imp.reload(module name)

(OR)

importlib.reload(module name) -----recommended

=>Purpose / Situation:

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

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

```
#shares.py---file and treated as module name
def sharesinfo():
        d={"Tech":19,"Pharma":11,"Auto":1,"Finance":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\t:{}".format(sn,sv))
        else:
                print("-"*50)
```

#main program

d=shares.sharesinfo() #previously imported module

disp(d)

time.sleep(15)

importlib.reload(shares) # relodaing previously imported module

d=shares.sharesinfo() # obtaining changed / new values of previously
                                         imported module

disp(d)

Program-1:

```
#shares.py---file name and acts as module name
def   sharesinfo():
       d={"IT":4,"Phamacy":2,"Auto":3,"Mrkt":2}
       return d
```

Program-2:

```
#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\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 the learn the following.
a) create a package
b) re-use the package
----

a) create a package:

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

Example:

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

==
b) re-use the package

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

i) By using sys module:

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

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

Example:

sys.path.append("E:\\KVR-PYTHON-11AM\\ACKAGES\\BANK")
(or)
sys.path.append("E:\KVR-PYTHON-11AM\ACKAGES\BANK")
(or)
sys.path.append("E:\KVR-PYTHON-11AM/ACKAGES/BANK")
----

ii)by using PYTHONPATH Environmental Variables:

=>PYTHONPATH is one of the Environmental Variable
Steps for setting PYTHONPATH=E:\KVR-PYTHON-11AM\PACKAGES\BANK
