PYTHON

Real time applications development by Python:

1. Web application development
2. GUI application development.
3. Console based applications.
4. Software development building tools.
5. Business applications.
6. Data science application with Artificial intelligence machine learning.
7. Scientific applications.
8. Audio & Video based applications.
9. Image processing applications.
10. 3D (CAD) based applications.
11. Networking with IOT (Internet Of Things)
12. Testing (selenium).
13. Healthcare sector.

What is PYTHON & History of PYTHON:

1. Python language is developed by "Guido Van Rossum" at CWI (Centrum Wiskunde & Informatica) in the Netherlands.
2. Python language was developed in "20th FEB 1991".
3. Python language is maintained by a Non-Commercial Organisation called "Python Software Foundation (PSF)". ~ 3.9.2 Version.
4. Python software can be downloaded Freely from www.python.org
5. Python language Doesn't backward compatibility.
6. Python language doesn't contain backward compatibility
   I.e, Python 3.x features are completely different from the 2.x version.
7. Python language has 2 versions
   - Python 2.x Versions-----Outdated,
   - Python 3.x Versions-----Currently Industries Using (CPYTHON)
     3.1, 3.2, 3.3, 3.4, 3.5, ……., 3.9, 3.9.2

Python language is inspired from;

- Functional Programming from **C.**
- Object Oriented Programming from **CPP (C++).**
- Scripting Programming from **PERL.**
- Modular programming language from **MODULO3.**

Companies are using Python:

Google, Instagram, Spotify, Netflix, DropBox, Facebook, Quora, Industrial Light and Magic.

Features of Python:

Features of Python are nothing but facilities or services provided by language developers (Rossum). Which are used by language programmers for developing real time applications.

1. **Simple and Easy**
   i) Python is a simple programming language because of 3 tech factors.
   - Python language provides a rich set of modules. So that python programmer can re-use the predefined code without our own code.

[**Definitions:**

Module: A module is a collection of Functions, Variables and Classes.

Ex., calendar, random, math, cmat, re, os, io,.....etc.

Garbage collector: A garbage collector is one of the software components in python software, which is running in the background of a regular python program and whose role is to collect / remove unused memory space.

Hence Garbage collector is taking care about automatic memory management.]

ii) Python programming provides an inbuilt facility called "Garbage collector", which collects unused memory space and improves the performance of python based applications.

iii) Python provides developer friendly syntaxes. So that we can develop error free programs in a limited span of time.

2. **Freeware and Open source:**
   i) Freeware: Python is downloaded in freely and hence it is freeware.
   ii)Open Source: The father of python is developed **"CPYTHON"** and it is treated as Standard Python Software.

Many companies customised Python distributions / flavours:

a) JPYTHON (or) JYTHON ---Used for running Java based applications.
b) IRON PYTHON----------Used for running C#.net applications.
c) MOCROPYTHON----------Used for developing Microcontrollers.
d) ANACONDA PYTHON-----Used for running Big data / Hadoop applications.
e) RUBY PYTHON----------Used for running Ruby based applications.
f) API's----------Use the collection of Modules

3. Dynamically typed programming language:

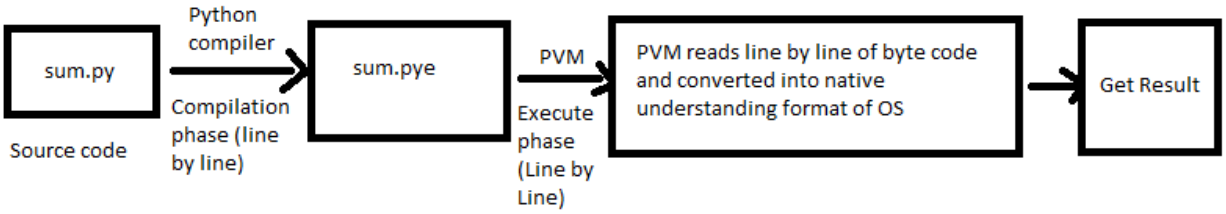
- In programming languages we have 2 types. They are,
  a) Static typed programming languages and
  b) Dynamic typed programming languages.

- In static typed programming language, it is mandatory to specify the datatype of the variable otherwise we get compile time error.
  Ex: C, CPP, Java,....etc

- In dynamically typed programming language, it is not necessary to specify the datatype of the variable.
  i.e., the Python environment takes care about assigning the data type to the variable automatically depending on the type value we place.
  Ex: Python
  >>>a=10
  >>>type(a)
  Output:
  <class 'int'>

4. Interpreted programming language:

- When we execute the python program, internally the source code (Ex: sum.py) is compiled with line by line conversion and this compiled version is called byte code (Ex: sum.yc). The intermediate code called bytecode (.pye) is ready by PVM (Python Virtual Machine) line by line and converted into native understanding format of OS.
- Consider the following diagram, it shows execution steps of a python program.



5. Platform Independent language:

- Concept: A language is said to be a platform if and only if whose applications runs on every OS.
- Property: "All the variables in python are treated as objects and they don't contain restrictions on size of data".

6. **High level programming language:**

The statements in the python program are looking like English statements and it is treated as “High level programming language”.

7. **Extensible:**

The other languages (C, C++, Java..etc.,) are integrating the code / snippets / scripts of python. The facility is called extensible.

8. **Embedded:**

The python code can also integrate the code of other languages. This property is called Embedded.

9. **Procedural (functional) and Object Oriented Programming Language:**

Python programs can be developed with functional programming and object oriented approaches and hence it is populated as “Functional and Object Oriented Programming language”.

10. **Portable:**

Python is one of the portable languages because applications / projects can run on any OS / platform without considering the hardware (processor) and software (OS) bench marks.

11. **Support third party API’s (scikit, scipy, numpy, matplotlib,....etc.)**

The python software can support third party API’s (scikit, scipy, numpy, matplotlib,....etc.) for developing artificial intelligence applications with machine learning by installing with a tool called PIP.

12. **Robust (strong):**

Python is one of the robust programming languages, because of an in-built programming facility called “Exception Handling”.

Exception: Runtime errors of a python program are called Exception.

- In general, Exceptions always generate technical error messages, which are not able to be understandable by end-users and they are understandable by programmers.
- Industry always recommends converting technical error messages into user-friendly error messages by making use of the exception handling concept.
- **Exception Handling:** The process of converting technical error messages into user-friendly error messages is known as exception handling.

PROGRAMMING FUNDAMENTALS IN PYTHON

1. Identifier's
2. Literal's
3. Object Referencing
4. Data types

Importance of Identifiers in Python:

- With the help of data types, Memory space can be allocated and data can be stored. To process the data which is stored in memory, The memory space must be given some Distinct/ Unique names and these unique names makes us identify the values present in memory and these distinct names are called Identifier's.
- Identifier's are also called Variables. Because identifier values can be changed during program execution.
- Hence in python, All values must be stored in memory in the form Identifier'/ Variables.

Definition of Variable:

A variable is one of the identifiers, Whose value can be changed during execution of the program.

Rules for using Identifier's/ Variable in Python:

- The variable name is a combination of alphabets, digits and special symbol underscore (_) only.
- The first letter of the variable name must start with either alphabat or underscore (_).

  Ex: >>> a=10-----Valid
  >>> _=20-----Valid
  >>>_$=40----Invalid
- Within the variable names, special symbols are not allowed except underscore (_).

  Ex:>>>tot sal=2.3-----Invalid
  >>>tot$sal=2.3-----Invalid
  >>>tot_sal=2.3-----Valid
- We should not use keywords as variable names (Because keywords are reserved words and give some specific meaning.)

  Ex: >>>if=20------Invalid
  >>>else=30---Invalid
  >>>if_=40----Valid
  >>>int=54----Valid (Int is a class, but not a keyword).
- There are no restrictions on size of the variable. (Recommended to take a short and sweet name).

- Variable names are case sensitive.

  Ex:>>>age=99-----Valid
  >>>AGE=100--Valid
  >>>Age=30-----Valid
  >>>a_g_e=40----Valid

Data Types in Python

- The purpose of data types to allocate memory space in main memory for storing input data
- In python, We have 14 data types and they are classified into 6 types.
  - **Fundamental / standard category data types**:
    - Int
    - Float
    - Bool
    - Complex
  - **Sequential categories data types:**
    - Str
    - Bytes
    - Bytearray
    - Range
  - **List category data types:**
    - List
    - Tuple
  - **Set category data types:**
    - Set
    - Frozenset
  - **Dict category data types:**
    - Dict
  - **None type category data types:**
    - None

I) Fundamental Category Data Types:

=>**purpose**:- To store a single value.
=>We have 5 fundamental data types . They are
a) int()
b) float()
c) bool()
d) complex()
e) str()

1) int():

=> This function is used for converting any valid other type value into int type value.
syntax:- varname2=int(varname1)
Examples:-

```
>>>a=12.34              # float value
>>>b=int(a)             # converting float value into int value
>>>print(b, type(b))------ 12   <class,'int'>

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

>>>a=2+3j
>>>b=int(a)---------------TypeError----unable to convert complex into int

>>>a="12"               # numerical String--possible to convert into int
>>>b=int(a)
>>>print(b, type(b))------12   <class, 'int'>
>>>a="ten"              # pure string---not possible to convert into int
>>>b=int(a)---------------ValueError--
>>>a="4x"                # alpha numeric String--not possible to convert
>>>b=int(a)---ValueError

>>>a="12.34"--------------float string--not possible to directly into int
>>>b=int(a)-----------------ValueError
```

Type casting Techniques in Python:

=>The process of converting one type value into another type value is called “Type Casting”
=>With this data type we can store diff values of Different Number Systems
=>We have 4 types of Number System. They are

a) Decimal Number System (default number System)

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

base:--- 10

b) Binary Number Systems

digits:- 0 ,1

base:- 2

c) Octal Number System:

digits:0,1,2,3,4,5,6,7

base: 8

d) Hexadecimal Number System:

digits: 0,1,2,3,4,5,6,7,8,9, A B C D E F

base: 16

b) Storing Binary Number Systems Data

=>To store Binary Number Systems data, in a python environment, binary data must be preceded with 0b or 0B.

=> **syntax:- varname=0b binary data**

Example: >>>a=0b1111

>>>print(a)----------15

>>>a=0B1010

>>>print(a)----------10

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

c) Storing Octal Number Systems Data

=>To store Octal Number Systems data, in a python environment, Octal data must be preceded with 0o or 0O.

=> **syntax:- varname=0o Octal data**

Examples:- >>>a=0o23

>>>print(a)----------19

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

>>>a=0o129----------inavlid, bcoz the digit 9 is not present in Octal number System

d) Storing Hexadecimal Number Systems Data

=>To store HexaDecimal Number Systems data, in a python environment, HexaDecimal data must be preceded with 0x or 0X.

=> **syntax:- varname=0x HexaDecimal data**

Example:

>>>a=0xAB

>>> print(a)----------171

```
>>> type(a)--------------<class 'int'>
>>> a=0xFaCE
>>> print(a)---------------64206
>>> type(a)---------------<class 'int'>
>>>a=oxACER----------invalid, the letter 'R' is not present in HexaDecimal Number System
```

Base Conversion Functions:

=>The purpose of Base Conversion Functions is that converting Decimal Number System into Binary , octal and HexaDecimal Number systems. We have 3 Base Conversion Functions. They are

a) bin()
b) oct()
c) hex()

a) bin()

=>This function is used for converting Decimal Number System data into binary number System data.

syntax:- varname= bin(decimal number System data)

Example:-

```
>>>a=15
>>> b=bin(a)
>>> print(b)--------0b1111
>>> a=10
>>> b=bin(a)
>>> print(b)-----0b1010
```

b) oct()

=>This function is used for converting Decimal Number System data into Octal number System data.

syntax:- varname= oct(decimal number System data)

Examples:

```
>>>a=19
>>>b=oct(a)
>>>print(b)--------0o23
```

c) hex()

=>This function is used for converting Decimal Number System data into HexaDecimal number System data.

Syntax:- varname= hex(decimal number System data)

Example:-

```
>>>a=171
>>>b=hex(a)
>>>print(b)-------0xAB   (or)  0xab
>>>print(hex(10))----------0xa
```

2) **float()**

=> This function is used for converting any valid other type value into float type value.

syntax:- varname2=float(varname1)

Example:

```
>>> a=12
>>> b=float(a)
>>> print(b, type(b))-----------12.0 <class 'float'>
>>> print(float(True))--------------1.0
>>> print(float(False))-------------0.0
>>> print(float("12.34"))---------12.34
>>> print(float(2+3j))-----can't convert complex to float
>>>print(float("4x.4y")---error
>>>print(float("12"))---------12.0
```

Note:-
```
print(int(0b1111))-----------------15
print(float(0b1010.0b1010))--------error
print(float(0b1111))-------------15.0
```

3)**bool()**

=>This function is used for converting any valid other type value into bool type value.

syntax:- varname2=bool(varname1)

Examples: (**HINT**:-Every **Non-Zero value is True** and **Zero value is False**)

```
>>>print(bool(10))-------------True
>>>print(bool(12.34))---------True
>>>print(bool(0))--------------False
>>>print(bool(0.0))------------False
>>>print(bool(2+3j))----------True
>>>print(bool("python"))-----True
>>>print(bool("0"))------------True
>>>print(bool("  "))------------True
>>>print(bool(""))-------------False
>>>print(bool("0.0"))---------True
```

4)**complex():-**

=>This function is used for converting any valid other type value into complex type value.

syntax:- varname2=complex(varname1)

Examples:

```
>>> a=10
>>> b=complex(a)
>>>print(b, type(b))--------------(10+0j) <class 'complex'>
>>>print(complex(10.5))---------(10.5+0j)
>>>print(complex(True))---------(1+0j)
>>>print(complex("12"))---------(12+0j)
>>> print(complex("12.35"))-----(12.35+0j)
>>>print(complex("python"))-----error---Non-Numerical strings can't
                                        convert into complex
>>>print(complex("3x.4y"))--------Error ----
```

5) **str()**

=>This function is used for converting any valid other type value into str type value.

syntax:- varname2=str(varname1)

Examples:

```
>>>a=10
>>>b=str(a)
>>>print(b, type(b))--------10  <class,'str'>
>>>print(str(12.34))--------'12.34'
>>>print(str(True))---------'True'
>>>print(str(2+4j))---------'2+4j'
```

II) Sequential Category data Types:

purpose:- To store Sequence of Values

=>This Category 4 data types. They are

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

1) str:

=>'str' is one of the pre-defined class

=>The purpose of this data type is to store string /character either in the form of a single line of text (or) multi line of text.

Organization / Storing String data:

=>We can store string data in two ways.

a) Single line text

b) Multi line text

a) Single line text:

Syntax:- ' single of text '

(or)

" single line of text"

Example:

```
>>>s1='Rossum'
>>>print(s1, type(s1)) ----------Rossum   <class,'str'>
>>>s2='R'
>>>print(s2,type(s2) )----------R  <class,'str'>
>>>s4="P"
>>>print(s4, type(s3))----------P  <class,'str'>
```

=>with Single or double quotes we can store only a single line of text but able to store a multi line of text.

b) Multi line of text:

syntax:- """ line of text-1
line of text-2
----
line of text-n """
(OR)
'''line of text-1
line of text-2
----
line of text-n '''

Example:-

```
>>>s1="""python is an oop language"
                (OR)
>>>s1='''python is an oop language'''

>>>print(s1, type(s1))----------python is an oop language   <class,'str'>
```

Operations on string data:

=>On String object data, we can perform two types of operations. They are

a) string indexing

b)string slicing

a) <u>string indexing</u>:

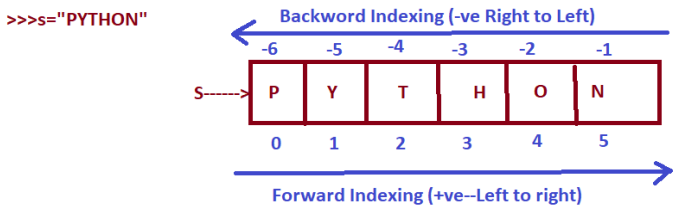
=>The process of obtaining a particular character from a given string by passing a valid index is called String Indexing.

Syntax:- **strobj[index]**

Here, index can be either +ve / -ve valid index.

if the index value is invalid, we get IndexError

=>Consider the following Statement and its memory management diagram.



Example:

```
>>>print( s[0])----P
>>>print(s[2])---- T
>>>print(s[4])---- O
>>>print(s[34])---IndexError
```

```
>>>print(s[-3])---- H
>>>print(s[-4])---- T
>>>print(s[-6])-----P
>>>print(s[-45])----IndexError
```

b)<u>String slicing</u>:

=>The process obtaining range of characters / substring from the given main String is called String Slicing.

Syntax:- **str object[begin:end]**

This syntax gives range of chars from begin index to end-1 index provided begin index < end index otherwise we never get any output(' ')

Examples:

```
>>>s="PYTHON"
>>>print(s[1:5])-----YTHO
>>>print(s[2:4])-----TH
>>>print(s[0:4])-----PYTH
>>>print(s[-6:-3])---PYT
>>>print(s[-5:-1])---YTHO
>>>print(s[12:1])---no output
>>>print(s[-6:-12])---no output
```

Special Points:

=>If we don't specify end index then python environment takes no.of chars-1 as end index position

=>If we don't specify begin index then python environment takes 0 (Initial Position) as begin index /position
=>If we don't specify both begin and end index then 0 (Initial Position) as begin index /position and no.of chars-1 as end index position

Example:-

```
>>>s="PYTHON"
>>>print(s[1:50])-----YTHON
>>>print(s[-100:-2])---PYTH
>>>print(s[1:])-----YTHON
>>>print(s[ : 4])----PYTH
>>>print(s[:])----PYTHON
```

2) **Bytes data type**:-

=>"bytes' is one of the pre-defined class
=>An object of bytes allows us to store sequence of +ve integer values with range (0,256), 0 is inclusive and 256 is exclusive
=>To convert one type value into bytes type value, we use bytes()
=>an object bytes is immutable bcoz 'bytes' object does not support item assignment

Examples:

```
>>>l=[10,20,256]
>>>b=bytes(l)-------error bcoz exceeds the range(0,256)
>>>l1=[10,20,255]
>>>b=bytes(l)
>>>type(b)----------<class, 'bytes')
>>> for x in b:
        print(x)------10 20  255
>>>print(b[0])-------10---- indexing allowed
>>>for x in b[0:3]:
        print(x)-------10  20---slicing allowed
>>>b[0]=122---------Error-- bcoz bytes object is immutable.
```

3) **Bytearray data types:**

=>"Bytearray' is one of the pre-defined class
=>An object of byte array allows us to store sequence of +ve integer values with range (0,256), 0 is inclusive and 256 is exclusive
=>To convert one type value into byte array type value, we use bytearray()
=>an object bytearray is immutable.
=>On the object of bytearray, we apply indexing and slicing.

Note:- The functionality of bytearray is exactly similar to bytes but an object of bytearray data type is mutable and bytes object is immutable.

```
Examples:--   >>>ls=[10,20,30,40]
              >>>ba=bytearray(ls)
              >>>print(type(ba))--------<class,'bytearray'>
              >>>for x in ba:
              >>>print(x)-----------  10 20   30 40
              >>>id(ba)-------------------xxxxxx248
              >>>ba[0]=12   # updating an object 'ba'
              >>>for x in ba:
              >>>print(x)-------------- 12  20  30  40
              >>>id(ba)-------------------xxxxxx248
```

4) **Range data type**:

=>'Range' is one the pre-defined class
=>The purpose of this data type is to store sequences of values by maintaining equal interval difference.
=>an object of range is immutable

Syntax-1:

1) range(start):

This syntax generates a range of values from 0 to start-1 values.

```
Example:-     >>>r=range(6)
              >>>print(type(r))-----<class,'range'>
              >>>print(r)------------range(0,6)
              >>> for x in r:
              >>>print(x)------- 0  1   2   3   4   5 (default step value is 1)
```

Syntax-2:

2) range(start,stop):

This syntax generates a range of values from start to stop-1 values.

Example:- Generate the values 10 to 20

```
              >>> for x in range(10,21):
                      print(x)---10  11  12  13  14..... 20
```

Example:- Generate the values 1000 to 1005

```
              >>>for x in range(1000,1006):
                      print(x)----  1000   1001   1002  1003  1004 1005
```

Syntax-3:

3) range(start,stop,step):

This syntax generates a range of values from start to stop-1 values with step value and can be +ve or -ve.

Example: Generate even numbers between 30 and 50

```
>>> for x in range(30,51,2):
        print(x)------- 30  32  34  36 38 ---  48  50
```

Example: Generate odd numbers between 50 and 40 in descending order

```
>>> for x in range(49,40,-2):
        print(x)-------49  47  45  43   41
```

Example: Generate -1 to -10

```
>>> for x in range(-1,-11,-1):
        print(x)--------1 -2  -3 ... -10
```

Example: generate -50 -40 -30 -20 -10

```
>>> for x in range(-50,-9,10):
        print(x)------  -50 -40 -30 -20 -10
```

Note:- On the object of range,we can apply indexing and slicing operations.

Example:-consider the following

```
>>>r=range(10,16)
```

For the above statement, the memory management diagram is given below

```
print(r[0])-----10
print(r[-6])-----  10
print(r[4])-------14
for x in r[2:5]:
print(x)------- 12 13 14
```

III) List category data types:

Purpose: To store multiple values of the same type or different type.

=>This List category contains 2 data types. They are,,

1) List
2) Tuple

1)List:

- ➔ List is one of the pre-defined class
- ➔ An object of list type allows us to store multiple values of the same type or different type of both types.
- ➔ An object of list type allows us to organise both unique and duplicate values.

- ➔ The elements of the list must be enclosed within square brackets [ ] and the values must be separated by comma (;)
- ➔ An object f list maintains insertion order.
- ➔ On the object of the list, we can perform both indexing and slicing.
- ➔ An object of list is mutable.
- ➔ To convert one type values into list type values, we use list().
- ➔ Empty list object can be created as follows

Example: >>>l1=[] or >>>l1=list()
>>>print(l1)----------[] <empty list>

Example:

```
>>> l1=[10,2,6,9,64,3,464,-20]
>>> l2=[12,"nav","lucky",98.3]
>>> print(l1,type(l1))----------[10, 2, 6, 9, 64, 3, 464, -20] <class 'list'>
>>> print(l2,type(l2))----------[12, 'nav', 'lucky', 98.3] <class 'list'>
```

Example:

```
>>> print(l1[0])----------10
>>> print(l1[1:4])----------[2, 6, 9]
>>> print(l2[1:4])----------['nav', 'lucky', 98.3]
>>> print(l2[2:4])----------['lucky', 98.3]
```

Functions in List:

1. append()
2. insert()
3. pop()
4. remove()
5. index()
6. copy()
7. reverse()
8. sort()
9. clear()
10. Extend()

1) Append() :-

- ➔ This function is used for adding an element to the object at the end of the list object.

Syntax: list obj.append(element)

2) Insert() :-

- ➔ This function is used for inserting an element in the list object by specifying valid existing position.

Syntax: listobj.insert(valid pos, element)

3) a) pop():-

➔ This function is used for removing an element from the list object by specifying a valid existing position.

Syntax:- list obj.pop(valid position)

3) b) pop()

➔ This function is used for removing the top most / latest element from the list object without specifying valid existing position.

Syntax:- list obj.pop()

Examples:-
```
>>> l1.append(10.3)
>>> print(l1)----------[10, 'rs', True, 10.3]
>>> l1.insert(2,"Naveen")
>>> print(l1)----------[10, 'rs', 'Naveen', True, 10.3]
>>> l1.insert(8,"KVR")
>>> print(l1)----------[10, 'rs', 'Naveen', True, 10.3, 'KVR']
>>> l1.append(123)
>>> print(l1)----------[10, 'rs', 'Naveen', True, 10.3, 'KVR', 123]
>>> l1.pop()----------123
>>> print(l1)----------[10, 'rs', 'Naveen', True, 10.3, 'KVR']
>>> l1.pop(3)----------True
>>> print(l1)----------[10, 'rs', 'Naveen', 10.3, 'KVR']
```

4) Remove():-

➔ This function is used for removing the specific element from list (first occurrence only) provided the element must present otherwise we get value error

Syntax:- listobj.remove(element)

Example:-
```
>>>l1=[10,20,10,20,30]
>>>l1.remove(20)
>>>print(l1)----------[10,10,20,30]
>>>l1.remove(100)-------Value error
```

5) Index() :-

➔ The function is used for finding an +ve index of a specified element of a list provided element present in the list otherwise we get Value error.

Syntax:- list obj.index(element)

Example:-
```
>>> l1=[10,20,10,20,30,40,-15]
>>> l1.index(20)----------1
>>> l1.index(30)----------4
>>> l1.index(200)----------ValueError: 200 is not in list
>>> l1.index(-15)----------6
```

6) **Copy() :-**

Points of Shallow copy:

➔ This function is used to copy the content of source list object into destination list object with same content with different address. This type of copy process is called Shallow copy.

Syntax:- dest.listobject=sourcelistobj.coy()

```
Example:-   >>>l1=[10,20,"HYD"]
            >>>l2=l1.copy()                #Shallow copy
            >>>print(l1,l2)---------[10,20,'HYD'] [10,20,'HYD']
            >>>print(id(l1),id(l2))-------xxxxx245        xxxxxx354
```

➔ Initial content of source and dest list objects are the same.

➔ Address of source and dest list objects are different

➔ Modification are reflected (Independent)

Points of Deep Copy:

➔ Initial content of source and dest list objects are the same.

➔ Modification are reflected to each other (Dependent)

Syntax:- destlistobj=sourcelist obj

```
Example:-   >>>l1=[10,"python"]
            >>>l2=l1        #Deep copy
            >>>print(l1,l2)------[10, 'python']     [10, 'python']
            >>>print(id(l1), id(l2)) ----------xxxxx245   xxxxxx354
            >>>l1.append("AI")
            >>>print(l1.l2)-------[10, 'python', 'AI']      [10, 'python', 'AI']
```

Special point : (Another way of shallow copy with slicing)

```
Example:    >>>l1=[10,"python"]
            >>>l2=l1[:]    #Shallow copy within slicing
            >>>print(l1.l2)-------[10, 'python', 'AI']      [10, 'python', 'AI']
            >>>print(id(l1), id(l2)) ----------xxxxx245   xxxxxx354
            >>>l1.append("India")
            >>>print(l1,l2)------[10, 'python', 'AI', 'India'] [10, 'python', 'AI']
```

7) **Reverse():-**

➔ This function is used for obtaining reverse of given elements of list objects.

Syntax:- listobj.reverse()

```
Example:-   >>> l1=[10,"HYD"]
            >>> l1.reverse()
            >>> print(l1)---------------['HYD', 10]
```

```
>>> l2=[10,2,-12,65,-65,-10,2]
>>> print(l2)----------------[10, 2, -12, 65, -65, -10, 2]
>>> l2.reverse()
>>> print(l2)----------------[2, -10, -65, 65, -12, 2, 10]
```

8) Sort():-

➔ This function is used for sorting the element of list objects in ascending order provided elements must be homogeneous.

Syntax:- listobject.sort()

Example:-
```
>>> l1=[10,2,-12,65,-65,-10,2]
>>> l1.sort()
>>> print(l1)---------[-65, -12, -10, 2, 2, 10, 65]        #Ascending
>>> l1.reverse()
>>> print(l1)---------[65, 10, 2, 2, -10, -12, -65]        #Descending


>>> l1=["nav", "vae", "swamy", "sai"]
>>> l1.sort()
>>> print(l1)--------['nav', 'sai', 'swamy', 'vae']
```

Special point:

```
>>> l2.sort(reverse=True)
>>> print(l2)--------[65, 10, 2, 2, -10, -12, -65]-------Directly display
```
in Descending order

9) Clear() :-

➔ This function is used for removing / clearing all the elements of list objects (Object remains same)

Syntax:- list obj.clear

Example:-
```
>>>l1=[10,20,30]
>>>print(len(l1)-------3
>>>l1.clear()
>>>print(len(l1)-------0
>>>print(l1)------------[ ]
```

Special points:

del() - is one of the pre-defined functions present in the built-in module and it is by default imported.

del() is used for deleting the values of any iterable object (list,str,...etc) and object also.

Syntax:- del(object name)

or

del(object name[index])

or

del(object name[start:sto])

Example:- >>>l1=[10,20,30]
>>>del(l1(10))
>>>print(l1)------[20,30]
>>>del(l1)
>>>print(l1)------Error ----> name 'l1' is not found

10) Extend() :-

➔ This function is used for adding the content of dest list object to the source list object content.

Syntax:- sourcelistobj.extend(destlistobj)

Example:- >>> l1=[10,20,30]
>>> l2=["python","AI"]
>>> l1.extend(l2)
>>> print(l1)---------[10, 20, 30, 'python', 'AI']
>>> print(l2)---------['python', 'AI']

Special Points:

We can use operator + for extending the functionality of different source list objects into deest list objects.

Syntax:- destlistobjj=sourcelistobj1+sourcelistobj2+sourcelistobj3+.....sourcelist obj n

Example:- >>>l1=[10,20]
>>>l2=["Python","Java"]
>>>l3=[True,2+3j]
>>>l4=l1+l2+l3
>>>print(l4)--------[10,20,'Python','Java',True,2+3j]

11) Count() :-

➔ This function is used for counting no.of occurrences of a specific element in a list object.

Syntax:- list obj.count(element)

Example: >>>l1=[10,20,10,10,-100]
>>>l1.count(10)----3
>>>l1.count(123)----0
>>>l1.count(-100)----1

Inner / Nested list():

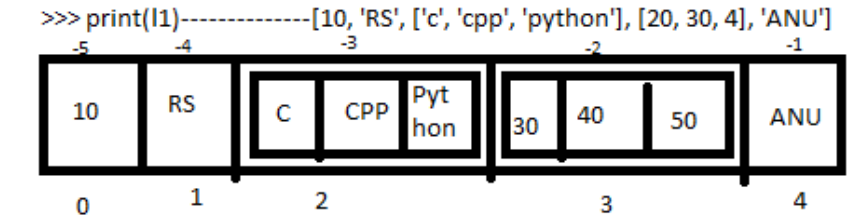
Def: The process of adding one list in another list is called inner / nested list.

Syntax:- list obj[elements of list, [elements of inner list-1].....[elements of inner list-n]

Example: Create a list with an inner list for representing student details as list elements, subjects as inner list and marks another inner list.

```
>>> l1=[10,"RS",["c","cpp","python"],[20,30,4],"ANU"]
```

The Diagrammatic representation for list an inner list is given below




```
>>>l1[0]-----10
>>>l1[2]-----[c,cpp,python]
>>>l1[2][-3:1]--------[c,cpp]
>>>l1[-2][-1]=99
>>>l1[2]----------[30,40,99]
```

Note: We can access all the functions of list with inner list also

Tuple:-

- ➔ Tuple is one of the pre-defined class
- ➔ An object of tuple type allows us to store multiple values of the same type or different type of both types.
- ➔ An object of tuple type allows us to organise both unique and duplicate values.
- ➔ The elements of the tuple must be enclosed within square brackets ( ) and the values must be separated by comma (,)
- ➔ An object f tuple maintains insertion order.
- ➔ On the object of the tuple, we can perform both indexing and slicing.
- ➔ An object of tuple is Immutable.
- ➔ To convert one type values into tuple type values, we use tuple().
- ➔ Empty tuple object can be created as follows

Example: >>>tp=() or >>>tp=tuple()
>>>print(tp)----------( )------Empty tuple

Example: >>>tp1=10,"HYD", 34.56
>>>print(tp1,tp(tp1))--------(10,'HYD',34.5)<class,"tuple">

Note:- On a tuple object, we use index() and count() of list objects but not other functions of list because tuple object belongs to immutable.

IV) Set category data types:

- ➔ This category data type allows us to store multiple values of the same type or different type with Unique values without maintaining Insertion order.
  a) Set
  b) Frozenset

a) SET:-

- ➔ 'Set' is predefined class
- ➔ An object of set allows us to store multiple values of the same type or different type and both types.
- ➔ An object of a set organizes only Unique elements.
- ➔ An object of set never maintains insertion order.
- ➔ Elements of set must be enclosed within curly braces { } and values separated by comma(,)
- ➔ An object of set is mutable (in the case of add()) and it is immutable in the case of "Item Assignment"
- ➔ On the object of set, we can't apply indexing and slicing operations because set object never maintains insertion order.
- ➔ To convert one type of elements into set type, we use set( )
- ➔ To create an empty set we use set( )

```
Example:    >>>s=set()
            >>>print(s,type(s))----------set() <Class 'set'>
```

Functions in SET:-

1) **add():**
   - ➔ This function is used for adding an element to set object

   Syntax:- set obj.add(element)

```
Example:-   >>>s={10,"Hyd"}
            >>>s.add("India")
            >>>print(s)----------------{10,'India', 'Hyd'}
```

2) **clear():**
   - ➔ This function is used for cleaning / removing all elements from set objects

   Syntax:- setobj.clear()

```
Example:-   >>>s1={10,20,"Hyd"}
            >>>len(s1)-----------3
            >>>s1.clear()
            >>>print(s1)---------set()--->empty
            >>>len(s1)--------0
```

3) **discard():-**

➔ This function is used for removing the specified value from set object and it won't return any deleted element (If element found the it is removed otherwise not deleted)

Syntax:- set obj.discard(value)

Example:- >>>s1={10,20,"Hyd","python"}
>>>s1.discard(10) #here 10 removed
>>>print(s1)--------{20,'Hyd','python'}
>>>s1.iscard(100) #here 100 not found and not deleted.

4) **Remove():-**

➔ This function is used for removing a value from a set object. If the value is unable to be removed we get "KeyError".

Syntax:- setobj.remove(value)

Example:- >>>s1={"apple","kiwi","banna"}
>>>s1.remove("apple")
>>>print(s1)-----{'kiwi','banna'}

5) **Pop():-**

➔ it is used for removing any random element from set object

Syntax:- set obj.pop()

Example: >>>s1={'e', 'u', 'a', 'i'}
>>>s1.pop()----------e
>>>print(s1)----------{'u', 'a', 'i'}

6) **issubset()**

➔ This function returns true provided one set is the subset of another set otherwise it returns False

Syntax:- setobj1.issubset(setobj2)

7) **issuperset()**

➔ This function returns true provided one set is the superset of another set otherwise it returns False

Syntax: setobj1.issuperset(setobj2)

Example:- >>>s1={10,20,30,40}
>>>s2={20,10}
>>>s1.issubset(s2)------False
>>>s2.issubset(s1)------True
>>>s1.issuperset(s2)-----True
>>>s2.issuperset(s1)-----False

Special cases:

```
>>>set().issubset(s1)------True
>>>s1.issubset(set())------False
>>>set().issubset(set())---True
```

8) isdisjoint():

➔ This function returns true provided both sets are having different elements. This function returns False provided both sets are having at least one common element.

Syntax: setobj1.isdisjoint(setobj2)

Example:
```
>>>s1={10,20,30,40}
>>>s2={"java","python"}
>>>s3={10,True}
>>>s4={"java",12.34}
>>>s1.isdisjoint(s2)--------True
>>>s1.isdisjoint(s3)--------False
>>>s1.isdisjoint(s4)--------True
```

9) union():

➔ This function obtains all the elements from those sets which are used in Union operation

Syntax:- setob3=set obj1.union(setobj2)

10) intersection():

➔ This function obtains common elements from any no of sets.

Syntax: setobj3=setobj1.intersection(setobj2)

11) Difference():

➔ This function is used obtaining only the elements of setobj1 by removing common elements from setobj1 and setobj2 (i.e., Exclusive element of setobj1)

Syntax:- setobj3=set obj1.difference(setobj2)

Example:-
```
>>>s1=setobj1.difference(setobj2)
>>>s1={10,20}
>>>s2={20,30}
>>>s3=s1.union(s2)
>>>s4=s1.intersection(s2)
>>>print(s4)------{20}
>>>s5=s1.difference(s2)
>>>print(s5)-------{10}
>>>s6=s2.difference(s1)
>>>print(s)----------{30}
```

Special cases:-

```
>>>print(s1|s2)----------{10,20,30}---here the symbol (|) is called BitWise OR
>>>print(s1&s2)----------{20}----------here the symbol (&) is called BitWise AND
>>>print(s1-s2)----------{10}
>>>print(s2-s1)----------{30}
```

12) **symmetric_difference():-**

➔ This function is used for obtaining exclusive elements of both the sets by removing common elements and placing those elements in the resultant set object.

Syntax:- resultant set obj=setobj1.symmetric_difference(setobj2)

Example:-
```
>>>s1={10,20,30,40}
>>>s2={30,40,45,65}
>>>s3=s1.symmetric_difference(s2)
>>>print(s3)---------(10,20,45,65}
```

13) symmetric__difference_update():

➔ This function is used for obtaining exclusive elements of both the sets and update source set objects by removing common elements and never place the element in the resultant set object.

Syntax:- sourcesetob.symmetric_difference_update(s2)

Example:-
```
>>>s1={10,20,30,40}
>>>s2={30,40,45,65}
>>>s3=s1.symmetric_difference_update(s2)
>>>print(s3)-------------None
>>>print(s1)------------{65,10,45,20}
```

14) update():-

➔ This function updates the elements of the source set with elements of the destination set object.

Syntax:- sourceset obj.update(destsetobj)

Example:-
```
>>>s1={10,20}
>>>s2={“Java”,”python”}
>>>s1.update(s2)
>>>print(s1)------------[20,10,’java’,’python’]
```

15) copy():-

➔ This function copies the elements source set object into destination set object (perform shallow copy)

Syntax:- destsetobj=sourceobj.copy()

Example:-
```
>>>s1={10,20}
```

```
>>>s2=s1.copy()
>>>print(s1,s2)--------[10,20] [10,20]
>>>s2.add("python")
>>>print(s1,s2)--------[10,20] [10,'python',20]
```

b) Frozen Set:-

- ➔ 'FrozenSet' is pre-defined class
- ➔ An object of the frozenset allows us to store multiple values of the same type or different type and both types.
- ➔ An object of a set organizes only Unique elements.
- ➔ An object of the frozenset never maintains insertion order.
- ➔ An object of frozenset is immutable
- ➔ On the object of the frozenset, we can't apply indexing and slicing operations.
- ➔ To convert one type of elements into frozenset type, we use frozenset( )
- ➔ To create an empty frozenset we use frozenset( )

Example:
```
>>>fs=frozenset()
>>>print(fs,type(fs))----------frozenset() <Class 'frozenset'>
```

Note:- The functionality of frozenset is similar to set but an object of frozenset belongs to immutable and object of set is both immutable (item assignment) and mutable (add()).

Note:- Frozenset contains the following functions.

a) issuperset()
b) issubset()
c) disjoint()
d) union()
e) intersection()
f) coy()
g) difference()
h) symmetric_difference()

V) Dict category Data type:-

Purpose:-

- ➔ This category contains only one data type. That is a) Dict.

a) Dict:-

- ➔ This data type allows us to store the data in the form of (Key, Value)
- ➔ In (Key,, Value), the values of key represents Unique and The values of value represents may or may not be Unique.
- ➔ The elements dict object must be enclosed within curly braces {} and whose (key, value) must be separated by colon (:) and terminated by comma (;).

- ➔ To convert one type of values into dict type we use dict () and it can also be used to create an empty dict object.
- ➔ An object of dict is Mutable.
- ➔ **Create a Dict:-**
  - ➔ We create two types of dict objects. They are
    a) Empty dict
    b) Non-Dict

Syntax for Empty dict: dictob={}
Or
dictobj=dict()

Example:- >>>d1={} (or)
>>>d1=dict()
>>>print(d1, type(d1))-------{} <class, 'dict'>

Syntax for adding (keys and values) to empty dict object:
dictobj[keyname-1]=value-1
dictobj[keyname-2]=value-2
dictobj[keyname-3]=value-3
----
dictobj[keyname-n]=value-n

Here if keyname is str type then keyname must be written within single / double Quotes. If keyname is numeric, we write keyname directly without quotes.

Example:- d1['stno']=123
d1[12]="OUCET"
print(d1)----------{'stno' : 123, 12 : 'OUCET'}
d1['place']="Hyd"
print(d1)----------{'stno' : 123, 12 : 'OUCET', 'place': 'Hyd'}

Syntax for creating non-empty dict:
dictobj={keyname-1:value1,keyname-2:value2,keyname-3:value3,.....keyname-n:value n}

Example:- d1={'stno':100, 12:"OUCET", 'place':"hyd"}
print(d1)-------{'stno':100, 12:"OUCET", 'place':"hyd"}

Updating the values of dict:

Syntax:- dictobj[keyname]=new value

- ➔ If the keyname is already existing in dict obj then the old value is replaced with a new value. If keyname is not existing in dict obj then Keyname, a new value is inserted newly in dict obj.

```
>>>d1={10:"AI", 11:"Python"}
>>>print(d1)------{10:"AI", 11:"Python"}
>>>d1[10]="java"
>>>print(d1)------{10:"java", 11:"Python"}  #updated dict
```

Functions in Dict:

1) **Get():**
   - ➔ This function is used for obtaining value of value from dict object by passing value of Key. if the key does not exist we get KeyError.

   Syntax: var=dictobj.get(key)

   Example:
   ```
   >>>d1={10:"OUCET", 11:"JNTUH"}
   >>>val=d1.get(10)
   >>>print(val)--------OUCET
   ```

2) **Values():**
   - ➔ This function is used for obtaining all the values of dict objects.

   Syntax: var=dictob.values

   Example:
   ```
   >>>keys=d1.keys()
   >>>print(keys)------[10,11]
   ```

3) **keys():**
   - ➔ This function is used for obtaining all the keys of dict objects.

   Syntax: var=dictob.keys

   Example:
   ```
   >>>keys=d1.keys()
   >>>print(keys)------[10,11]
   ```

4) **pop():-**
   - ➔ It is used for removing the (key,value) by passing value of key

   syntax: dictobj.pop(key)

   Example: d2.pop('qual')

5) **clear():**
   - ➔ This function clears/ removes all elements from the dict object and it becomes empty.

   Syntax: dictobj.clear()

   Example: d1.clear()

6) **update():**
   - ➔ This function updates the one dict obj elements with another dict object elements.

   Syntax: dictob1.update(dictobj2)

Here dictobj1 is updated with dictob2 elements

Example:
```
>>>d1={10,"OUCET"}
>>>d2={11:"JNTU",12:"KU"}
>>>d1.update(d2)
>>> print(d1)----------{11, 10, 'OUCET', 12}
>>> print(d2)----------{11: 'JNTU', 12: 'KU'}
```

VI) None category Data Types:

- ➔ This data type is used for string a value which is not any valid python data type value. I.e., None.
- ➔ The value None is of type <class,'None Type'>

  Example:
  ```
  >>>x=None
  >>>print(x,type(x))------None <class, 'None Type'>
  ```

No. Of approaches to develop python program:

To develop a python program, we have two approaches. They are

1) Interactive Mode approach
2) Batch Mode approach

1) **Interactive Mode approach:**
   - ➔ In this Mode of approach, a Python programmer writes a single statement and obtains a single result at a time.
     Example: Python cmd prompt
   - ➔ This approach is not recommended to develop a python program which contains a group of statements to solve a problem.

2) **Batch Mode approach:**
   - ➔ This approach always recommended to develop python based applications / program, where it can contain block of statements and those statements must be saved on some file name with an extension .py (source code)
   - ➔ The batch mode programming can be with the following.
     a) Python IDLE Shell
     b) Editplus (From 4.0 version)
     c) IDE's (Integrated Development Environment)
        i) pycharm
        ii) spyder
        iii) Jupiter, Anaconda, ….etc.

a) Batch Mode programming by using python IDLE shell:

- ➔ Python IDLE shell is coming along with python software installation steps:
  1. Launch Python IDLE shell
  2. Choose file → New file (Ensure that a window will be opened and there we can develop / write the program.
  3. Write the source and save it on file name with an extension .py (Ex:- First.py)
  4. Run the program. Choose Run→ Run module (F5) and ensure the result. OR
  5. Perform (1), (2) and (3) steps
  6. Run the python program by going to the command prompt and using the tool 'py'.
     Syntax:- python filename.py
     Example:- f:\python\python.py

b) By Using Edit Plus (From 4.0 version):-

Steps:

1. Launch Edt plus
2. Choose file → New → others → Python (click) [Ensure that window will be opened]
3. Write the source code and save it one some file name with an extension .py
   Ex: sum.py
4. Execute the python program by coming to the command prompt.
   Ex: f:\\python\py sum.py [ensure that we get result]

Reading the data from the Keyboard:

➔ To read the data from the keyboard, in python we have 2 pre-defined functions. They are,

a) inut()
b) input(message)

a) Input():

➔ This function is used for the reading any type of data dynamically from keyboard and returns in the form of 'str'

➔ **Syntax: varname=input()**

b) input(message):

➔ This function is used for reading any type of data dynamically from keyboard and returns in the form str by prompting the message to end user on the console

➔ **Syntax: varname=input(user prompting message)**

➔ Ex: print("Enter any value:")
a=input() or
a=input("Enter any value:")

★ **1) <u>Program Input</u>**

```
#Write a python program which acceptable integer values and add them
#add.py
a=float(input("Enter value of a\n"))
b=float(input("Enter value of b\n"))
c=a+b
print("-"*40)
print("Result")
print("-"*40)
print("value of a=",a)
print("value of b=",b)
print("Sum=",c)
print("-"*40)
```

Output

```
============ RESTART: E:/Naresh IT K.V.Rao sir class programs/add.py ===========
Enter value of a
10
Enter value of b
20
----------------------------------------
Result
----------------------------------------
value of a= 10.0
value of b= 20.0
Sum= 30.0
----------------------------------------
```

★ **2) Program:**

```
#Write a Python program which will calculate simple intrest and the total amount to pay
p=float(input("Enter principle amount in (Rupees):"))
t=float(input("Enter Time in (Months):"))
r=float(input("Enter Rate of intrest in (%):"))
si=(p*t*r)/100
amountpay=p+si
#Display the result
print("-"*50)
print("Result")
print("-"*50)
print("Principle Amount:",p)
print("Time:",t)
print("Intrest:",r)
print("-"*50)
print("Simple intrest in amount:",si)
print("Total amount to pay:",amountpay)
print("-"*50)
```

Output:

```
Enter principle amount in (Rupees):1000
Enter Time in (Months):2
Enter Rate of intrest in (%):12
--------------------------------------------------
Result
--------------------------------------------------
Principle Amount: 1000.0
Time: 2.0
Intrest: 12.0
--------------------------------------------------
Simple intrest in amount: 240.0
Total amount to pay: 1240.0
--------------------------------------------------
```

★ **3) Program Input:**

Write a python program which will accept student details from the keyboard and display the student result:

```
#Write a Python rogram which accept Student data & Display the student details
sno=int(input("Enter Student Number:"))
sname=input("Enter Student Name:")
smarks=float(input("Enter Student Marks:"))
cname=input("Enter College Name:")
#Display Employee details
print("-"*50)
print("Employee Details")
print("-"*50)
print("Student Number:{}".format(sno))
print("Student Name:{}".format(sname))
print("Student Marks:{}".format(smarks))
print("College Name:{}".format(cname))
print("-"*50)
```

Output:

```
= RESTART: E:/Naresh IT K.V.Rao sir class programs/print student details input :
rom keyboard.py
Enter Student Number:24
Enter Student Name:Naveen Kumar
Enter Student Marks:7.0
Enter College Name:Prakasam Engineering College
--------------------------------------------------
Employee Details
--------------------------------------------------
Student Number:24
Student Name:Naveen Kumar
Student Marks:7.0
College Name:Prakasam Engineering College
--------------------------------------------------
```

4) Program

```
#Write a Python rogram which accept employee data & Display the employee details
eno=int(input("Enter Employee Number:"))
ename=input("Enter Employee Name:")
esalary=float(input("Enter Employee Salary:"))
edisignation=input("Enter Employee Disignation:")
#Display Employee details
print("-"*50)
print("Employee Details")
print("-"*50)
print("Employee Number:{}".format(eno))
print("Employee Name:{}".format(ename))
print("Employee Salary:{}".format(esalary))
print("Employee Designation:{}".format(edisignation))
print("-"*50)
```

Output

```
= RESTART: E:\Naresh IT K.V.Rao sir class programs\4. print employee details,
ut takn from keyboard.ppy.py
Enter Employee Number:123
Enter Employee Name:Naveen kumar
Enter Employee Salary:2100.2
Enter Employee Disignation:Developer
--------------------------------------------------
Employee Details
--------------------------------------------------
Employee Number:123
Employee Name:Naveen kumar
Employee Salary:2100.2
Employee Designation:Developer
--------------------------------------------------
```

5) Program Input

```
#write a Python Program to convert temperature in celsius to fahrenheit
celsius = float(input("Enter The temperature in Celcius:"))

# calculate fahrenheit
fahrenheit = (celsius * 1.8) + 32
print("-"*50)
print('{}.1f degree Celsius is equal to {}.1f degree Fahrenheit'.format(celsius,fahrenheit))
```

Output

```
= RESTART: E:\Naresh IT K.V.Rao sir class programs\5. celsius to faurenheat temp
erature.py
Enter The temperature in Celcius:97
--------------------------------------------------
97.0.1f degree Celsius is equal to 206.6.1f degree Fahrenheit
```

OPERATORS

- ➔ An operator is a symbol, which is used to perform some operations
- ➔ If the operator is connected with two or more variables/ operands/ literals then it is called expression.
- ➔ In other words, An expression is a collection of variables / operands/ literals connected with an operator.
- ➔ In python programming, We have different types of operators. They are,
    1) Arithmetic operators
    2) Assignment operators
    3) Relational operators
    4) Logical operators
    5) Bitwise operators
    6) Membership operators
    7) Identify operators

1) Arithmetic operators:

- ➔ These operators are used for performing arithmetic operations i.e., (add, sum, mul,..etc)
- ➔ If arithmetic operators are connected with literals / operands / variables then it is called arithmetic expression.
- ➔ The following gives a list of arithmetic operators.

Arithematic Operators

S.No	Symbol	Meaning	Example	a=10; b=3
1	+	Addition	c=a+b	13
2	-	Substraction	c=a-b	7
3	*	Multiplication	c=a*b	30
4	/	Division (Float quotient)	c=a/b	3.33
5	//	Floor Division (Integer quotient)	c=a//b	3
6	%	Modula division (Reminder)	c=a%b	1
7	**	Exponentation	c=a**b	1000

6) Program input

Write a python program using Arithmetic operators:

```
#Arithematic Oprations
a=int(input("Enter a value of a:"))
b=int(input("Enter a value of b:"))
print("-"*30)
print("Arithematic Operators")
print("-"*30)
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))
```

Output

```
Enter a value of a:9
Enter a value of b:5
------------------------------
Arithematic Operators
------------------------------
9+5=14
9-5=4
9*5=45
9/5=1.8
9//5=1
9%5=4
9**5=59049
```

2) Assignment operators:

- ➔ The symbol of assignment operator is "="
- ➔ This operator is used for assigning RHS value / expression value to the LHS variable
- ➔ In python programming two types of assignment operators.
  1) Single line assignment
  2) Multi line assignment

Syntax: LHS variable=RHS var/value/literal (or) LHS variable=RHS expression

Example:

```
>>>a=10
>>>b=a
>>>x=10
>>>y=20
>>>z=x+y
```

Syntax for Multi line assignment:

LHS var1, LHS var2,LHS var3,...=RHS var/exp1,RHS var/exp2,RHS var/exp3,....

Example:

```
>>>x,y=1,20
>>>print(x,y)---------10        20
>>>a,b,c=x+y,x-y,x*y,x/y,x//y
```

7) <u>Program Input</u>:

Write a python program which will accept any two values and interchange them without using third variable

```
#Write a python program which will accept any two values and interchange them wi
a=input("Enter First value")
b=input("Enter second value")
print("-"*50)
print("Original value of a:{}".format(a))
print("Original value of b:{}".format(b))
print("-"*50)
a,b=b,a #Multiline assignment
print("Swapped value of a:{}".format(a))
print("Swapped value of b:{}".format(b))
```

<u>Output:</u>

```
= RESTART: E:/Naresh IT K.V.Rao sir class programs/7. Swapping two values withou
t 3rd variable.py
Enter First value6
Enter second value5
--------------------------------------------------
Original value of a:6
Original value of b:5
--------------------------------------------------
Swapped value of a:5
Swapped value of b:6
```

3) Relational operators:

- ➔ The purpose of these operator is that compare two values of variables
- ➔ If two operands are connected with relational operators then it is known as relation expression.
- ➔ The following table list of relational operators.

s.no	symbol	meaning	Example	a=1, b=20, c=10
1	>	Greater than	print(a>b)	FALSE
2	<	Less than	print(a<b)	TRUE
3	==	Equality	print(a==c)	TRUE
4	!=	Not equal	print(a!=b)	TRUE
5	>=	Greater than or Equal to	print(a>=b)	FALSE
6	<=	Less than or Equal to	print(a<=c)	TRUE

8) Program Input:

Write a python program which demonstrate the concept of relational operators

```
#Relational operators
a=int(input("Enter value of a:"))
b=int(input("Enter value of b:"))
print("-"*30)
print("{}>{}={}".format(a,b,a>b))
print("{}>{}={}".format(b,a,b>a))
print("{}<{}={}".format(a,b,a<b))
print("{}<{}={}".format(b,a,b<a))
print("{}=={}={}".format(a,b,a==b))
print("{}!={}={}".format(a,b,a!=b))
print("{}>={}={}".format(a,b,a>=b))
print("{}>={}={}".format(b,a,b>=a))
print("{}<={}={}".format(a,b,a<=b))
print("{}<={}={}".format(b,a,b<=a))
```

Output

```
== RESTART: E:/Naresh IT K.V.Rao sir class programs,
Enter value of a:5
Enter value of b:3
------------------------------
5>3=True
3>5=False
5<3=False
3<5=True
5==3=False
5!=3=True
5>=3=True
3>=5=False
5<=3=False
3<=5=True
```

4) Relational Operators:-

- ➔ The purpose of Logical operators is to combine two or more relational expressions.
- ➔ Logical operators are used for forming compound conditions (multiple condition.
- ➔ If two or more relational expressions are connected with logical operators then it is called Logical expression.

Syntax: (rel expr1) logical operator (rel expr2)

Rel Exp-1	Rel Exp-2	RelExp1 AND Rel Exp-2	Rel Exp-1 OR RelExp-2
FALSE(0)	FALSE(0)	FALSE	FALSE
TRUE(1)	FALSE(0)	FALSE	TRUE
FALSE(0)	TRUE(1)	FALSE	TRUE
TRUE(1)	TRUE(1)	TRUE	TRUE

The following table gives logical operators:

Sl no	Symbol	Meaning	Example	a=10, b20, c=10
1	and	Physical ANDing	(a<b)and(a<c)	FALSE
			a<b) and (a==c)	TRUE
2	or	Physical ORing	(a>b) or (a<c)	TRUE
			(a>b) or (a>c)	FALSE
3	not	not true--> false	not(20==10)and (10!=5)	TRUE
		not false-->True	not(20!=10)or (10>5)	FALSE

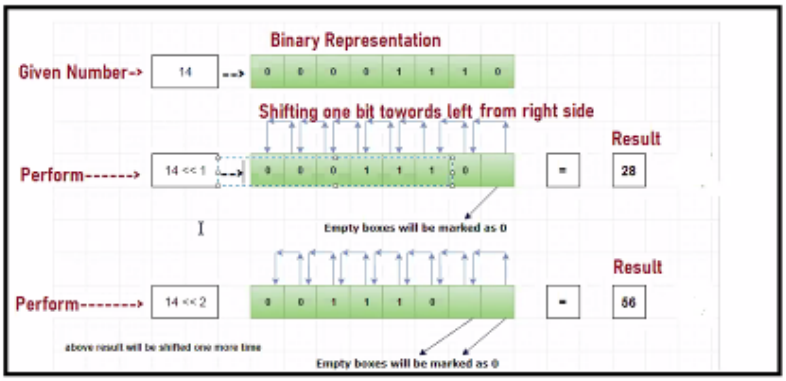
5) Bitwise operators:(written test)

- ➔ In python, Bitwise operators are used to perform bitwise calculations on integer data.
- ➔ When we use Bitwise operators on integer data, first integer data converted into binary format and then operations are performed on the data bit by bit and hence these operators are named as bitwise operators.
- ➔ Bitwise operators applicable only on Integer data but not on Floating point values.
- ➔ We have:
  1) Left shift operator(<<)
  2) Right shift operator(>>)
  3) Bitwise AND operator (&)
  4) Bitwise OR operator (|)
  5) Bitwise XOR operator (^)
  6) Bitwise complement operator (-)

1) Left shift operator(<<):

- ➔ **Syntax: resultant variable = given number << no.of bits**
- ➔ **Concept:** Left shift operator (<<) shifts the specified number of bits of a given number to the left and fills zeros (0) on voids and leaves as a result.
- ➔ (Or) This operator shifts the specified number of bits towards the left side by adding no. of zeros (equal to no. of bits) at right side.

Diagramatic Representation Of Left shift operator (< <)



Formula:

general statement of left shift operator

result = number << no. of bits ==========> result = number x $2^{\text{no. of bits}}$

Example:- result= 12<<3====> 12 x 2^{3} => 12 x 8 = 96

print(result)----> 96

2) **Right shift operator(>>):**

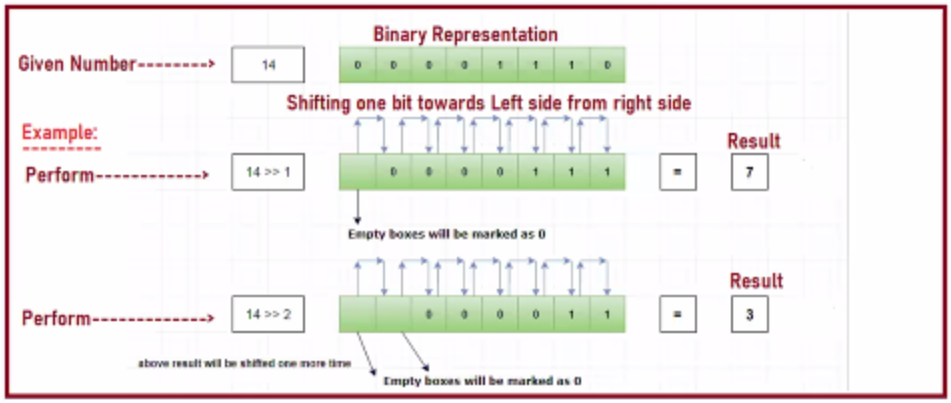
➔ **Syntax: res var = Given number >> no.of bits**

Here given number can be +ve or -ve

No.of bits must be +ve (but not +ve because -ve no.of bits cant shifted)

➔ **Concept:** This operator shifts specified number of bits towards right side by adding no.of zeros (equal to no.of bits) at left side

Diagramatic Representation of Right Shift Operator(>>):



Formula for Right shift operator(>>):

result = Given Number >> no. of bits

$$result = \frac{Given\ Number}{2^{no.\ of\ bits}}$$ (OR) in python Givem Number // $2^{no.\ of\ bits}$

>>>print(10>>2)----------$10//2^2$ -----> 10 //4 = 2

3) **Bitwise XOR operator (^):**
   - ➔ **Syntax: res = var1 ^ var2**
   - ➔ **Concept:** Similar bits of var1 and var2 is 0
     Dissimilar bits of var1 and var2 is 1

Truth table XOR (^)

var1	var2	var1 ^ var2
0	0	0
1	0	1
0	1	1
1	1	0

Example:-

```
>>>a=4----------------> 0 1 0 0
>>>b=3----------------> 0 0 1 1
>>>c=a^b---------->    ------------
                        0 1 1 1----------7
                       --------------
>>>print(c)------->7
```

Example:

```
>>>a=14---------------- 1 1 1 0
>>>b=15---------------- 1 1 1 1
>>>c=a^b------------   ------------
                       0 0 0 1----------1
>>>print(c)-------- 1
```

Special Case: - Swapping of numbers

Example:- (Orginal valoe of a =4 original val of b=3)

```
>>>a=4---------- 0 1 0 0
>>>b=3---------- 0 0 1 1
>>>a=a^b-------- 0 1 1 1--------7--->val of a
-----------------------------------------
>>>b=a^b---------- 0 1 1 1
                  0 0 1 1
                  ---------
                  0 1 0 0--------4--->val of b
>>>a=a^b----------> 0 1 1 1
                   0 1 0 0
                  --------------
                   0 0 1 1-------3-->val of a
>>>print(a)-----3--swapped
>>>print(b)-----4----swapped
```

9) Program input:

Write a python program which will swap two integer values

```
#Swap two values
x=int(input("Enter value of x:"))
y=int(input("Enter value of y:"))
print("-"*40)
print("Original value of x {}".format(x))
print("Original value of y {}".format(y))
print("-"*40)
x=x^y
y=x^y
x=x^y
print("Swapped value of x={}".format(x))
print("Swapped value of y={}".format(y))
print("-"*40)
```

Output

```
= RESTART: E:\Naresh IT K.V.Rao sir class p
rograms\9. Swapping of two values using Bit
wise XOR operator.py
Enter value of x:5
Enter value of y:3
----------------------------------------
Original value of x 5
Original value of y 3
----------------------------------------
Swapped value of x=3
Swapped value of y=5
----------------------------------------
```


4) **Bitwise AND operator (&):**
   - ➔ **Syntax: res = var1 & var2**
   - ➔ **Concept:** If both the bits of var1 and var2 is 1 then result is 1 Otherwise result is 0

```
Truth table for AND  (&)
-------------------------------------------------
var1    var2        var1 & var2
-------------------------------------------------
0       0           0
1       0           0
0       1           0
1       1           1
-------------------------------------------------

Example:-
          >>>a=4-------------  0 1 0 0
          >>>b=3-------------  0 0 1 1
          >>>c=a&b--------    0  0 0 0--------0
          >>>print(c)---------0
==================================
```

5) **Bitwise OR operator(|):**
   - ➔ **Syntax: res = var1 | var2**
   - ➔ **Concept:**This operator returns 1 provided if any one of bits of var1 or var2 is 1. Otherwise returns 0.

```
Truth table for OR  ( | )
-------------------------------------------------------------------------
var1    var2        var1 | var2
-------------------------------------------
0       0           0
1       0           1
0       1           1
1       1           1
-------------------------------------------------
Example:-
-------------     >>>a=4-------------> 0100
                  >>>b=3-------------> 0011
                  >>>c=a|b ----------> 0111--------->Result is 7
                  >>>print(c)------------7
Example:
-------------     >>>a=14------------>1110
                  >>>b=15-----------> 1111
                  >>>c=a|b----------->  1111-----Result is 15
                  >>>print(c)-------->15
```

6) **Bitwise Complement Operator(~):**
   - ➔ **Syntax: res var = ~var**
   - ➔ **Concept:** This operator returns one's complement of a given number. (Or) res var = -(var+1)

```
Example:
----------      >>>a=10----------------> 1010
                >>>res=~a-------------> ~1010----> -(1010+1)
                                                   -(1011)--->Result is -11
                >>>print(res)------> -11
Example:
                >>>print(~-99)------------>98
                >>>print(~102)-------------> -103
                >>>print(~(100+1))--------->-102
```

6) Membership operator:

- ➔ **Purpose:** The purpose of Membership operators is that "To check the extension of value in an iterable object". If the element / value present in iterable object then we get "True" otherwise we get "False"
- ➔ We have 2 types of membership operators in python. They are
  a) In
  b) Not in

a) In:

- ➔ **Syntax: value in table_object**
- ➔ "In" operator returns True provided specified value present in iterable_object otherwise it returns False.

```
Example: >>> fruits_bag=["apple","cherry","guava","kiwi"]
         >>> "apple" in fruits_bag-------->True
         >>>"orange" in fruits_bag------->False
         >>>'a' in fruits_bag[0]---------->True
         >>>"cherry" in fruits_bag[1:3]---->True
         >>>name="koushal tata"
         >>>"tata" in name--------True
         >>>"shal" in name--------True
```

b) Not in:

- ➔ **Syntax: value not in iterable_object**
- ➔ "Not in" operator returns True provided specified value not present in iterable_object. Otherwise it returns False.
- ➔ **Example:** >>>s1={"AI", "PYTHON", "JAVA"}
  >>>"python" not in s1----------> True
  >>>s1[2] not in s1----------> False
  >>>s1[2][1:3]in s1[2][:]----------> True

7) Identify operators:

- ➔ The purpose of Identify operators is that "To check the memory addresses of two objects".
- ➔ We have two types of identity operators. They are

1) Is
2) Is not

1) Is:

- ➔ **Syntax: var1 is var 2**
- ➔ This operator returns True provided both var1 and var2 points have the same memory address. (id(var1) and id(var2) is different) otherwise it returns False.

2) Is not:

- ➔ **Syntax: var1 is not var 2**
- ➔ This operator returns True provided both var1 and var2 points have the different memory address. (id(var1) and id(var2) is different) otherwise it returns False.

Example:

```
>>>a=234
>>>b=234
>>>a is b -------->True (0-256)
>>>a is not b-------->False
>>>a=257
>>>b=257
>>>a is b--------> False(More than 256)
>>>a is not b------>True
>>>a=-5
>>>b=-5
>>>a is b--------> True(-1 to -5)
>>>a is not b------>False
```

Flow controls (or) Control structures in Python

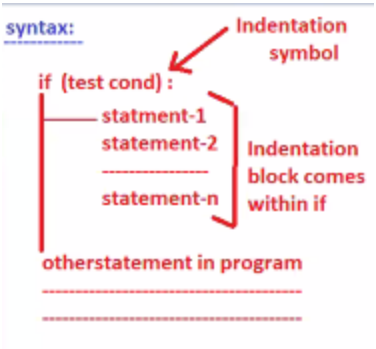
Purpose:

- ➔ Control structures are used for performing either one time operation depending on condition evaluation (True or false) or performing certain operations repeatedly for a finite number of times until condition is false.
- ➔ In python, We have 3 types of flow controls. They are
  a) Conditional / selectional / branching statements (if, if - else, if - elif - else)
  b) Looping / iterative / repetitive statements
  c) Misc. control structures (Break, Continue, Pass)

a) Conditional / selectional / branching statements:

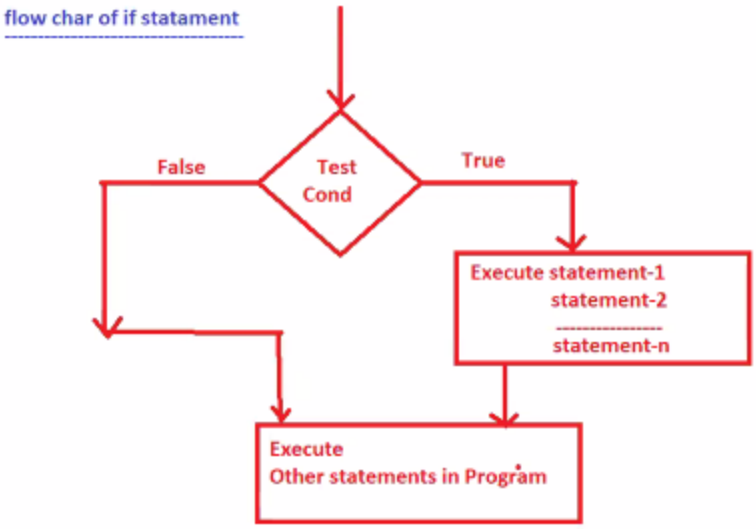
- ➔ To perform a certain operation for one time depends on condition evaluation. In other words, if the condition is true the perform x- operation once or if the condition is false then perform y- operation once.
- ➔ We have 3 conditional statements. They are,
  i) Simple if statement
  ii) if - else statement
  iii) if - elif - else

i) Simple if statement:



Explanation:

- ➔ Here "test condition" evaluates first.
- ➔ If the test condition is true then Python Virtual Machine executes statement-1, statement-2, …….statement-n, (known as block of statements) and also executes other statements in the program.
- ➔ If the test condition is false then Python Virtual Machine executes other statements in the program only without executing a block of statements.

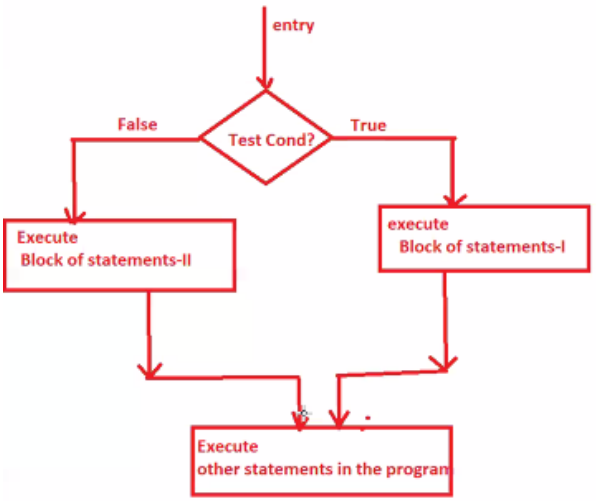


ii) If-Else statement:

Syntax: **if (test cond):**
Block of statement-1
Else:
Block of statement-2
Other statements in program
……

Explanation:

- ➔ If the condition is true then execute Block of statement-1 and also execute other statements in the program.
- ➔ If the condition is False then execute Block of statement-2 and also execute other statements in the program



10) Program input:

Write a python program which will accept 3 integer values and find the biggest among them:

```
#biggest of given 3 integer values
a=int(input("Enter 1st number:"))
b=int(input("Enter 2nd number:"))
c=int(input("Enter 3rd number:"))
big=a
if(b>big):
    big=b
    if(c>big):
        big=c
        print("Biggest={}".format(big))
    else:
        print("Biggest={}".format(big))
else:
        print("Biggest={}".format(big))
```

(or)

```
#biggest of given 3 integer values
a=int(input("Enter 1st number:"))
b=int(input("Enter 2nd number:"))
c=int(input("Enter 3rd number:"))
if(a==b)and (b==c):
    Print("All values are equal")
else:
    big=a
    if(b>big):
        big=b
    if(c>big):
        big=c
    print("Biggest of ({}, {}, {} is {})".format(a,b,c,big))
```

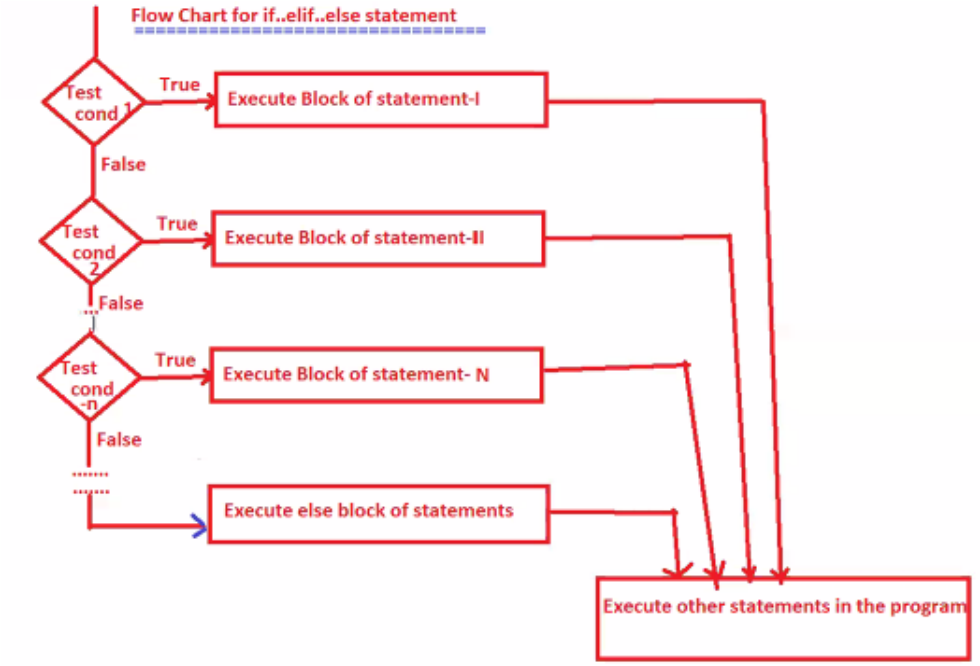
Output:

```
= RESTART: E:/Naresh IT K.V
.Rao sir class programs/10.
Biggest value of given 3 in
tegers.py
Enter 1st number:5
Enter 2nd number:6
Enter 3rd number:7
Biggest=7
```

iii) if - elif - else:

syntax:

```
----------
        if (test cond-1):
              execute Block of Staments-I
        elif(test cond-2):
              execute Block of statement-II
        elif(test cond-3):
              execute Block of statement-III
        -------------------------
        -------------------------
        elif(test cond-n):
              execute Block of statement-N
        else:
              execute else of Block of statement
        ----------------------------
        ----------------------------
        Other Statements in the program
        ---------------------------
        ---------------------------
```



Explanation:

- ➔ Here test condition 1 is evaluated. If test condition 1 result is True then PVM will execute Block of statements 1 and other statements in the program.
- ➔ If test condition 1 is False and PVM control goes to test condition 2 and if test condition 2 is True then execute Block of statements 2 and other statements in the program.
- ➔ If test condition 2 is False and PVM control goes to test condition 3 and if test condition 3 is True then execute Block of statements 3 and other statements in the program.

➔ This process will continue until all test conditions are evaluated.

➔ If all test conditions are False then execute else block of statements which are written under else block and also execute other statements in the program.

11) <u>Program Input</u>:

Write a python program which will accept digits 0-9 and prints its name

```
#Digits
d=int(input("Enter any digit:"))
if(d==0):
    print("Zero")
elif(d==1):
    print("ONE")
elif(d==2):
    print("TWO")
elif(d==3):
    print("THREE")
elif(d==4):
    print("FOUR")
elif(d==5):
    print("FIVE")
elif(d==6):
    print("SIX")
elif(d==7):
    print("SEVEN")
elif(d==8):
    print("EIGHT")
elif(d==9):
    print("NINE")
else:
    print("It is not a digit")
print("\nI am other part of this program")
```

<u>Output:</u>

```
= RESTART: E:/Naresh IT K.V.Rao sir class
programs/11. Print given digit usinf (if.
. elif..else).py
Enter any digit:6
SIX

I am other part of this program
```


b) Looping/ iterative / repetitive statements:

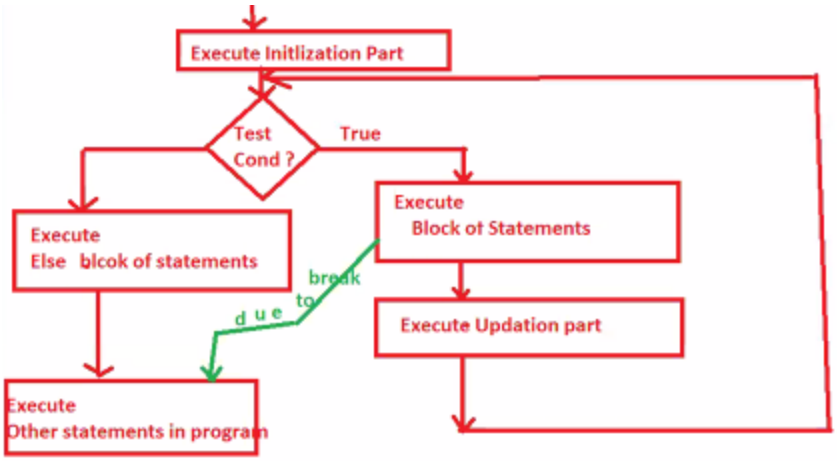
- ➔ The purpose of looping statements is to perform / execute certain operations repeatedly for a finite number of times.
- ➔ We have 2 types of looping statements. They are,
  1) while (or) while---else
  2) for (or) for---else
- ➔ At the time of dealing with looping statements, the programmer must ensure that there must exist 3 parts. They are,
  a) Initialisation part
  b) Condition part
  c) Updation part (increment / decrement)

1) <u>While (or) while...else</u>:

Syntax:

```
----------
  ----------
  ----------
while (Test cond):                    (OR)    ----------
                                              ----------
                                              while(Test Cond):
    statement-1                                   statement-1
    statement-2                                   statement-2
    ----------                                    ----------
    ----------                                    ----------
    statement-n                                   statement-n
----------                                    else:
----------                                        else block of stsments
Other statements in the program               ----------
----------                                    Other statements in the program
```

<u>Flow chart</u>



Explanation:

- ➔ Here test condition evaluated. If test condition is true then execute statement 1, statement 2…. Statement n (block of statements) and PVM goes to test condition again. Again if the test condition is true then again execute the block of statements. Hence the block of statements will execute for a finite number of times until the test condition becomes False.
- ➔ Once the test condition is false then PVM comes out of the while loop and executes a block of statements which are written within the else block and later executes other statements in the program.
- ➔ Writing else blocks of statements is optional. Else block will not execute when PVM control comes out of the while loop due to break statement.

12) Program Input:

Write a python program which will generate ONE to N numbers. Where n must be a positive integer value.

```
#number generation
n=int(input("Enter a number:"))
if(n<=0):
    print("{} is invalid input".format(n))
else:
    print("-"*40)
    print("Numbers with in: {}".format(n))
    print("-"*40)
    i=1  #initialisation part
    while(i<=n):  #condition
        print("\t{}".format(i))
        i=i+1 #updation(increament)
    print("-"*40)
```

Output:

```
Enter a number:5
----------------------------------------
Numbers with in: 5
----------------------------------------
        1
        2
        3
        4
        5
----------------------------------------
...
```

13) Program Input:

Write a python program which will accept +ve number or -ve number and print 1,2,....n. And also print -1,-2,-3,....-n.

```
#numgenration2.py
n=int(input("Enter a number:"))
if(n==0):
    print("{} is invalid input:".format(n))
else:
    print("-"*40)
    print("Numbers within:{}".format(n))
    print("-"*40)
    i=1 #initialisation part
    while(i<n): #condition
        print("\t{}".format(i))
        i=i+1 #updation (incre)
    else:
        print("-"*40)
        j=-1
        while(j>=n):
            print("\t{}".format(j))
            j=j-1
        else:
                print("-"*40)
```

Output:

```
Enter a number:9
----------------------------------------
Numbers within:9
----------------------------------------
        1
        2
        3
        4
        5
        6
        7
        8
----------------------------------------
----------------------------------------

Enter a number:-5
----------------------------------------
Numbers within:-5
----------------------------------------
----------------------------------------
        -1
        -2
        -3
        -4
        -5
----------------------------------------
```

13) Program Input:

Write a python program which will generate a multiplication table for a given number.

```
#Multiplication table.py
n=int(input("\tEnter a number:"))
if(n<=0):
    print("{} is invalid number:".format(n))
else:
    print("-"*40)
    print("\tMultiplication Table for:{}".format(n))
    print("-"*40)
    i=1
    while(i<=10):
        print("\t{} * {} = {}".format(n,i,n*i))
        i=i+1
    else:
        print("-"*40)
```

Output:

```
        Enter a number:5
----------------------------------------
        Multiplication Table for:5
----------------------------------------
        5 * 1 = 5
        5 * 2 = 10
        5 * 3 = 15
        5 * 4 = 20
        5 * 5 = 25
        5 * 6 = 30
        5 * 7 = 35
        5 * 8 = 40
        5 * 9 = 45
        5 * 10 = 50
----------------------------------------
```

14) Program Input:

Write a python program which will find the sum of elements of a list.

```
#lstsum.py
lst=[10,20,30,12.34,-12,45]
s=0
i=0
print("-"*50)
print("Elements of list:")
print("-"*50)
while(i<len(lst)):
    print("\t{}".format(lst[i]))
    s=s+lst[i]
    i=i+1
else:
    print("-"*50)
    print("\tsum={}".format(s))
    print("-"*50)
```

Output:

```
-------------------------------------------------
Elements of list:
-------------------------------------------------
        10
        20
        30
        12.34
        -12
        45
-------------------------------------------------
        sum=105.34
-------------------------------------------------
```

2) For Loop (or) For--Else:

Syntax:

```
For varname in iterable_obj:
        Statement-1
        Statement-2
        ----------
        ----------
        Statement-n
```

Explanation:

- ➔ 'For' and 'in' are keywords.
- ➔ Varname is one valid variable name.
- ➔ Iterable_object represents any object containing more number of values such as sequential type (str, bytes, bytearray, range), list type (list, tuple), set type (set, frozenset), dict type (dict).
- ➔ The execution behaviour of for loop is that every value of iterable_object is placed in varname one by one and executes block of statements. (Statement-1, Statement-2,----------,Statement-n).
- ➔ In general if an iterable_obect contains 'n' values then n-times block of statements will execute by placing value by value in varname n-times.
- ➔ **Example1:**

```
#forex1.py
s="PYTHON PROG"
for x in s:
    print("I am in for loop: ", x)
print("---------------------------------")
i=0
while (i<len(s)):
    print("in while loop:",s[i])
    i=i+1
```

➔ **Example2:**

```
#forex2.py
tpl=(10,"KVR","OUCET",12.34,"NIT",True)
print("Elements in List:")
print("-"*40)
for val in tpl:
    print(val)
else:
    print("-"*40)
```

➔ **Example3:**

```
#forex3.py
s={12,34,34.56,"NIT","Hyd","Python"}
print("elements in set")
print("-"*40)
for x in s:
    print(x,end=" ")
else:
    print()
    print("----------------------------------")
```

➔ **Example4:**

```
#forex4.py
d={1:"Hyd",2:"AP",3:"MUMBAI",4:"BANGLORE",5:"PUNE"}
print("\tKey\tValue")
print("-"*45)
for k,v in d.items():
    print("\t{}\t{}".format(k,v))
else:
    print("-"*45)
```

15) Program Input:

Write a python program which will accept list of elements dynamically in list and find sum and average of elements of list

```
#Print sum & average of given list elements
list=[]
n=int(input("Enter How many elements you want in the list:"))
print("-"*20)
for i in range(0,n):
    m=float(input())
    list.append(m)
    print("List Elements are: {}".format(list))
print("-"*20)
s=0
i=0
while(i<len(list)):
    print("\t{}".format(list[i]))
    s=s+list[i]
    i=i+1
else:
    print("-"*20)
    print("Sum={}".format(s))
    print("-"*20)
    avg=s/n
    print("Average of Given Elements:{}".format(avg))
    print("-"*20)
```

(OR)

```
#listop.py
n=int(input("Enter how many number of elements:"))
if(n<=0):
   print("{} is invalid input:".format(n))
else:
   lst=list()  # create empty
   print("Enter {} values:".format(n))
   for v in range(1,n+1):
      val=float(input())
      lst.append(val)
   #logic for sum and average
   s=0
   print("-" * 40)
   print("Elements of List:")
   print("-" * 40)
   for val in lst:
      print("\t{}".format(val))
      s=s+val

   print("-"*40)
   print("sum={}".format(s))
   print("Average={}".format(s/len(lst)))
   print("-" * 40)
```

Output:

```
Enter How many elements you want in the list:5
--------------------
2
List Elements are: [2.0]
1
List Elements are: [2.0, 1.0]
63
List Elements are: [2.0, 1.0, 63.0]
5
List Elements are: [2.0, 1.0, 63.0, 5.0]
45
List Elements are: [2.0, 1.0, 63.0, 5.0, 45.0]
--------------------
        2.0
        1.0
        63.0
        5.0
        45.0
--------------------
Sum=116.0
--------------------
Average of Given Elements:23.2
--------------------
```

16) Program Input:

Write a python program which will accept dynamic no of values in a list and sort these values in ascending and descending orders.

```
#Ascending and Descending orders
n=int(input("Enter how many values you entered:"))
if(n<=0):
    print("You are entered Invalid input of {}".format(n))
else:
    list=[]
    for i in range(0,n):
        m=float(input())
        list.append(m)
    else:
        print("Given list elements:\n{}".format(list))
        print("-"*40)
        list.sort()
        print("list elements in Ascending order:\n{}".format(list))
        print("-"*40)
        list.reverse()
        print("list elements in descending order:\n{}".format(list))
```

Output:

```
Enter how many values you entered:3
5
-4
0
Given list elements:
[5.0, -4.0, 0.0]
----------------------------------------
list elements in Ascending order:
[-4.0, 0.0, 5.0]
----------------------------------------
list elements in descending order:
[5.0, 0.0, -4.0]
```

Output2:

```
Enter how many values you entered:-5
You are entered Invalid input of -5
```

17) Program Input:

Write a python program which will accept dynamic no of values and find no. of occurrences of each element.

3) Miscellaneous control structures:

➔ As a part of Miscellaneous control structures, we have 3 types of statements. They are,

1) Break
2) Continue
3) Pass

1) Break:

➔ The break statement is used for logical termination of repetition of loop and comes out of loop and executes other statements in the python programs.

Syntax1: While(test condition):

……
if(another test condition):
break
……
…….
….Other statements in python

Syntax2: for varname in iterable_object:

……..
if(another test condition):
Break
……..
……..
...Other statements in python

Example1:

```
#bex1.py
s="PYTHON"
for x in s:
   if (x=='H'):
      break
   print(x)
else:
   print("i am out of for loop")
print("other statements in the program")
```

Example2:

```
#bex2.py
lst=[10,"shyam",34.56,True,"Hyd",2+3j]
i=0
while(i<len(lst)):
   if (lst[i]==True):
      break
   print(lst[i])
   i=i+1
else:
   print("i am from else...while")
print("other statements in the program")
```

18) Program Input:

Write a python program which will accept dynamic no. of values in list and print the list elements. Accept an element from the keyboard and search in a list. If the element is found then display its position. Otherwise the Display element does not exist.

```
#searchprog.py
n=int(input("Enter how many elements u have in a list:"))
if(n<=0):
    print("{} invalid input:".format(n))
else:
    lst=list()
    for i in range(1,n+1):
        print("Enter {} value:".format(i))
        val=int(input())
        lst.append(val)
    else:
        print("Elements of list:")
        print("-"*20)
        for val in lst:
            print("{}\n".format(val),end="")
        print("-"*20)
    print("Enter which element you want to search:")
    selement=int(input())
    res=False
    for i in range(0,len(lst)):
        if(lst[i]==selement):
            res=True
            break
    if(res):
        print("Element is found succesfully at Indexing:",i)
    else:
        print("Element does not Exists")
```

Output:

```
Enter how many elements u have in a list:4
Enter 1 value:
6
Enter 2 value:
67
Enter 3 value:
98
Enter 4 value:
45
Elements of list:
--------------------
6
67
98
45
--------------------
Enter which element you want to search:
65
Element does not Exists
```

2) Continue:

➔ Continue statement is used for repeating the loop when the condition is satisfied without executing those statements which are written after the continue statement.

Syntax: **while(test condition):**

……

if(another test condition):

continue

statement-1

statement-2

……..

statement-n

….Other statements in python

Syntax2: **for varname in iterable_object:**

……..

if(another test condition):

continue

statement-1

statement-2

……..

statement-n

...Other statements in python

Example1:

```
#contex1.py
# program printing those characters except H and Y letters
s="PYTHON"
for x in s:
   if (x=='H') or (x=='Y'):
      continue
   print("\t",x)
else:
   print("i am in else... for loop")
```

Example2:

```
#contex2.py
#program displaying +ve number and negative numbers separately.
lst=[10,-20,30,40,-50,-60,70,0]
for val in lst:
    if(val<=0):
        continue
    print("\t",val)
print("================================")
for val in lst:
    if(val>=0):
        continue
    print("\t",val)
```

Nested / inner loops:

➔ The purpose of writing one loop in another loop is called nested or inner loop.

Syntax1: **While loop in while loop**

```
while (test cond1):                 #outer while loop
        ……..
        ……..
        while (test cond2):         #inner while loop
                ……..
                ……..
        else:
                ……..
else:
        ……..
        ……..
```

Syntax2: **For loop in for loop:**

```
for var name1 in iterable_object:                   #outer for loop
        ……
        ……....
        for var name2 in iterable_object:           #inner for loop
                ……
                ……
        else:
                ……
else:
        ……
```

Syntax3: **For loop in while loop:**

```
while (test cond1):                          #outer while loop
        ……..
        ……..
        for var name1 in iterable_object:    #inner for loop
                ……
                ……
        else:
                ……
else:
        ……
```

Syntax4: **While loop in for loop:**

```
for var name1 in iterable_object:            #outer for loop
        ……
        ……....
        while (test cond1):                  #inner while loop
                ……..
                ……..
        else:
                ……..
else:
        ……..
        ……..
```

19) Program Input:

Write a python program which will generate 1 to n multiplication tables. Where n must be the +ve integer value.

```
#Multiplication tables using inner loop
nom=int(input("Enter how many multiplication tables u want:"))
if(nom<=0):
    print("{} invalid input".format(nom))
    print("-"*20)
else:
    for n in range(1,nom+1):
        print("Mul table for :{}".format(n))
        print("-"*20)
        for i in range(1,11):
            print("\t{} * {} = {}".format(n,i,n*i))
        else:
            print("-"*20)
    else:
        print("All mul tables are printed above")
```

Output:

```
Enter how many multiplication tables u want:2
Mul table for :1
--------------------
        1 * 1 = 1
        1 * 2 = 2
        1 * 3 = 3
        1 * 4 = 4
        1 * 5 = 5
        1 * 6 = 6
        1 * 7 = 7
        1 * 8 = 8
        1 * 9 = 9
        1 * 10 = 10
--------------------
Mul table for :2
--------------------
        2 * 1 = 2
        2 * 2 = 4
        2 * 3 = 6
        2 * 4 = 8
        2 * 5 = 10
        2 * 6 = 12
        2 * 7 = 14
        2 * 8 = 16
        2 * 9 = 18
        2 * 10 = 20
--------------------
All mul tables are printed above
```

20) Program Input:

Write a python program which will display multiplication tables for various selected values.

```
#Multiplication tables using inner loop
nom=int(input("Enter how many multiplication tables u want:"))
if(nom<=0):
    print("{} invalid input".format(nom))
    print("-"*30)
else:
    lst=list()
    for n in range(1,nom+1):
        print("Enter a number for which mul table u want:")
        v=int(input())
        lst.append(v)
    else:
        print("-"*30)
        print("mul table u want:",lst)
    for n in lst:
        print("-"*30)
        print("\tMul table for :{}".format(n))
        print("-"*30)
        for i in range(1,11):
            print("\t{} * {} = {}".format(n,i,n*i))
        else:
            print("-"*30)
    else:
        print("All mul tables are printed above")
```

Output:

```
Enter how many multiplication tables u want:2
Enter a number for which mul table u want:
3
Enter a number for which mul table u want:
25
------------------------------
mul table u want: [3, 25]
------------------------------
        Mul table for :3
------------------------------
        3 * 1 = 3
        3 * 2 = 6
        3 * 3 = 9
        3 * 4 = 12
        3 * 5 = 15
        3 * 6 = 18
        3 * 7 = 21
        3 * 8 = 24
        3 * 9 = 27
        3 * 10 = 30
------------------------------
------------------------------
        Mul table for :25
------------------------------
        25 * 1 = 25
        25 * 2 = 50
        25 * 3 = 75
        25 * 4 = 100
        25 * 5 = 125
        25 * 6 = 150
        25 * 7 = 175
        25 * 8 = 200
        25 * 9 = 225
        25 * 10 = 250
------------------------------
All mul tables are printed above
```

Write a python program which will display the multiplication table for the selected +ve numbers by rejecting -ve and 0 numbers.

21) Program Input:

Write a python program print Right angle triangle pattern

*

**

***

****

***** like this.

```
#print format of right angle triagle
n=int(input("Enter how many star lines you want:"))
if(n<=0):
    print("{} invalid input".format(n))
else:
    for i in range(0,n):
        for j in range(0,i+1):
            print("*",end="")
        print()
```

Output:

```
Enter how many star lines you want:4
*
**
***
****
```

Development of Functions:

Index:

- ➔ Importance of functions
- ➔ Definition, Advantages and syntax for defining functions
- ➔ Approaches to define functions
- ➔ Parameters and arguments
- ➔ Types of parameters / Arguments
    1) Positional parameters
    2) Default parameters
    3) Keyword parameters
    4) Variable length parameters
    5) Keyword variable length parameters
- ➔ Local variable and global variable
- ➔ Global and globals()
- ➔ Anonymous functions / lambda functions
- ➔ Special functions in python
    1) filter()
    2) map()
    3) reduce()

Development of Functions:

Importance of functions:

- ➔ In any programming, to perform any type of operation, we use the concept of functions.In python, without a function concept, we can't do a single operation.
- ➔ Example: print(), id(), int(), float(), bin(), type()....etc

Types of functions in python:

- ➔ We have two types of functions.they are,
  1) Predefined / built in functions.
  2) Programmer / user defined functions.

1) Predefined / built in functions:

- ➔ These functions are already developed by python language developers and available in python software and they deal with universal requirements.
- ➔ Ex: list() tuple() hex() oct() sort() reverse()... etc

2) Programmer / user- defined functions:

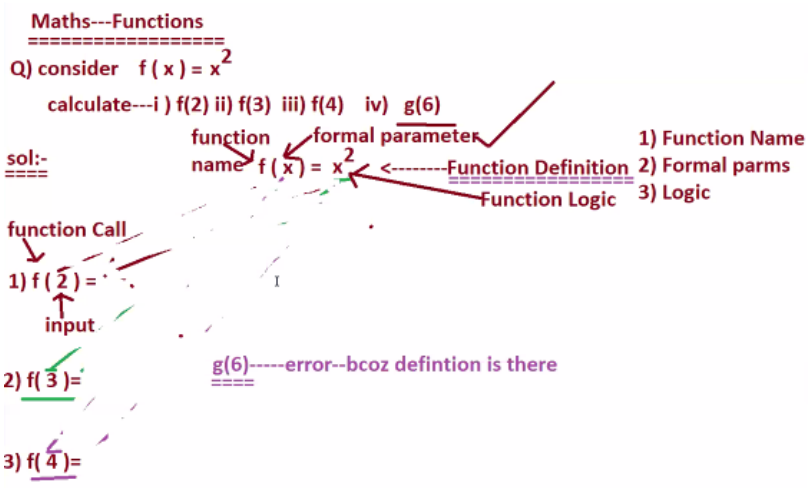
- ➔ These functions are already developed by python programmers and available in python projects and they deal with common requirements.

Definitions of Functions:

- ➔ A part of the main program is called a function.
- ➔ (or) Sub program of the main program is called a function.
- ➔ The aim of functions is to provide re-usability and to perform some operations.

Parts of functions:

- ➔ At the time of dealing with functions, we must ensure 2 parts. They are,
  1) Functional definition
  2) Function call
- ➔ Function definition exists only once.
- ➔ Function calls can exist many times.
- ➔ For every function call, there must exist a function definition otherwise we get errors.

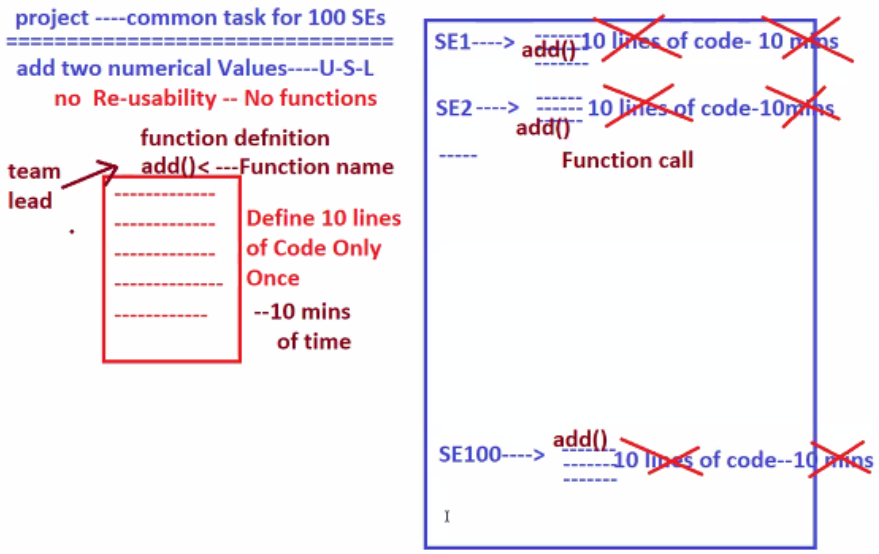


1) Un-structured Programmer Languages------No concept of Functions

Limitations without functions:
1. Application development time is More.
2. Application Memory Space is More.
3. Application Execution Time is More.
4. Application Performnace is Degraded
5. Redundency of the code More.

2) structured Programmer Languages---Functions available

1. Application development time is Less
2. Application memory Space is Less
3. Application execution is less
4. Application Performnce is enhanced
5. Redundency of code is Minimized
6. Write Once and Call Any where.



Syntax for defining a function in python:

(1) (2) (3)

Def functionname (list of formal params if any):

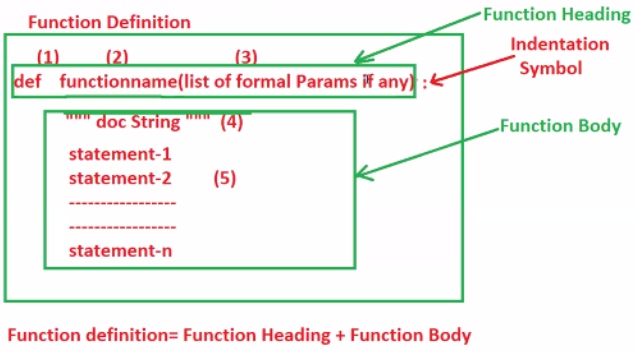
"""Doc string""" (4)

statement-1

statement-2 (5)

……..

statement-n



Explanation:

1) 'def' is a keyword used for defining a function.
2) 'function name' is one of the valid variable names treated as the name of the function.
3) 'list of formal parameters' represents variable names used in function heading and they are used for storing the input which is coming from the function calls.
4) 'doc string' stands for documentation string and this represents description of the functionality of code used in the function (nothing but comment) and it is optional.
5) Statement1, Statement2, Statement3,....Statement-n (called block of statements) represents a set of executable statements which is providing a solution to the given problem.
6) In some circumstances, we use some special variables inside the function body and those variables are local variables. The purpose of a local variable is to store temporary results.
7) The values of formal parameters and local variables can be used / accessed inside of corresponding function definitions only but not possible to access in other function definitions. (Known as scope of local and formal parameters)

Ex:

Define a function for calculating addition of two numerical values

Approach-1

```
#approach1.py

#Approach1---Logic followed
#-------------------------------------------------------------------------
#Taking Input:--- Function Call (outside)
#processing:---Function Body (inside)
#Giving Output:---Function call (outside--return)
#-------------------------------------------------------------------------


def  addition(a,b): # here 'a' and 'b' are called formal params
    c=a+b   # here 'c' is called local variable
    return c

#main program
x1=int(input("Enter First value:"))
x2=int(input("Enter Second value:"))
res1=addition(x1,x2) # function call
print("{} + {}={}".format(x1,x2,res1))
```

A1 : RV fun TV

A2: NRV fun NTV

A3: NRV fun TV

A4: RV fun NTV

Approach-2

```
#Approach2.py

#Approach2---Logic followed
#-------------------------------------------------------------------------
#Taking Input:--- Function Body (Inside)
#processing:---Function Body (inside)
#Giving Output:---Function Body (Inside)
#-------------------------------------------------------------------------

def  addition():
    #taking input---inside of Function Body
    a=int(input("Enter First value:"))
    b=int(input("Enter Second value:"))
    #processing inside of Function Body
    c=a+b
    #Give output--inside of Function Body
    print("{} + {}={}".format(a,b,c))

#main program
addition()
```

Approach-3

```
#Approach3.py

#Approach3---Logic followed
#-------------------------------------------------------------------------
#Taking Input:--- Function Call (Inside)
#processing:---Function Body (inside)
#Giving Output:---Function Body (Inside)
#-------------------------------------------------------------------------
def  addition(a,b):
    c=a+b
    print("sum of {} and {}={}".format(a,b,c))

#main program
x1=float(input("Enter First value:"))
x2=float(input("Enter Second value:"))
addition(x1,x2)
```

Approach-4

```
#Approach4---Logic followed
#-------------------------------------------------------------------------
#Taking Input:--- Function Body (Inside)
#processing:---Function Body (inside)
#Giving Output:---Function Call (outside)
#-------------------------------------------------------------------------
def   addition():
    a=float(input("Enter First value:"))
    b=float(input("Enter Second value:"))
    c=a+b
    return a,b,c

#main program
a,b,c=addition()
print("sum of {} and {}={}".format(a,b,c))
print("-----------------------------------------------------------")
x=addition() # valid, here variable 'x' is holding all values, which are comming from
             function definition and whose type is tuple
print("sum of {} and {}={}".format(x[0],x[1],x[2]))
```

Note: In python programming, return statement can return one or more no.of values.

Define the functions for displaying the values of

(Program pending)

Write a python program for calculating x^n

```
#base.py
def  powerton():
    x=int(input("Enter the base value:"))
    n=int(input("Enter the power value:"))
    result=x**n
    return x,n,result


#main program
x,n,res=powerton()
print("power({},{})={}".format(x,n,res))
```

Write a python program which can find max and min elements from the given list of elements

```
#maxmin.py
def findmaxmin(lst):
    maxelement=max(lst)
    minelement=min(lst)
    return maxelement,minelement

def dispresult(xl,he,se):
    print("-----------------------------------")
    print("Elements of List:")
    print("-----------------------------------")
    for v in xl:
        print("\t{}".format(v))
    print("-----------------------------------")
    print("Max Element={}".format(he))
    print("Min Element={}".format(se))

#main program
lst=[10,20,30,4,-5,0,56,23,234,-45]
maxe,mine=findmaxmin(lst)
dispresult(lst,maxe,mine)
```

Work

1) Write a python program which will accept a numerical +ve integer value and generate a multiplication table by using functions.
2) Write a python program which will accept a list of elements and sort the elements of both ascending and descending order using functions.
3) Write a python program which will accept numerical +ve integer values and find the square root of a given value using functions.

Parameters and arguments:

Parameter:

➔ Parameters or formal parameters are those, which are used in function heading and whose values can be accessed within the corresponding function definition but not accessed outside of the function.

Arguments:

➔ Arguments are the variables used in function calls, which also acts as global variables.

Local variables:

➔ Local variables are those, which are used inside of the function body for storing temporary results and whose values can be accessed within the corresponding function definition but not accessed outside of the function.

Types of parameters / arguments:

In the context of the function of python, The arguments / parameters are classified into 5 types. They are,

1) Positional parameters / arguments
2) Keyword parameters / arguments
3) Default parameters / arguments
4) Variable length parameters / arguments
5) Keyword variable length parameters / arguments

1) Positional parameters / arguments

➔ The concept of positional parameters is that,
i) The no.of arguments in function call must be similar in function definition (Number of arguments must equal no. of formal parameters)
ii) The order and meaning of arguments in function call must be maintained in function definition also for accuracy of results.

Syntax: (function definition)

```
Def functionname(param1, param2,...param-n)
…….
…….
```

Syntax: (Function call)

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

Here the mechanism of passing the data is that the values of arg1, arg2,...arg-n are placed in param1, param2,...param-n by position, order and meaning.

Write a python program which demonstrates the concept of

```
#posparamex1.py
def  disp(stno,stname,stmarks,stcname):    #function definition---stno,stname,stmarks and
                                             stcname  are called formal parameters
   print("====================================")
   print("Student Number={}".format(stno))
   print("Student Name={}".format(stname))
   print("Student Marks={}".format(stmarks))
   print("Student College Name={}".format(stcname))
   print("====================================")

#main program
sno=10
sname="KVR"
marks=45.77
cname="OUCET"
disp(sno,sname,marks,cname)  #functional call---sno,sname,marks and cname are called
                                                                    arguments
```

2) <u>Keyword parameters / arguments:</u>

- ➔ We know the function definition and we don't know in which order the formal parameters available and if we want to pass the arguments data from function call by maintaining position, order and meaning we must use the concept of keyword arguments
- ➔ Programatically, the concept of keyword arguments never follows order, position and meaning but it archives the accuracy of the result.
- ➔ Programmatically, keyword arguments must be written after positional parameters otherwise we get errors.

Syntax: (function definition):
Def functionname(param1,param2,....param-n):

Syntax: (function call):
functionname(param-n=val-n,param-1=val1.....param-2=val-2)

Here the formal param name(s) must used as keywords in the function call and hence this

Write a python program which demonstrates the concept of keyword arguments

```
#kwdargsex1.py

def  disp(stno,stname,stmarks,stcname):
   print("{}\t{}\t{}\t{}".format(stno,stname,stmarks,stcname))

#main program
print("================================================")
print("Number\tName\tMarks\tColl.Name")
print("================================================")
disp(stcname="OUCET",stno=10,stname="KVR",stmarks=66.66) #function call-1
disp(stmarks=34.56,stcname="JNTU(H)",stno=11,stname="sharma") #function call-2
disp(12,"Bilal",stcname="HCU",stmarks=33.33)  #function call-3
#disp(stcname="JNTU(K)",stmarks=23.33, 13,"Sampath") # error
disp(13,stmarks=23.44, stcname="SKU",stname="Sampath")#function call-3
print("================================================")
```

3) <u>Default parameters / arguments:</u>

- ➔ The default parameters are those, the formal parameters initialized with a common value in function heading.
- ➔ When the values are common for many function calls then it is always recommended to use the concept of default parameters mechanism.
- ➔ The advantage of this mechanism is that it reduces to and fro calls between function call and function definition and improves the performance of functional programming.
- ➔ When we are using both positional parameters and default parameters in the same function heading, it is mandatory to write first positional parameters (non-default) and later we write default parameter(s).

<u>Syntax</u>:(function definition)

```
Def functionname(param1, param2,.... param_n-1=val1, param_n-2=val2):
---------------------------
---------------------------
```

Here param1, param2,.... are positional parameters
and param_n-1 and param_n are called default parameters.

Write a Python program which demonstrates the concept of default parameters (If passable use positional and keyword arguments)

```
#defparmex1.py
def  dispstudinfo(sno,sname,smarks,crs1="PYTHON",crs2="DS"):
   print("\t{}\t{}\t\t{}\t{}\t{}".format(sno,sname,smarks,crs1,crs2))

|
#main program
print("=============================================================")
print("\tNumber\tStudent Name\t Marks\tCourse")
print("=============================================================")
dispstudinfo(10,"dprasad",56.78)
dispstudinfo(20,"naveen",66.78)
dispstudinfo(30,"yshwant",66.78)
dispstudinfo(40,"vishnu",76.88)
dispstudinfo(50,"KRISH",78.99,"JAVA")
dispstudinfo(60,"MALLESH",68.99)
dispstudinfo(70,"anil",78.99,crs2="JAVA")
dispstudinfo(80,"anil",78.99,crs2="R",crs1="ML")
dispstudinfo(sname="sumeet",sno=90,crs1="R",crs2="SPRING",smarks=45.67)
#dispstudinfo(sname="karthik",sno=75, 45.67) error
dispstudinfo(75,"Karthik",smarks=34.56);
print("=============================================================")
```

Output:

```
=============================================================
        Number  Student Name     Marks   Course
=============================================================
        10      dprasad          56.78   PYTHON  DS
        20      naveen           66.78   PYTHON  DS
        30      yshwant          66.78   PYTHON  DS
        40      vishnu           76.88   PYTHON  DS
        50      KRISH            78.99   JAVA    DS
        60      MALLESH          68.99   PYTHON  DS
        70      anil             78.99   PYTHON  JAVA
        80      anil             78.99   ML      R
        90      sumeet           45.67   R       SPRING
        75      Karthik          34.56   PYTHON  DS
=============================================================
```

4) Variable length parameters / arguments

- ➔ When we come across n-similar function calls with variable number of values then in normal python programming, we need to define n-function definitions. This approach leads more development time, takes more execution time and more code redundancy.
- ➔ To overcome this type problem, we define “one function definition” for n-similar function calls with variable number of values. This approach gives less development time, less execution time and redundancy of the code is reduced.
- ➔ To hold variable number of values, in the function definition, we use a formal parameter preceded with asterisk(*) and that formal parameter is called variable length parameter and whose type is tuple.

Syntax: (function definition)

def functionname(list of formal parameters, *params):

Program:

Write a python program which demonstrates the concept of variable length parameters. (or) Write a python program which will accept and display variable no.of values.

```
#varargssex1.py
def  show ( *n): #  here *n is called variable length parameter name whose type is tuple
        for val in n:
                print("{}".format(val), end=" ") # 10    20
        print("\n")


def  disp ( *Nav): #  here *n is called variable length parameter name whose type is tuple
        for val in Nav:
                print("{}".format(val), end=" ") # 10    20
        print("\n")

#main program
#n-similar function calls with Variable number of values
show(10)
show(10,20)
show(10,20,30)
show(10,20,30,40)
show("kvr","python","hyd")
show()
#n-similar function calls with Variable number of values
print("----------------------------------------------------------------")
disp(10,"java")
disp("java","python","DS")
disp(10,12.34,34.56,4)
disp(2+3j,True,False,None)
```

Output

```
10

10 20

10 20 30

10 20 30 40

kvr python hyd


----------------------------------------------------------------
10 java

java python DS

10 12.34 34.56 4

(2+3j) True False None
```

Input:

Write a python program which will find sum & average of variable no.of numerical values

```
def sumavg(sname,place,*n,crs="PYTHON"):
    print("----------------------")
    print("Name=",sname)
    print("Place=",place)
    print("Course=",crs)
    print("----------------------")
    s=0
    for val in n:
        print("\t{}".format(val))
        s=s+val
    else:
        print("Sum={}".format(s))
        print("Average={}".format(s/len(n)))

#main program
sumavg("Dileep","HYD",10,20)
print("----------------------")
sumavg("Naveen","AP",10,20,90,195)
sumavg("Swamy","AP",10,20,90)
print("----------------------")
```

Output:

```
----------------------
Name= Dileep
Place= HYD
Course= PYTHON
----------------------
        10
        20
Sum=30
Average=15.0
----------------------
----------------------
Name= Naveen
Place= AP
Course= PYTHON
----------------------
        10
        20
        90
        195
Sum=315
Average=78.75
----------------------
Name= Swamy
Place= AP
Course= PYTHON
----------------------
        10
        20
        90
Sum=120
Average=40.0
----------------------
```

5) Keyword variable length parameters / arguments:

- ➔ When we come across n-similar function calls with Keyword variable length of values then in normal python programming, we need to define n-function definitions. This approach leads more development time, takes more execution time and more code redundancy.
- ➔ To overcome this type problem, we define "one function definition only" for n-similar function calls with Keyword variable length of values. This approach gives less development time, less execution time and redundancy of the code is reduced.
- ➔ To hold Keyword variable number of values, in the function definition, we use a formal parameter preceded with double asterisk(**) and that formal parameter is called Keyword variable length parameter and whose type is Dict.

Syntax: (function definition)

def functionname(list of formal parameters, **params):

Input:

Write a python program which will display keyword variable no of values

```
#keyword var length example
def show(**n): #here n is called kwd var length parameter and whose type is dict
    print("------------------")
    print("Key\tValue")
    print("------------------")
    for k,v in n.items():
        print("{}\t{}".format(k,v))
    print("------------------")

#mainprogram
show(sno=10,sname="Arjun")
show(sname="Allu", bobby="Research",place="Ap")
show(fno=1,fname="VNK",sub1="PYTHON",sub2="Java",sub3="Django")
```

Output:

```
------------------
Key     Value
------------------
sno     10
sname   Arjun
------------------
------------------
Key     Value
------------------
sname   Allu
bobby   Research
place   Ap
------------------
------------------
Key     Value
------------------
fno     1
fname   VNK
sub1    PYTHON
sub2    Java
sub3    Django
------------------
```

Input:

Write a python program which will find total marks of various students who are securing different subjects, who are studying in different classes.

```
def findtotalmarks(sname,scls,**marks):
    tot=0
    print("-----------------------")
    print("Student Name = {}".format(sname))
    print("Student class = {}".format(scls))
    print("-----------------------")
    for k,v in marks.items():
        print("{}\t{}".format(k,v))
        tot=tot+v
    print("-----------------------")
    print("Total Marks = {}".format(tot))
    print("-----------------------")

#main program
findtotalmarks("Harshitha","X",Eng=50,Hindi=60,Sco=70,Sci=80,Maths=55)
findtotalmarks("Raju","XII",Bio=55,Zolo=60,Che=58)
findtotalmarks("Naveen","B-Tech",C=70,CPP=74,Python=85,Java=78)
findtotalmarks("Rossum","M-Tech")
```

Output:

```
-----------------------
Student Name = Harshitha
Student class = X
-----------------------
Eng     50
Hindi   60
Sco     70
Sci     80
Maths   55
-----------------------
Total Marks = 315
-----------------------
-----------------------
Student Name = Raju
Student class = XII
-----------------------
Bio     55
Zolo    60
Che     58
-----------------------
Total Marks = 173
-----------------------
-----------------------
Student Name = Naveen
Student class = B-Tech
-----------------------
C       70
CPP     74
Python  85
Java    78
-----------------------
Total Marks = 307
-----------------------
-----------------------
Student Name = Rossum
Student class = M-Tech
-----------------------
-----------------------
Total Marks = 0
-----------------------
```

Local Variable and Global Variables:

Local Variable:

Local variables are those, which are defined in the corresponding function body for storing temporary results. The scope of local variables is available only to the corresponding function definition. In other words local variable values can be accessed only within the corresponding function definition but not possible to access in other functions.

Input

Write a python program which makes understand scope of global variables

```
#globalvarex1.py
crs="PYTHON"   #global variable can be accessed in  all the function definitions
def   learnds():
   print("To learn DATA SCIENCE we need programming lang {}".format(crs))
def   learniot():
   print("To learn IOT we need programming lang {}".format(crs))
def learnml():
   print("To learn ML we need programming lang {}".format(crs))


#main program
learnds()
learniot()
learnml()
```

Global Variable:

Global variables are those, which are defined before all the functions definitions. The purpose of global variables is to place common values, which are accessed by all the function definitions.

➔ The scope of global variables is more than scope of local variables.

Input

Write a python program which makes understand scope of local variables

```
#localvarex1.py
def    faculty1():
   fname1="KVR" # local variable
   print("Facult Name--Faculty1()={}".format(fname1))
   #print("Facult Name--Faculty2()={}".format(fname2)) error-> fname2 can't access bcoz it
                                                              is local to faculty2()

def    faculty2():
   fname2="Sampath" # local variable
   print("Facult Name--Faculty2()={}".format(fname2))
   #print("Facult Name--Faculty1()={}".format(fname1)) error-> fname1 can't access bcoz it
                                                              is local to faculty1()
#main program
faculty1()
faculty2()
```

Input

Write a python program which makes understand scope of Local and Global variables

```
#globallocalex1.py
cap="INDIA" #global variable

def  live1():
   myplace="TS" #local variable
   print("{} is part of {}".format(myplace,cap))

def  live2():
   myplace="AP" #local variable
   print("{} is part of {}".format(myplace,cap))
def  live3():
   myplace="MH" #local variable
   print("{} is part of {}".format(myplace,cap))
def  live4():
   myplace="KARNATAKA" #local variable
   print("{} is part of {}".format(myplace,cap))

#main program
live1()
live2()
live3()
live4()
```

Global Keyword and globals():

Global Keyword:

➔ When we want to modify the value of a global variable within the function definition then first we must refer / access global variable by using a global keyword and modification must take place. (Without a global keyword, global variable values can't be modified within the function definition and we get errors).

Syntax for referring / accessing global variables by using global keyword:

varname1=val1 #here varname is called global variable
def functionname():

    ----
    global name

Input:

Write a python program which will modify the global variable in various function definitions

```
#modifygv.py
a=10 # global varibale
def  modifyval():
   global a # refering global variable value
   a=a+1    # modifying global variable value
   print("val of a in modifyval()={}".format(a)) #11
def updateval():
   global a  # refering global variable value
   a=a*2     # modifying global variable value
   print("val of a in updateval()={}".format(a)) #22

#main program
print("val of a before modification by the functions={}".format(a))  # 10
modifyval()
print("val of a after modiyval()={}".format(a)) # 11
updateval()
print("val of a after updateval()={}".format(a)) # 22
```

```
-------------------------------------------------------------------------------------------------
Syntax for referering / accessing global variables by using global kwd:
-------------------------------------------------------------------------------------------------

varname=val1  # here varname is called global variable

def   functionname():
      ---------------------------
      ---------------------------
     global  varname     # refers global variable name before modification.
     varname=varname+2----------valid
      ---------------------------
      ---------------------------
=========================================================================
```

Globals():

- ➔ Whenever the global variable names and local variables are same then to do the operations both global and local variables directly, Python virtual machine gets ambiguity (there is no clarity between multiple duplicate var names).
- ➔ To differentiate between global variable names and local variables in corresponding functions, the global variables must be retrieved / extracted by using globals().
- ➔ **globals() returns all global variables in the form dict.**

Syntax:

```
globalvar1=var1
--------------------
globalvar-n=val-n
Def functionname():
        ---------------
        --------------
```

varname1=globals()[‘globalvar1’]
varname2=globals()[‘globalvar2’]
---
varname-n=globals()[‘globalvar-n’]

Input:

Write a py program which perform the operations on local and global variables (add the values of global and local variables)

Ex-i)

```
#globalsex1.py
a=10
b=20
c=30
d=40   # here 'a','b','c' and 'd' are called global Variables.
def  operations():
    global c,d
    c=c+1     # val of c= 31
    d=d+1    #  val of d=41
    a=100
    b=200
    ga=globals()['a']  # ga=10
    gb=globals()['b'] # gb=20
    print("val of a(local)={}".format(a))
    print("val of b(local)={}".format(b))
    print("val of a(gobal)={}".format(ga))
    print("val of b(gobal)={}".format(gb))
    result=c+d+a+b+ga + gb      # 31+41+100+200+10+20
    print("result={}".format(result)) # 402

#main program
operations()
```

Ex-ii)

```
#globalsex2.py
a=10
b=20
c=30
d=40  # here 'a','b','c' and 'd' are called global variables
def operations():
    a=1
    b=2
    c=3
    d=4  # here 'a','b','c' and 'd' are called local variables
    ga=globals()['a']
    gb=globals()['b']
    gc1=globals()['c']
    gd=globals()['d']
    localresult=a+b+c+d+ga+gb+gc1+gd
    print("result=",localresult)

#main program
operations()
```

Anonymous (or) Lambda functions in python:

- ➔ Anonymous functions do not contain names explicitly.
- ➔ The purpose of Anonymous functions is to perform "instant operations". Instant operations are those, which are performing at that point of time (no longer interested to use in further).
- ➔ To develop Anonymous function, we use a keyword / operator called "Lambda" and hence Anonymous functions are called "Lambda functions"
- ➔ Anonymous functions return the result automatically. We need not write a return statement explicitly.

- ➔ **Syntax:**

  varname=lambda args-list:expression

- ➔ **Explanation:**
  - ◆ Varname is the valid name of python, which is treated as Anonymous function name.
  - ◆ Lambda is a keyword / operator used for defining Anonymous functions.
  - ◆ args-list represents formal params
  - ◆ Expression represents a single executable statement

Input:

Define a Lambda function or Anonymous functions for finding sum, sub and mul operations on two numbers

```
#lambda ex1.py
#program for finding sum of two numbers

def sumop(a,b): #normal function def
    c=a+b
    return c

addop=lambda x,y:x+y #Anonymous function
subop=lambda a,b:a-b #Anonymous function
mulop=lambda v,n:v*n #Anonymous function

#main programm
result1=sumop(10,20)
print("Sum of 2 numbrs with normal function = {}".format(result1))
result2=addop(100,200)
print("Sum of 2 numbrs with Lambda function = {}".format(result2))
print("Sum of 2 numbrs with Lambda function = {}".format(addop(10,50)))
print("sub of 2 numbrs with Lambda function = {}".format(subop(10,20)))
print("mul of 2 numbrs with Lambda function = {}".format(mulop(4,16)))
```

Input:

Define a lambda function for finding biggest among two numbers

```
#Lambda ex 2

bigop=lambda a,b:a if (a>b) else b

#main program
x=int(input("Enter First number:"))
y=int(input("Enter second number:"))
print("big ({},{}) = {}".format(x,y,bigop(x,y)))
```

Special functions in Python:

➔ In python programming, we have 3 special functions, they are

1) filter()
2) map()
3) reduce()

1) filter():

➔ **Purpose:** The purpose of filter() is to filter out the elements by applying elements of iterable objects to the specified function.

➔ **Syntax of filter():**

varname=filter(functionname,iterable_obj)

➔ **Explanation:**

1) Varname is an object of <class, 'filter'>
2) filter() is a pre_defined function, which is used for filtering out the elements of any iterable object by applying it to the function. The execution behaviour of filter() is that if the function returns True then filter that element. If the function returns False then dont filter that element.
3) Function name can be either normal function or anonymous function.
4) Iterable objects can be either sequence, list, set, dict type.

Input:

Write a python program which will filter +ve elements and -ve elements separately (With normal functions).

```
#filter ex1.py
def positive(x):
    if(x>0):
        return True
    else:
        return False

def negative(a):
    if(a<0):
       return True
    else:
        return False

#main program
lst=[10,-20,3,-40,-50,23,45,60,-70,56]

pl=list(filter(positive,lst))
print("Original elements List={}".format(lst))
print("Positive elements List={}".format(pl))
nl=list(filter(negative,lst))
print("Negative elements List={}".format(nl))
```

Input:

Write a python program which will filter +ve elements and -ve elements separately (With Anonymous (or) Lambda functions).

```
#filter ex2.py

#main program
lst=[10,-20,3,-40,-50,23,45,60,-70,56]
pl=tuple(filter(lambda x:x>0,lst))
print("-"*20)
print("Original elements List={}".format(lst))
print("Positive elements List={}".format(pl))
nl=tuple(filter(lambda a:a<0,lst))
print("Negative elements List={}".format(nl))
```

Input:

Write a python program which will filter even numbers and odd numbers separately from the list of given lists of numbers.

```
#Even and odd.py
print("Enter list of elements dynamically:")
lst=[int(x) for x in input().split()]
#define filters for even and odd
el=list(filter(lambda x:x%2==0,lst))
ol=list(filter(lambda a:a%2!=0,lst))
print("-"*20)
print("Original elements={}".format(lst))
print("\nEven list elements={}".format(el))
print("\nOdd list elements ={}".format(ol))
```

2) Map():

- ➔ The purpose of map() is to generate / create a new iterable object from an existing iterable object by applying existing elements of the iterable object to the particular function.
- ➔ **Syntax for map():**

  varname=map(function name ,iterable obj)
- ➔ **Explanation:**
  1) Varname is an object of <class, 'map'>
  2) map() is a pre_defined function. The execution behaviour of map() is that it applies each element of an iterable object to a specified function and generates a new iterable object based on the logic we write in the function (normal function, anonymous function).
  3) Function name can be either normal function or anonymous function.
  4) Iterable objects can be either sequence, list, set, dict type.

Input:

Write a python program Which will accept a list of integer values and square all the values of the list with normal function

```
#mapex1.py
def square(x):
    return(x**2)

print("Enter Integer values dynamically seperated with space:")
oldlst=[int(x) for x in input().split()]
sqlst=list(map(square,oldlst))
print("-"*20)
print("Old list={}".format(oldlst))
print("New list={}".format(sqlst))
```

Input:

Write a python program Which will accept a list of integer values and square all the values of the list with Anonymous (or) Lambda functions function.

```
#mapex2.py
print("Enter Integer values dynamically seperated with space:")
oldlst=[int(x) for x in input().split()]
sqlst=list(map(lambda x:x**2,oldlst))
print("-"*20)
print("Old list={}".format(oldlst))
print("New list={}".format(sqlst))
```

Input:

Write a python program Which will add the contents of two lists with map(). (The list may contain different sizes of elements).

```
#mapex3.py
lst1=[10,20,80,56]
lst2=[1,4,3]

sumlist=list(map(lambda x,y:x+y,lst1,lst2))
print("List1 elements={}".format(lst1))
print("List2 elements={}".format(lst2))
print("Sum of List1 & List2 elements={}".format(sumlist))
```

3) **Reduce():**

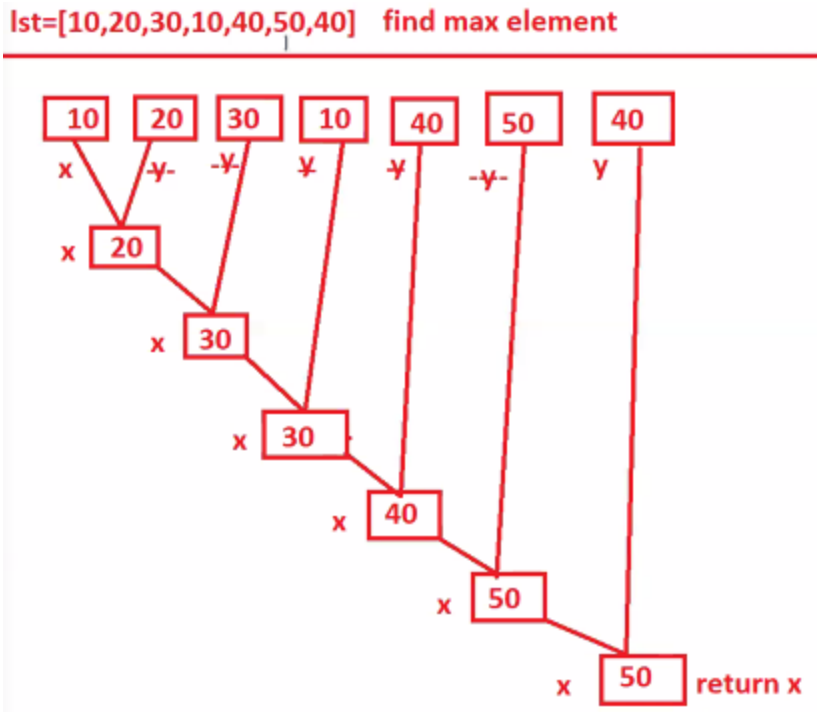
- ➔ The purpose of reduce() is to obtain a single element / result from the iterable object by applying it to the function.
- ➔ The reduce() present in a predefined module called “function tools”
- ➔ **Syntax:**

  varname=reduce(function name,iterable obj)
- ➔ **Explanation:**
  1) Varname can be either fundamental data types and strings.
  2) Function names can either be normal function or lambda function.
  3) Iterable objects can be either sequence, list, set, dict type.

- ➔ **Working functionality of reduce():**
  1) At first step, First two elements of iterable objects are picked and the result is obtained.
  2) Next step is to apply the same function in such a way that the previously obtained result and the succeeding element of the iterable object and the result obtained again.
  3) This process continues till no more elements are left in the iterable object.
  4) The final result must be returned.

Example:

Find the max element from the list of given numbers and draw the diagram.



Input:

Write a python program which will find highest and lowest elements from the given list of elements.Accet the list of elements dynamically.

```
#reduceex1.py
import functools
print("Enter list of values seperated by space:")
lst=[int(x) for x in input().split()]
big=functools.reduce(lambda x,y:x if (x>y) else y,lst)
small=functools.reduce(lambda x,y:x if (x<y) else y,lst)
print("-"*20)
print("Original Elements:{}".format(lst))
print("Big={}".format(big))
print("small={}".format(small))
print("-"*20)
lst.sort()
print("Sorted Elements={}".format(lst))
print("Second Biggest={}".format(lst[len(lst)-2]))
```

Input:

Write a python program which will find the sum of a list of numerical values.

```
#reduceex2.py
import functools
print("Enter list of values seperated by comma:")
lst=[int(x) for x in input().split(",")]
listsum=functools.reduce(lambda x,y:x+y,lst)

print("-"*20)
print("Original Elements:{}".format(lst))
print("-"*20)
print("sum={}".format(listsum))
```

Write a python program which will find the product of n natural numbers using reduce function. Where n must be the +ve integer value.

```
#reduceex3.py
from functools import reduce
n=int(input("Enter of n:"))
if(n<=0):
    print("{} is invalid".format(n))
else:
    resl=reduce(lambda a,b:a*b,range(1,n+1))
    print("-"*20)
    print("Product of first {} natural numbers".format(n))
    print("-"*20)
    for i in range(1,n+1):
        print("\t{}".format(i))
    else:
        print("-"*20)
        print("Product: {}".format(resl))
        print("-"*20)
```

1) Write a python program which will calculate the sum of first n natural numbers using reduce().
2) Write a python program which will find the sum of squares of first n natural numbers using reduce().
3) Write a python program which will find the sum of cubes of n natural numbers using reduce()..
4) Write a python program which will accept different lists of words and convert into a single sentence using reduce().

Input:

Write a python program which will accept ordinary numbers and convert them into equivalent roman numbers.

```
#roman.py
def normaltoroman(n):
    if(n<=0):
        print("{} invalid input".format(n))
    else:
        while(n>=1000):
            print("M",end="")
            n=n-1000
        if(n>=900):
            print("CM",end="")
            n=n-900
        if(n>=500):
            print("D",end="")
            n=n-500
        if(n>=400):
            print("CD",end="")
            n=n-400
        while(n>=100):
            print("C",end="")
            n=n-100
        if(n>=90):
            print("XC",end="")
            n=n-90
        if(n>=50):
            print("L",end="")
            n=n-50
        if(n>=40):
            print("XL",end="")
            n=n-40
        while(n>=10):
            print("X",end="")
            n=n-10
        if(n>=9):
            print("IX",end="")
            n=n-9
        if(n>=5):
            print("V",end="")
            n=n-5
        if(n>=4):
            print("IV",end="")
            n=n-4

        while(n>=1):
            print("I",end="")
            n=n-1
    print()

#main program
n=int(input("Enter a Number for Roman number :"))
normaltoroman(n)
print("-"*20)
```

Importance of Modules:

- ➔ We know that functions concept makes us understand how to re-use the variables (global variable) / data and functions in the same program but never allows us to re-use the variables (data) and functions across the program.
- ➔ In order to re-use global variables (data), function and classes within the program across the programs, we use the concept of MODULES.
- ➔ Hence functions provide local re-usability within the program and modules provide global re-usability across the programs.

Definition of Module:

- ➔ A module is a collection of variables(Data), Functions and classes

Types of Modules

- ➔ We have 2 types of modules. They are
  a) Pre-defined / build-in modules
  b) Programmer / user / custom defined modules.

a) Pre-defined / build-in modules:

- ➔ These modules are developed by language developers and they are available in languages and whose purpose is to deal with universal requirements.
- ➔ **Examples:** Calendar, random, re, pickle, math, cmath, sys, time, os, functions.. ...etc
- ➔ cs_Oracle, mysql.connector...etc

b) Programmer / user / custom defined modules:

- ➔ These modules are developed by language programmers and they are available in python projects and whose purpose is to deal with common requirements.

- ➔ **Creating a module in Python:**
- ➔ In Python, creating a module is nothing but write / define variables (Data), functions and classes in a python program and save it on some file name with extension .py (Filename.py)
- ➔ Here the filename itself acts as module name and it can be re-used across the programs.

Re-using the variables, functions and classes of the modules in other programs:

- ➔ We have two approaches to re-use the variables, functions and class of modules in other programs. They are
  1) By using import statement
  2) By using from …… import statement

1) **By using import statement :**
   - ➔ Import is a 'keyword'
   - ➔ Import statement is used for referring to the variables, functions and classes on modules in other programs.
   - ➔ When we refer to the variables, functions and classes with approach in other programs then the variable, functions and classes must be accessed with respect to module name otherwise we get Error.
   - ➔ **Syntax:**

     Modulename.variable Name
     Modulename.function Name
     Modulename.class Name

Syntaxes for importing a module:

Before accessing variables, functions and classes, First we must import module(s) by using the following syntaxes.

Syntax1:- import modulename

Syntax2:- import modulename1,modulename2,...modulename-n

Syntax3:- import modulename as alias name

Syntax4:- import modulename1 as alias name1,import modulename2 as alias name2,.....import modulename-n as alias name-n

a)
```
#greet.py --> Here file name acts as module name

def greetperson(sname):
    print("Hi:{} Good Evening".format(sname))
```

b)
```
#test1.py #filename-acts as module name

pi=3.14 #data
```

c)
```
#sel.py--> main program.. resulting the modules with import statement
import test1 as ts
import greet as g, calendar as c
print("Val of pi={}".format(ts.pi)) #we are using the var pi os some program in other program
g.greetperson("Ramu") #we are reusing the function greetperson() of some program in other program
g.greetperson("Naveen")
g.greetperson("Rithu")
print("-"*20)
print(c.month(2021,8))
```

2) **By using from …… import statement:**

- ➔ From Import are 'keywords'
- ➔ With this approach we can refer to the variables, functions and classes of modules in other programs directly without preceded by module name.

Syntax 1:- From module name import variablename1, …. Variable name-n, function name-1….function name-n

Syntax 2:- from module name import *

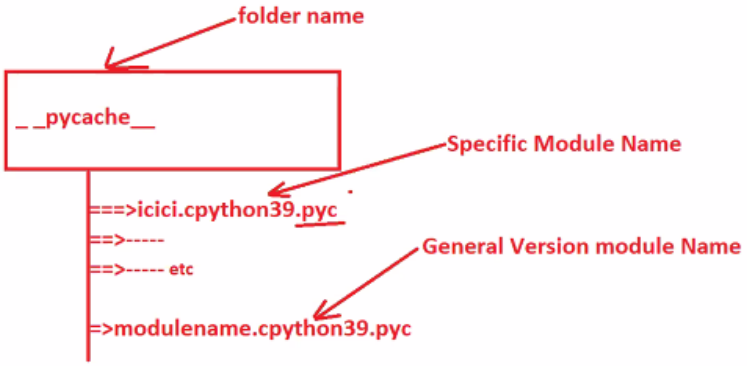
Syntax 3:- from module name import variable name1 as alias name1..variable nam-n as alias name-n, function name 1 as alias name-1….function name-n as alias name-n

Example Program:

```
#icici.py .... file name acts as module name
addr="H.No:1-72, VelamuriPadu, Addanki, Prakasam, Andra Pradesh" #data
def calintrest(p,t,r): #function
    si=(p*t*r)/100
    totamt=si+p
    print("-"*20)
    print("Intrest calculation:")
    print("-"*20)
    print("Principle Amount:{}".format(p))
    print("Time in Months:{}".format(t))
    print("Rate of interest:{}".format(r))
    print("-"*20)
    print("Simple intrest:{}".format(si))
    print("Total Amount to pay:{}".format(totamt))
    print("-"*20)
```

```
#prog1.py
from icici import addr as a
from icici import calintrest as ca
print("Address of ICICI :\n {}".format(a))
print("-"*20)
p=float(input("Enter Principle Amount:"))
t=float(input("Enter Time in months:"))
r=float(input("Enter Rate of Interest in rupees:"))
ca(p,t,r)
```

Note:- When we treat any python program as module name, internally the folder created automatically and structure is shown bellow.



Write a python program for calculating all arithmetic operations with modules in the form of menu driven approach.

1.
```
#menu.py
def menuop():
    print("-"*20)
    print("A R I T H M E T I C  O P E R A T I O N S")
    print("-"*20)
    print("\t1.ADDITION:")
    print("\t2.SUBSTRACTION:")
    print("\t3.MULTIPLICATION:")
    print("\t4.DIVISION:")
    print("\t5.MODULO DIVISION:")
    print("\t6.EXPONENTATION:")
    print("\t7.EXIT:")
    print("-"*20)
```

2.
```
#aop.py
def readvalues(op):
    print("Enter Two values for {}:".format(op))
    a=float(input())
    b=float(input())
    return a,b

def addition():
    a,b=readvalues("addition")
    print("Sum of {} and {} = {}".format(a,b,a+b))

def substraction():
    a,b=readvalues("substraction")
    print("Sub of {} and {} = {}".format(a,b,a-b))

def multiply():
    a,b=readvalues("multiplication")
    print("Sub of {} and {} = {}".format(a,b,a*b))

def division():
    a,b=readvalues("division")
    print("Sub of {} and {} = {}".format(a,b,a/b))

def modulas():
    a,b=readvalues("modulas")
    print("Sub of {} and {} = {}".format(a,b,a%b))

def expo():
    a,b=readvalues("Exponentation")
    print("Sub of {} and {} = {}".format(a,b,a**b))
```

```
#aopmain.py
from menu import menuop
from aop import*;
import sys
while(True):
    menuop()
    ch=int(input("Enter Ur Choice:"))
    if(ch==1):
        addition()
    elif(ch==2):
        substraction()
    elif(ch==3):
        multiply()
    elif(ch==4):
        division()
    elif(ch==5):
        modulas()
    elif(ch==6):
        expo()
    elif(ch==7):
        print("Thanks for using this application")
        sys.exit()
    else:
        print("Ur selection of operation is Wrong")
```

3.

<u>Re-Loading the modules</u>:

- ➔ The purpose of re-loading the module is to get new updated data from the previous imported module.
- ➔ During current execution of the program, externally, the imported module may be changed. In that circumstance, to get an update of the previous imported module, we must reload the existing module.
- ➔ To reload the module, we use a pre-defined function called reload(). This function present in imp module of python 2.x version and it is deprecated as importlib module in python 3.x version (recommended to use)

Syntax:

Import importlib

---
importlib.reload(modulename)

Write a python program to reload the module

1.
```
#shares.py.....treated as module
def shareinfo():
    sh={"IT":230,"pharmacy":450,"product":150}
    print("-"*20)
    print("Share name\tShare Value")
    print("-"*20)
    for sn,sv in sh.items():
        prnt("\t{}\t\t{}".format(sn,sv))
    print("-"*20)
```

2.
```
#sharesdemo.py
import shares
import time
import importlib

shares.sharesinfo()
print("Line-5..> Hello viewers i am going to sleep for 10 sec")
time.sleep(10)
print("Line-7..> Hello viewers i am out from the sleep for 10 sec")
importlib.reload(shares)
shares.sharesinfo()
```

Packages in python:

```
=====================================================
           Handling The Exceptions
=====================================================
=>Handling The Exceptions are nothing but converting Technical error
Messages into User-friendly error messages.
=>To Convert Technical error Messages into User-friendly error messages,
In python programming, we have 5 key words. They are
      1) try
      2) except
      3) else
      4) finally
      5) raise
-------------------------------------------------------
syntax for handling the exceptions:
-------------------------------------------------------
      try:
           block of statements causes
           exceptions in the program
      except  <exception class name-1>:
            block of statements provides
            User-Friendly Error Messages
      except  <exception class name-2>:
            block of statements provides
            User-Friendly Error Messages
      -------------------------------------------------
      -------------------------------------------------
      except  <exception class name-n>:
            block of statements provides
            User-Friendly Error Messages
      else:
             block of statements--recommended
             provide display results
      finally:
             block of statements which will
             execute compulsorily
=================================================
1) try:
---------------
=> It is the block , In which we write  block of statements causes
problems at runtime
     In otherwords, what are all the statements generating exceptions then
those statements must be written within try block. Hence try block is
called "exception Monitoring block"
=> When the exception occurs in try block then PVM comes out of try block
and executes        appropriate except block(if found otherwise PVM goes
out of program flow)
```

=> After executing appropriate except block PVM never comes to try block to execute rest of the statements in try block.
=>Each and Every Try block must have alteast one except block(otherwise we can't give User friendly Error Messages)
=>Each and Every Try block must be immediately followed by except block.
==========
2) except block:

=> It is the block , In which we write block of statements provides user-friendly error messages. In otherwords except block supresses the technical error messages and generates user-friendly error messages. Hence this block is called "exception Processingh block".

Note:- Handling the exception = try Block + except block

=> except block will execute provided an exception occurs in try block
=>Even though we write multiple except blocks, PVM can execute appropriate except block depends on on type of exception occurs in try block.
=>Industry is highly recommended to write multiple except blocks for generating multiple User-friendly errors messages.
=>except block to be written after try block and before else block (if we write else block)
==========
3) else block:

=> It is the block , In which we are recommended to write block of statements for generating results and hence this block is called Result block
=> else block will execute provided there is no exception occurs in try block.
=> Writing else block is optional
=>The place of writting else block is after except block and before finally block(if we write finally block)
==========
4) finally block:

==========
Exception Handling
==========
=>The aim of Exception Handling is that to develop Robust (Strong) Applications.
=> In Real Time , To develop any real time project, we need to choose a lang / tech.

=>With the language, we can develop programs, comiple and execute those Programs, During this process, we get 3 types of errors. They are

a) Compile Time Errors
b) Logical Errors
c) Runtime Errors

==========

a) Compile Time Errors:

=>These errors occurs during Compilation(.py------>.pyc) time
=>These errors occurs due to syntaxes are not followed.
=>These errors solved by Programmers at Development Level

b) Logical Errors:

=>These errors occurs during execution time
=>These errors occurs due to wrong represenation / mis-interpretation of logic and we get wrong Results
=>These errors solved by Programmers at Development Level

c) Runtime Errors

=>These errors occurs during execution time
=>These errors occurs due to wrong Input entered by Application User / end user
=>Runtime errors solved by Programmers at Implementation Level.

==========

Points to be Remembered

=>When The application user enters invalid / wrong input then we get Runtime Errors
(Invalid Input---------->Runtime Errors)
=>All Runtime errors are called Exceptions
(Invalid Input----->Runtime Errors---->Exceptions)
Hence all invalid inputs gives exceptions.
=>All exceptions gives by default "technical Error Messages "

=>Exception Handling:

The Process of Converting Technical Error Messages into User-Friendly Error Messsages is called Exception handling

=> When the exception occurs in the python program, Internally 3 stpes takes place

a) PVM terminates the program execution abnormally
b) PVM comes out of Program flow
c) By default, PVM generates Technical Error Messages

=>To do (a),(b) and (c) steps, " PVM creates an object of one exception class "
"Technically, all exceptions of python are treated as objects"

=>When the exception occurs in python program then PVM creates an object of appropriate exception class and by default generates technical error messages.

Types of exceptions in Python

=>In python Programming, we have two types of exceptions. They are

1. Pre-defined / Built-in exceptions
2. Programmer / user / Custom defined exceptions

1. Pre-defined / Built-in exceptions:

=>These exceptions developed by Python Language developpers and available as a part of languages API and they always deals with Universal Problems.

=>Some of the Universal Problems are
a) Division by zero problems (ZeroDivisionError)
b) Invalid number formats (ValueError)
c) Un-intended blocks (IndentationError)
d) wrong numbers of params / args (TypeError).....etc

2. Programmer / user / Custom defined exceptions:

=>These exceptions developed by PythonProgrammers and available as a part of python project and they always deals with Common Problems.
=> Some of the Common Problems are:
a) Attempting to enter Invalid Pin in ATM applications (PINError)
b) Attempring with draw more amount than existing bal from ac (
c) Attempting to enter Invalid User name and pwd ...etc

PROGRAM

```
#program cal division of two numbers by accepting the values dynamically
from KBD
#div1.py
s1=input("Enter First Value:")
s2=input("Enter Second Value:")
a=int(s1) #---------------------->problemetic statement---ValueError
b=int(s2)#---------------------->problemetic statement---ValueError
print("Val of a={}".format(a))
print("Val of b={}".format(b))
c=a/b #---------------------->problemetic statement--->ZeroDivisionError
print("Div={}".format(c))
print("....Successfully Executed.....")
```

```
#program cal division of two numbers by accepting the values dynamically
from KBD
#div2.py
try:
      s1=input("Enter First Value:")
      s2=input("Enter Second Value:")
      a=int(s1)
      b=int(s2)
      c=a/b
except ValueError:
      print(" Don't enter Strs / Alpha-numeric / Special Symbols as Values
:")
except ZeroDivisionError:
      print("Don't enter zero for Den....")
else:
      print("============================")
      print("\tResult")
      print("============================")
      print("Val of a={}".format(a))
      print("Val of b={}".format(b))
      print("Div={}".format(c))
      print("============================")
finally:
      print("i am from finally block:")
```

```
#ex1.py
a=10
b=0
print("Line 4 : a={}".format(a))
print("Line 5 : b={}".format(b))
c=a/b  # exception occurs---ZeroDivisionError
print("Line 7 : c={}".format(c))
print("Program executed Sucessfully:")
```

```
#ex2.py
a="$"
print("Val of a(str)=",a)
b=int(a) # invalid input---exception-->object-->ValueError
print("Val of b(int)=",b)
```

```
==================================================
  Handling The Exceptions
==================================================
=>Handling The Exceptions are nothing but converting Technical error
Messages into User-friendly error messages.
=>To Convert Technical error Messages into User-friendly error messages,
In python programming, we have 5 key words. They are
      1) try
      2) except
      3) else
      4) finally
      5) raise
-------------------------------------------------------
syntax for handling the exceptions:
-------------------------------------------------------
      try:
           block of statements causes
           exceptions in the program
      except  <exception class name-1>:
            block of statements provides
            User-Friendly Error Messages
      except  <exception class name-2>:
            block of statements provides
            User-Friendly Error Messages
      -------------------------------------------------
      -------------------------------------------------
      except  <exception class name-n>:
            block of statements provides
            User-Friendly Error Messages
      else:
             block of statements--recommended
             provide display results
      finally:
             block of statements which will
             execute compulsorily
==================================================
1) try:
---------------
=> It is the block , In which we write  block of statements causes
problems at runtime
     In otherwords, what are all the statements generating exceptions then
those statements must be written within try block. Hence try block is
called "exception Monitoring block"
=> When the exception occurs in try block then PVM comes out of try block
and executes        appropriate except block(if found otherwise PVM goes
out of program flow)
```

=> After executing appropriate except block PVM never comes to try block to execute rest of the statements in try block.
=>Each and Every Try block must have alteast one except block(otherwise we can't give User friendly Error Messages)
=>Each and Every Try block must be immediately followed by except block.
==========
2) except block:

=> It is the block , In which we write block of statements provides user-friendly error messages. In otherwords except block supresses the technical error messages and generates user-friendly error messages. Hence this block is called "exception Processingh block".

Note:- Handling the exception = try Block + except block

=> except block will execute provided an exception occurs in try block
=>Even though we write multiple except blocks, PVM can execute appropriate except block depends on on type of exception occurs in try block.
=>Industry is highly recommended to write multiple except blocks for generating multiple User-friendly errors messages.
=>except block to be written after try block and before else block (if we write else block)
==========
3) else block:

=> It is the block , In which we are recommended to write block of statements for generating results and hence this block is called Result block
=> else block will execute provided there is no exception occurs in try block.
=> Writing else block is optional
=>The place of writting else block is after except block and before finally block(if we write finally block)
==========
4) finally block:

=> It is the block , In which we are recommended to write block of statements which will relinquish (close / release / terminate / clean-up) the resources (file / databases) which are obtained in try block.
=>Writting the finally block is optional
=>finally block will execute compulsorily (if we write)
=>finally block to be written after else block (if we write else block)
==========

```
==================================================
           Various forms of except block
==================================================
```
=> We know that except block is used for supresssing the technical error messages and generating User frinedly error messages.
=>We can have various except blocks. They are

Flavour1:- This approach displays the messages caused due that exception occurence

```
--------------
syntax:
--------------
           try:
               -----------
             -----------
             -----------
         except  exception class name as varname:
             print(varname)
```

Example:

```
--------------
      try:
            a=10
            b=0
            c=a/b
      except ZeroDivisionError as kvr:   #  here kvr is a var name holds the msg
            print(kvr)   #  division by zero   # which is generated due to ZeroDivisionError
```

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

Flavour2:- This approach makes us understand single except handles multiple specific exceptions and displays multiple specific Messages at a time

```
--------------
Syntax2:
--------------
           try:
               -----------
             -----------
             -----------
         except ( exception class name1, exception class name2...exception class name-n)
             bock of stataments provides
             User-friendly error Messages.
```

Example:

#program cal division of two numbers by accepting the values dynamically from KBD

```
#div3.py
try:
      s1=input("Enter First Value:")
      s2=input("Enter Second Value:")
      a=int(s1)
      b=int(s2)
      c=a/b
except ( ValueError, ZeroDivisionError) :
      print("\nDon't enter Strs / Alpha-numeric / Special Symbols as
Values :")
      print("\nDon't enter zero for Den....")
else:
      print("============================")
      print("\tResult")
      print("============================")
      print("Val of a={}".format(a))
      print("Val of b={}".format(b))
      print("Div={}".format(c))
      print("============================")
finally:
      print("i am from finally block:")
```

==========

Flavour3:- This approach makes us understand single except we can display generic messages by handling all types of exceptions.

syntax:

```
------------
        try:
              -----------
            -----------
            -----------
        except :
            bock of stataments provides
            generic Messages.
```

Example:

```
--------------
      try:
            a=10
            b=0
            c=a/b
      except :
            print("exception occured:")
```

==========

==========
raise keyword
==========

=>raise kwd is used for hitting / raising / generating the exception which is occured as part of python program when certain condtion is satisfied.

=>Syntax1:-

```
raise    exception class name
```

syntax2:

```
def   functionname(list of formal params if any):
      ------------------------------
      ------------------------------
      if ( test cond ):
              raise exception class name
      ------------------------------
      ------------------------------
```

Program

```
#program cal division of two numbers by accepting the values dynamically
from KBD
#div3.py
try:
      s1=input("Enter First Value:")
      s2=input("Enter Second Value:")
      a=int(s1)
      b=int(s2)
      c=a/b
except ( ValueError, ZeroDivisionError) :
      print("\nDon't enter Strs / Alpha-numeric / Special Symbols as
Values :")
      print("\nDon't enter zero for Den....")
else:
      print("============================")
      print("\tResult")
      print("============================")
      print("Val of a={}".format(a))
      print("Val of b={}".format(b))
      print("Div={}".format(c))
      print("============================")
finally:
      print("i am from finally block:")
```

```
#hyd.py----->treated as module----------(3)
              # (1)               (2)(step1: raising)
  class KvrZeroError(Exception):pass
```

Step2: developing another module which contains main operational program

```
#mathop.py-----treated as a module name
from hyd import KvrZeroError
def    division():
      a=int(input("Enter Value of a:"))
      b=int(input("Enter Value of b:"))
      if (b==0):
            raise KvrZeroError     # hitting / raising an exception by
using raise kwd
      else:
            c=a/b
            print("div={}".format(c))
```

Step3: creating last module

```
#divdemo.py
from mathop import division
from hyd import KvrZeroError
try:
      division()     # handling the exception
except KvrZeroError :
      print("DON'T ENTER ZERO FOR DEN...")
```

CASE STUDY OF ATM PROGRAM

Step1:

```
   class DepositError(BaseException):pass

   class WithDrawError(Exception):pass
   class InSuffFundError(Exception):pass
```

Step2:

```
#atmmenu.py----->treated as module name
def atmmenu():
      print("-----------------------------------")
      print("\tATM Operations:")
      print("-----------------------------------")
      print("\t1. DEPOSIT:")
      print("\t2. WITHDRAW")
      print("\t3. BALANCE ENQ..")
      print("\t4. EXIT")
      print("-----------------------------------")
```

Step3:

```
#bank.py---treated as module
from bankexcept import DepositError
from bankexcept import WithDrawError
from bankexcept import InSuffFundError
acbal=500.00
def   deposit():
      damt=float(input("Enter the amount to deposit:")) #ValueError
      if(damt<=0):
            raise  DepositError
      else:
            global acbal
            acbal=acbal+damt
            print("U Have Deposited Rs:{}".format(damt))
            print("Available Bal after deposit: {}".format(acbal))

def  withdraw():
      wamt=float(input("Enter the amount to withdraw:")) # ValueError
      if(wamt<=0):
            raise WithDrawError
      else:
            global acbal
            if(wamt>acbal):
                  raise InSuffFundError
            else:
                  acbal=acbal-wamt
                  print("Take the cash and enjoy!")
                  print("Available Bal after with draw: {}".format(acbal))

def   balenq():
      print("UR AVAILABLE BALANCE: {}".format(acbal))
```

Step4:

```
   #atmdemo.py
from atmmenu import atmmenu
from bankexcept import DepositError
from  bankexcept import WithDrawError
from bankexcept import InSuffFundError
from bank import deposit
from bank import withdraw
from bank import balenq
import sys
while(True):
      try:
            atmmenu()
            ch=int(input("Enter ur Choice:"))
            if (ch==1):
                  try:
                        deposit()
                  except ValueError:
                        print("Don't try to deposit sts/alpha-numeric value
/ special symbols:")
                  except DepositError:
                        print("Don't try to deposit -ve and zero amount:")
            elif(ch==2):
                  try:
                        withdraw()
                  except ValueError:
                        print("Don't try to withdraw sts/alpha-numeric
value / special symbols:")
                  except WithDrawError:
                        print("Don't try to withdraw  -ve and zero
amount:")
                  except InSuffFundError:
                        print("U don't have sufficient Funds--read
python!")
            elif(ch==3):
                  balenq()
            elif(ch==4):
                  print("Thanks for this App!")
                  sys.exit()
            else:
                  print("Ur Choice of Operation is wrong--Select Properly")
      except ValueError:
            print("Don't enter strs/ alpha-numeric / special symbols as
choice:")
```

==========

FILES IN PYTHON

==========

INDEX:

=>Purpose of Files:

=> Def of File

=>Def of Record:

=>Types of Application in Files

- a) Non-Pesrsistant Applications
- b)Persistant Applications

=>Types of Operations on Files:

- a) Write Operation
- b) Read Operation

=>Types of Files in Python:

- a) Text Files
- b) Binary Files

=>Files Opening Modes:

- a) r
- b) w
- c) a
- d) r+
- e) w+
- f) a+
- g) x

=>Functions used for Writting the data to the file:

- a) write()
- b) writelines()

=>Functions used for Reading the data from file:

- a) read()
- b) read(no. of chars)
- c) readline()
- d) readlines()

=> Random File Accesssing

- a) seek()
- b) tell()

=> Advanced Concepts in Files

- a) Pickling
- b) Un-pickling

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

==========

Introduction to Files of Python

==========

=>The purpose of files is that to store the data permanently.

=>Programatically , To store the data permanently we need file name. Technically file name is one of the named location in Secondary Memory (Hard Disk). Hence File name(s) always resides in secondary memory(HDD)

==========

=>In the context of files , we can develop two types of applications. They are

a) Non-Persistent Applications

b) Persistant Applications

a) Non-Persistent Applications:

=>In this application development, we can read data from key board, store that data main memory and results are displayed on Monitor.

Example: ALL OUR PREVIOUS PROGRAM COMES UNDER NON-PERSISTANT APPLICATIONS

=>The Result stored in Main Memory (RAM) is temporary.

---

b) Persistant Applications:

=>In this application development, we can read data from files / database into main memory and results are displayed on Monitor and they stored once again in files / data base.

=>In real time, we can achieve the persistency (Storing the data permanently) in two ways. They are a) by using files

b) by using data base software (oracle, mysql, postgrey sql, Mongodb, sqlite3, Supersonic data base, DB2, SQL Server...etc)

Note:- Industry is not recommended to use file for data persistency bcoz files of any langauge is not secured. Hence industry always recommends to use Data base software.

==========
Def of File:

=>A file is a collection of Records

Def of Record:

=>Collection of Fields Values is called Record.

Example:- (60,Arindam,JNTU,88.88) is called Record
(70, mallesh, HCU, 88.89) is called Record
==========
Operations on Files
==========
=>On files , we can perform two types of operations. They are
a) Write Operation
b) Read Operation

a) Write Operation:

=>Write Operation is used for Transfering Temporary Data from main memory into file of Secondary Memory
=>To Perform Write Operation, we use 3 steps:
a) Choose the file name
b) Open the file name in Write Mode
c) Perform cycle of write operations

=> While we are performing write operation on files, we get an exception called EOFError,FileExistsError
==========
b) Read Operation:

=>Read Operation is used for retrieving / reading the data from file of secondary memory into object(variable) of main memory

=>To perform read operation, we use 3 steps. They are

a) Choose the file name
b) Open the file name in Read Mode
c) Perform cycle of Read Operations.

=>While we are performing Read Operation, we get a pre--defined exception called FileNotFoundError.

==========
Opening the file
==========
=>In Python to open the file in specified mode, we have 2 syntaxex. They are

1. by using open()
2. by using with....open()....as

---
1. by using open()

Syntax:
---
varname=open("file name", "file mode")

Explanation:

=>var name represents a valid var name and it is treated as file pointer and it can be read pointer or write pointer.
=>open() is a pre-defined function , which is used for opening the file in spefied file opening mode.
=>file mode can be r,a,w,r+,w+,a+ and x
=>With this syntax once we open the file, It highly recommended to close the file in finally block for achieving the consistency.
==========
=
2) by using with....open()....as

Syntax:

```
          with  open("File Name", "file mode") as <varname>:
                    ------------------------------------------
                  ------------------------------------------
                  ------------------------------------------
------------------
Explanation:
------------------
=>here 'with' and ' as ' are key words
=>var name represents a valid var name and it is treated as file
pointer and it can be read pointer or write pointer.
=>open() is a pre-defined function , which is used for opening the
file in spefied file opening mode.
=>file mode can be r,a,w,r+,w+,a+ and x
=>With this syntax once we open the file, It is automatically
closing the file provided the PVM control comes out of with ...
opne()...as   syntax
================================================================


File Opening Modes
          ========================================
=>File Opening Modes are used for specifying in which mode the file
opened
=>In python , we have 7 file opening modes. They are

1)  r :
-----------
=>'r' is used for opening the specified file in Read Mode provided
the specified file must exists in secondary memory otherwise we get
exception called FileNotFoundError
=>If we don't specify the file opening mode then by default file
opening mode is 'r'
------------
2) w :
-----------
=>'w' is used for opening the specified file in write mode newly
always. If we open existing file in 'w' mode then existing file
also opened in write mode newly by removing existing records.
-----------
3) a :
-----------
=>'a' mode is used opening the spefcied file in write mode, if file
does not exists add the records. If the file already exists then it
```

opened in write mode and new records will be appended.
---
4) r +

=>'r+' mode is used for opens a file for performing reading and writting.
---
5) w +

=>'w+' mode is used for opening the file in write mode always newly and performs read and write operations. If the existing file is opened in 'w+' mode then that file opened in write mode newly by removing existing records
---
6) a +

=>=>'a+' mode is used for opening the file in write mode provided if it is new file and performs read and write and append operations. If the existing file is opened in 'a+' mode then that file opened in write mode and we can write , read and append operations.
---
7) x :

=>'x' mode is used for opening the file in write mode exclusively.
==========

Types of Files in Python

=>In Python, we have two types of files. They are
1. Text Files
2. Binary Files

---
1) Text File:

=>Text files contains the data in the form Alphabets, digits and special symbols
Example:-
sum.py, sum.java,
sum.c sum.cpp
resume.rtf.....etc
=>In python Text files are denoted by a letter 't'

2)Binary Files:

=>Binary Files contains the data in the form bytes or bits (binary format)
Examples:- .exe files
.jpeg, .jpg, .png, .gif -->Images
audio video files...etc

=>In python, Binary files are denoted by a letter 'b'
=>In python Programming, if we did't specify the type of the file then it is by default treated as text file.

Reading the data from files

=>To read data from files, First we must open the file in read mode.
=>To read the data from the file, we have 4 functions. They are
a) read()
b) read(no. of chars)
c) readline()
d) readlines()

a) read():

=>This function is used for reading entire data from the file in the form of str

syntax:

```
data=filepointer.read()
```

---
b) read(no.of chars):

=>This function is used for reading specified no. of chars from the file in the form of str.
syntax:

```
     data=filepointer.read(no. of chars)
```
---
c) readline()

=>This function is used for reading one line at a time from the file in the form of str.

syntax:

```
     data=filepointer.readline()
```
---
d) readlines():

=>This function is used for reading entire lines of data from the file in the form of list type
syntax:

```
     data=filepointer.readlines()
```

==========
Writting the data to the file
==========
=>To write the data to the file, the file must be opened in write mode (w,a,w+,a+, x)
=>To write the data to the file, we have two pre-defined functions. They are
1) write(str)
2) writelines(str)

1) write(str):

=>This function is used for writting any type of data to the file

in the form of str

Syntax:- filepointer.write(str data)

---

2) writelines(str):

=>This function is used for writting any iterable object data to the file in the form of str

syntax:- filepointer.writelines(iterabel-object)

Example:-

```
lst=[10,"KVR",94.25,"HYD"]
tpl=(20,"RS",99.99,"NL")
wp.writelines(lst)
wp.writelines(tpl)
```

Pickling and Un-pickling

Pickling:

=>Let us assume there exists an iterable object with collection of values. If we want to store / save them in file of secondary memory with normal file programming then we need perform multiple write operations. This process is one of the time consuming Processes.
=>To overcome this problem, we use the concept of Pickling.
=>The advantage of Pickling concept is that with Single Write Operation, we save / store entire iterable object content in the file of secondary memory.

=>Def. of Pickling:

=>The process of storing / saving entire iterable object content into the file of secondary memory by performing single write Operation is called Pickling.
=>Pickling Concept participates in Write operation.

=>Steps for implementing Pickling Operation:

=> import pickle module

=>Transfer / save / store the object data into file by performing single write operation by using dump(), which is present in pickle

syntax: pickle.dump(objectname, file pointer)

=>Pickling and un-pickling concepts always deals with any file mode with binary file

==========

2) Un-pickling:

=>Let us assume, there exists a record in file of secondary memory. To read record values, with normal file programming, we need to perform multiple read operations. which is one of the time consuming process.

=>To overcome this problem, we must use the concept of un-pickling.

=>The advantage of Un-pickling concept is that with single read operation, we read entire record from the file of secondary memory.

=>Def. of Un-Pickling:-

=>The process of reading the entire record from the file of secondary memory into object of main memory by performing single read operation is called Un-pickling.

=>Un-Pickling Concept participates in Read operation.

=>Steps for implementing UnPickling Operation:

==========

=> import pickle module

=>Transfer /read the record from the file of secondary memory into the object of main memory by using load(), which is present in pickle module

syntax: objectname=pickle.load(file pointer)

Note:

=>Pickling and un-pickling improves the performance of ordinary file programming of python.

==========

PROGRAMS:

#FileEx1.py

```
try:
     rp=open("stud.data")
     print("type of rp=", type(rp))   # TextIOWrapper
except FileNotFoundError:
     print("Files does not exists:")
else:
     print("File opened in read mode successfully:")
     print("-------------------------------------------------------
------")
     print("File opening Mode=",rp.mode)  # r
     print("is readable ?=",rp.readable()) # True
     print("is writable ?=",rp.writable()) # False
     print("is files closed?=", rp.closed) # false
     print("-------------------------------------------------------
------")
finally:
     if rp!=None:

          print("File closed successfully:")
          rp.close()
          print("Line-19-->is files closed?=", rp.closed) # True
```

#fileex2.py

```
wp=open("stud.data","w")
print("File Opened in write mode successfully:")
print("Type of wp=",type(wp))  # TextIOWrapper
print("-------------------------------------------------------------
-")
print("File opening Mode=",wp.mode)  # w
print("is readable ?=",wp.readable()) #False
print("is writable ?=",wp.writable()) # True
print("-------------------------------------------------------------
-")
```

#fileex3.py

```
wp=open("emp.data","r+")
print("File Opened in write mode successfully:")
print("Type of wp=",type(wp))  # TextIOWrapper
print("-------------------------------------------------------------
-")
print("File opening Mode=",wp.mode)  # a+
print("is readable ?=",wp.readable()) # True
print("is writable ?=",wp.writable()) # True
print("-------------------------------------------------------------
-")
```

#fileex4.py

```
try:
     wp=open("atten1.data","x")
     print("File Opened in write mode successfully:")
     print("Type of wp=",type(wp))  # TextIOWrapper
     print("-------------------------------------------------------
------")
     print("File opening Mode=",wp.mode)  # x
     print("is readable ?=",wp.readable()) # False
     print("is writable ?=",wp.writable()) # True
     print("-------------------------------------------------------
------")
except FileExistsError:
     print("File already opened in write mode, we can't open
again")
```

#FileEx5.py

```
try:
     with open("stud.data","r") as rp:
          print("File opened in read mode successfully:")
          print("-------------------------------------------------
-----------")
          print("File opening Mode=",rp.mode)  # r
          print("is readable ?=",rp.readable()) # True
          print("is writable ?=",rp.writable()) # False
          print("line-9-->is files closed?=", rp.closed) # false
          print("-------------------------------------------------
-----------")
```

```
     print("i am out of with open()...line-11 is files closed?=",
rp.closed) # True
except FileNotFoundError:
     print("File does not exists:")
else:
     print("i am out of with open()...line-15 is files closed?=",
rp.closed) # True
finally:
     print("i am out of with open()...line-17 is files closed?=",
rp.closed) # True
```

#FileReadEx1.py

```
rp=open("hyd.data","r")
fdata=rp.read()
print(fdata)
print("\n\ntype of fdata=",type(fdata))
```

```
#FileReadEx2.py
rp=open("hyd.data","r")
fdata=rp.read(3)
print(fdata)
print("-----------------------------")
print("Pos of rp=", rp.tell())
fdata=rp.read(6)
print(fdata)
print("Pos of rp=", rp.tell())
rp.seek(0)
print("after seek, Pos of rp=", rp.tell())
```

#FileReadEx3.py

```
rp=open("hyd.data","r")
fdata=rp.readline()
print(fdata)
fdata=rp.readline()
print(fdata)
print("type of fdata=",type(fdata))
```

```
#FileReadEx4.py
rp=open("hyd.data","r")
fdata=rp.readlines()
for line in fdata:
     print(line, end=" ")


#program for wrting the data to the file
#FileWriteEx1.py
wp=open("addr.data","a")
wp.write("Sai Kumar\n")
wp.write("14,KPHB\n")
wp.write("HYD\n")
print("Data written to the file--verify")


#FileWriteEx2.py
wp=open("emp.data","a")
lst=[30,"Krishna",44.25,"Bang"]
tpl=(40,"Sampath","AP")
wp.writelines(str(lst)+"\n")
wp.writelines(str(tpl)+"\n")
s="Python\n is an\n oop lang"
wp.writelines(s)
print("Data written to the file--verify")


#FileWriteEx3.py
with open("hyd.data","a") as wp:
     print("Enter Multiple Lines of Text (To stop press stop)")
     while(True):
          std=input()
          if (std!="stop"):
               wp.write(std+"\n")
          else:
               print("\n data written to the file sucessfully--
verify")
               break
```

```
#pick.py
import pickle
with open ("emp.data","ab") as wp:
     noe=int(input("Enter How Many Emplyees data u have:"))
     for i in range(1, noe+1):
          print("-------------------------------------------------
----")
          print("Enter {} Employee Data:".format(i))
          print("-------------------------------------------------
----")
          eno=int(input("Enter Employee Number:"))
          ename=input("Enter Employee Name:")
          esal=float(input("Enter Employee Salary:"))
          lst=list()
          lst.append(eno)
          lst.append(ename)
          lst.append(esal)
          pickle.dump(lst,wp)
          print("-------------------------------------------------
----")
          print("{} Emp Data Record Saved in a file
successfully:".format(i))
          print("-------------------------------------------------
----")


#unpick.py
import pickle
try:
     with open("emp.data","rb") as rp:
          print("-----------------------------------------")
```

```
          print("\tEmplyee Records")
          print("-----------------------------------------")
          while(True):
               try:
                    emprec=pickle.load(rp)
                    for val in emprec:
                         print("{} ".format(val), end="  ")
                    print()
               except EOFError:
                    print("---------------------------------------
-")
                    break
except FileNotFoundError:
     print("File does not exists")
#FileCopy.py
sfile=input("Enter The source File:")
try:
     with  open(sfile) as rp:
          dfile=input("Enter Destination File:")
          with open(dfile,"a") as wp:
               sfdata=rp.read()
               wp.write(sfdata)
               print("{}  data  copied into {}--plz verify
".format(sfile,dfile))
except FileNotFoundError:
     print("Source File does not exists")



copy binary file content into another binary file
#BinCopy.py
bsfile=input("Enter Binary Source File:")
try:
     with open(bsfile, "rb" ) as rp:
          bdfile=input("Enter Binary Dest File:")
          with open(bdfile,"w+b") as wp:
               sfdata=rp.read()
               wp.write(sfdata)
               print("{}  data copied into {}--plz
verify".format(bsfile,bdfile))
except FileNotFoundError:
     print("Source File Does not exists:")
```

```
            ======================================
                 os module in python
            ======================================
=>"os" is one the pre-defined module in python
=>The purpose of this module to perform certain operations on
folders / directories.
=>To perform certain operations on folders / directories, First we
must import "os" module
=>The "os" module various functions to perform certain operations
on folders / directories such  as
           a) getting the current working Directory
           b) Creating a folder
           c)renaming folders
           d) removing folders
           e) obtaing the files of folders...etc
============================================================
Getting Current Working Folder
============================================================
=>To get current working folder, we use getcwd()
=>Syntax:
                varname=os.getcwd()
Examples:
---------------------
import os
cwdinfo=os.getcwd()
print(cwdinfo)
============================================================
```

=======
Creating a Folder:
==========
=>To create a Folder , we use mkdir()

syntax:- os.mkdir("folder name")

=>mkdir() can create one folder at a time but not root folder, sub folder, sub-sub folder etc

=>If the folder was already created then it can't create again and we get pre-feind exception
FileExistsError
=>If the The Hierarchy of Folders does not exists then we OSError as exception
---
Example:

```
import os
try:
     os.mkdir("g:\Mango\apple\kiwi")
     print("Folder Created--Verify")
except FileExistsError:
     print("Folder was already created-it can't create again")
except OSError:
     print("The Hierarchy of Folders unable create:")
```
==========
Creating Root Folder, sub Folder, sub-sub Folders ecta at time
==========
=>To Creating Root Folder, sub Folder, sub-sub Folders etc at time, we use a pre-defined function called makedirs()

Syntax:- os.makedirs("Folder Hierarchy")
=>Folder Hierarchy represents information about RootFolder\subfolder\sub-sub folder...etc

---
Example:

```
import os
try:
     os.makedirs("G:\mango/apple/grapes")
     print("Folder Hierarchy Created..")
```

```
except FileExistsError:
     print("Folder Hirrachy was already created:")
================================================================
=
Remove a folder / Directory:
=========================================================
=>To Remove a folder / Directory, we use rmdir()
=>Syntax:-      os.rmdir("folder name")

=>rmdir() removes the single folder , provided folder name should
not contain any files otherwise we get an exception called OSError.
=>If folder name does not exists we get FileNotFoundError as
exception.
=>rmdir() can remove one folder at a time.
------------------
Example:
------------------
import os
try:
     os.rmdir("G:\KV234")
     print("Folder Removed-verify")
except FileNotFoundError:
     print("Folder Name does not Exists")
except OSError:
     print("Specified  Folder Contains files-can't be removed")
============================================================
Remove Root folder, sub folder, sub-sub folders   etc
============================================================
=>To Remove Root folder, sub folder, sub-sub folders   etc, we use
removedirs()

Syntax:-    os.removedirs("folder Hierarchy")

=>Here folder Hierarchy represents Root Folder\sub folder\sub-sub
folder etc
----------------
Example:
---------------
import os
try:
     os.removedirs("G:\mango/apple/kiwi")
     print("Folder(s) removed--verify")
except FileNotFoundError:
```

```
     print("specifed folder hierarchy does not exists")
except OSError:
     print("Specified Folder is not empty:")
```

==========

To rename a folder

==========

=>To rename a folder , we use rename()

```
Syntax:-    os.rename("older Folder name", "new Folder name"))
---------------
Example:
---------------
import os
try:
     os.rename("Hyderabad","HYD")
     print("Folder Renamed---Verify")
except FileNotFoundError:
     print("Old Folder Does not exists")
```

==========

To find the content of a folder:

==========

=>To find content of a folder , we use listdir()

```
syntax:-    os.listdir("folder hierachy")
```

=>This function shows content of speficied folder only but not showing the content of sub folder and sub -sub folders.

=>if "folder hierachy" does not exists we get FileNotFoundError

Example:

```
--------------------
#listdirex1.py
import os
lst=os.listdir(".")
for x in lst:
     print(x)
--------------------------
```

==========

To find the content of a Root folder, sub folder and sub -sub folders:

==========

=>To find the content of a Root folder, sub folder and sub -sub folders:, we use a walk()

Syntax:

```
------------
          os.walk("Folder Hierarchy")

=>if "folder hierachy" does not exists we get FileNotFoundError
================================================================
Example
-------------
#listfiles.py
import os
kvr=os.walk(".")
for k,v,r in kvr:
     print("Folder path={}".format(k))
     print("sub folder path={}".format(v))
     print("Files={}".format(r))
```

PROGRAMS

```
#createfoldex1.py
import os
try:
     os.mkdir("g:\Mango\apple\kiwi")
     print("Folder Created--Verify")
except FileExistsError:
     print("Folder was already created-it can't create again")
except OSError:
     print("The Hierarchy of Folders unable create:")

#cwdex1.py
import os
cwdinfo=os.getcwd()
print(cwdinfo)



#listdirex1.py
import os
lst=os.listdir(".")
for x in lst:
     print(x)
```

```
#listfiles.py
import os
kvr=os.walk(".")
for k,v,r in kvr:
     print("Folder path={}".format(k))
     print("sub folder path={}".format(v))
     print("Files={}".format(r))


#renamefoldex1.py
import os
try:
     os.rename("Hyderabad","HYD")
     print("Folder Renamed---Verify")
except FileNotFoundError:
     print("Old Folder Does not exists")


#rmdiresx.py
import os
try:
     os.removedirs("G:\mango/apple/kiwi")
     print("Folder(s) removed--verify")
except FileNotFoundError:
     print("specifed folder hierarchy does not exists")
except OSError:
     print("Specified Folder is not empty:")



#rmdirex.py
import os
try:
     os.rmdir("G:\KV234")
     print("Folder Removed-verify")
except FileNotFoundError:
     print("Folder Name does not Exists")
except OSError:
     print("Specified  Folder Contains files-can't be removed")
```

==========

Introduction to Regular Expressions in Python

==========

=>Regular Expressions one of the Programming languages Independent Concept and This concept can be used any programming lang.

=>Def of Regular Expressions:

=>A Regular Expressions is one of the string data, which is searching / matching / finding in the Given Data and obtains

Desired Result.
==========
=>To deal with Regular Expressions Programming, we use a pre-defined module called "re"
=>The "re" module contains some pre-defined functions, which are used develop regular exprssions applications.
==========
Example:

Q) Search for thr word "Python"

Search pattern ="python"
01234567890..... 20.....
Given Data: "Python is an OOP lang. Python is also POP lang"

Result
---
0 6----------Python
20 26----------Python
No. of occurence = 2

Example: Given Data: " aniket is studen of python whose mail id is aniket@ibm.com and
ramesh is student of java whose mail id is ramesh@tcs.com, ram is student data scienece whose mail id is ram@wipro.com. "

Q) find all mail ids

Given data="AaUu@43at#qW"
Q1) find capital alphabets only
Q2) find small alphabets only
Q3) find special symbols only
Q4) find small and cap alphabets
Q5) find digits only----------etc
==========
Programmer-Defined Character Classes
==========
=>Character Classes Prepared By Programmer according to search pattern which is used search in the Given Data.
=>The complete Programmer-Defined Character Classes are given Bellow

1) [kvr]--->search for either 'k' or 'v' or 'r'
2) [^kvr]--->search for all except 'k' or 'v' or 'r'
3) [a-z]--->search for all small alphabets only
4) [^a-z]--->search for all except small alphabets
5) [A-Z]--->search for all capital alphabets only
6) [^A-Z]--->search for all except capital alphabets
7) [0-9]----->search for all digits only
8) [^0-9]----->search for all except digits only
9) [A-Za-z]--->search for capital and small alphabets only
10) [^A-Za-z]--->search for all except capital and small alphabets only
11) [A-Za-z0-9]--->search for capital and small alphabets and digits only(except special synmbols)
12) [^A-Za-z0-9]--->search for only special symbol

PROGRAM

```
#regexprex1.py
import re
strdata="Python"
givendata="Python is an OOP Lang. Python is an POP lang also"
matinfo=re.finditer(strdata,givendata)
print("------------------------------------------------------------
-----")
for mat  in matinfo:
     print("start Index: {}\t End Index: {}\t Value:
{}".format(mat.start(),mat.end(),mat.group()))
else:
     print("----------------------------------------------------
----------")
```

```
#regexprex2.py
#program for seraching the word "python"
```

```
# in the line "Python is an OOP Lang. Python is an POP lang also"
import re
noc=0
matinfo=re.finditer("a","Python is an OOP Lang. Python is an POP
lang also")
print("------------------------------------------------------------
------")
for mat  in matinfo:
     noc+=1
     print("start Index: {}\t End Index: {}\t Value:
{}".format(mat.start(),mat.end(),mat.group()))
else:
     print("------------------------------------------------------
-----------")
     print("No. of Occurences={}".format(noc))
     print("------------------------------------------------------
-----------")
```

```
#regexprex3.py
import re
matinfo=re.finditer("[kvr]","Ak#4v@R5$Vw%rP")
print("------------------------------------------------------------
------")
for mat  in matinfo:
     print("start Index: {}\t End Index: {}\t Value:
{}".format(mat.start(),mat.end(),mat.group()))
else:
     print("------------------------------------------------------
-----------")
```

```
#regexprex4.py
import re
matinfo=re.finditer("[^kvr]","Ak#4v@R5$Vw%rP")
```

```
print("-------------------------------------------------------------
------")
for mat  in matinfo:
     print("start Index: {}\t Value:
{}".format(mat.start(),mat.group()))
else:
     print("-------------------------------------------------------
-----------")


#regexprex5.py
#search for all small alphabets only
import re
matinfo=re.finditer("[a-z]","Ak#4v@aR5g$Vw%rP")
print("-------------------------------------------------------------
------")
for mat  in matinfo:
     print("start Index: {}\t Value:
{}".format(mat.start(),mat.group()))
else:
     print("-------------------------------------------------------
----------")



#regexprex6.py
#search for all except small alphabets only
import re
matinfo=re.finditer("[^a-z]","Ak#4v@aR5g$Vw%rP")
print("-------------------------------------------------------------
-----")
for mat  in matinfo:
     print("start Index: {}\t Value:
{}".format(mat.start(),mat.group()))
else:
     print("-------------------------------------------------------
---------")




#regexprex7.py
```

```
#search for all Capital alphabets only
import re
matinfo=re.finditer("[A-Z]","Ak#4v@aR5g$Vw%rP")
print("------------------------------------------------------------
----")
for mat  in matinfo:
     print("start Index: {}\t Value:
{}".format(mat.start(),mat.group()))
else:
     print("------------------------------------------------------
---------")


#regexprex8.py
#search for all  except Capital alphabets only
import re
matinfo=re.finditer("[^A-Z]","Ak#4v@aR5g$Vw%rP")
print("------------------------------------------------------------
------")
for mat  in matinfo:
     print("start Index: {}\t Value:
{}".format(mat.start(),mat.group()))
else:
     print("------------------------------------------------------
-----------")


#regexprex9.py
#search for all  digits only
import re
matinfo=re.finditer("[0-9]","Ak#4v@aR5g$Vw%8rP")
print("------------------------------------------------------------
----")
for mat  in matinfo:
     print("start Index: {}\t Value:
{}".format(mat.start(),mat.group()))
else:
     print("------------------------------------------------------
---------")

#regexprex10.py
#search for all except  digits only
import re
```

```
matinfo=re.finditer("[^0-9]","Ak#4v@aR5g$Vw%8rP")
print("------------------------------------------------------------
------")
for mat  in matinfo:
     print("start Index: {}\t Value:
{}".format(mat.start(),mat.group()))
else:
     print("------------------------------------------------------
---------")
#regexprex11.py
#search for all Alphabets--small and capital
import re
matinfo=re.finditer("[A-Za-z]","Ak#4v@aR5g$Vw%8rP")
print("------------------------------------------------------------
-----")
for mat  in matinfo:
     print("start Index: {}\t Value:
{}".format(mat.start(),mat.group()))
else:
     print("------------------------------------------------------
----------")


#regexprex12.py
#search for all  except Alphabets--small and capital
import re
matinfo=re.finditer("[^A-Za-z]","Ak#4v@aR5g$Vw%8rP")
print("------------------------------------------------------------
-----")
for mat  in matinfo:
     print("start Index: {}\t Value:
{}".format(mat.start(),mat.group()))
else:
     print("------------------------------------------------------
---------")


#regexprex13.py
#search for all  except Alphabets--small and capital
import re
matinfo=re.finditer("[A-Za-z0-9]","Ak#4v@aR5g$Vw%8rP")
print("------------------------------------------------------------
-----")
```

```
for mat  in matinfo:
     print("start Index: {}\t Value:
{}".format(mat.start(),mat.group()))
else:
     print("-----------------------------------------------------
---------")

#regexprex14.py
#search for all  except Alphabets--small and capital
import re
matinfo=re.finditer("[^A-Za-z0-9]","Ak#4v@aR5g$Vw%8rP")
print("-----------------------------------------------------------
-----")
for mat  in matinfo:
     print("start Index: {}\t Value:
{}".format(mat.start(),mat.group()))
else:
     print("-----------------------------------------------------
---------")
#regexprex15.py
#program searching space character
import re
matinfo=re.finditer("\s", " A#6aB^ 7@H Pa")
print("-----------------------------------------------------------
-----")
for mat in matinfo:
     print("start index={}\t
value={}".format(mat.start(),mat.group()))
print("-----------------------------------------------------------
-----")

#regexprex16.py
#program  searching all except space character
import re
matinfo=re.finditer("\S", " A#6aB^ 7@H Pa")
print("-----------------------------------------------------------
-----")
for mat in matinfo:
     print("start index={}\t
value={}".format(mat.start(),mat.group()))
print("-----------------------------------------------------------
-----")
```

```
#regexprex17.py
#program  searching digit character
import re
matinfo=re.finditer("\d", " A#6aB^ 7@H Pa")
print("-------------------------------------------------------------
-----")
for mat in matinfo:
     print("start index={}\t
value={}".format(mat.start(),mat.group()))
print("-------------------------------------------------------------
-----")
```

```
#regexprex18.py
#program  searching for all except  digit
import re
matinfo=re.finditer("\D", " A#6aB^ 7@H Pa")
print("-------------------------------------------------------------
-----")
for mat in matinfo:
     print("start index={}\t
value={}".format(mat.start(),mat.group()))
print("-------------------------------------------------------------
-----")
```

```
#regexprex19.py
#program  searching for all word character
import re
matinfo=re.finditer("\w", " A#6aB^ 7@H Pa")
print("-------------------------------------------------------------
-----")
for mat in matinfo:
     print("start index={}\t
value={}".format(mat.start(),mat.group()))
print("-------------------------------------------------------------
```

```
-----")


#regexprex20.py
#program  searching for all special symbols except word character
import re
matinfo=re.finditer("\W", " A#6aB^ 7@H Pa")
print("------------------------------------------------------------
-----")
for mat in matinfo:
     print("start index={}\t
value={}".format(mat.start(),mat.group()))
print("------------------------------------------------------------
-----")


#regexprex21.py
#program  searching for all .
import re
matinfo=re.finditer(".", " A#6aB^ 7@H Pa")
print("------------------------------------------------------------
-----")
for mat in matinfo:
     print("start index={}\t
value={}".format(mat.start(),mat.group()))
print("------------------------------------------------------------
-----")


#regexprex22.py
#program  for searching only 'a'
import re
matinfo=re.finditer("a", "abaabaaab")
print("------------------------------------------------------------
-----")
for mat in matinfo:
     print("start index={}\t
value={}".format(mat.start(),mat.group()))
print("------------------------------------------------------------
-----")
```

```
#regexprex23.py
#program  for searching one 'a' or more 'a' s
import re
matinfo=re.finditer("a+", "abaabaaab")
print("----------------------------------------------------------
-----")
for mat in matinfo:
     print("start index={}\t
value={}".format(mat.start(),mat.group()))
print("----------------------------------------------------------
-----")

#output
#start Index       end index   values
#    0                    1                  a
#    2                     4                 aa
#    5                     8                 aaa


#regexprex24.py
#program  for searching zero or one 'a' or more 'a' s
import re
matinfo=re.finditer("a*", "abaabaaab")
print("----------------------------------------------------------
-----")
for mat in matinfo:
     print("start index={}\t
value={}".format(mat.start(),mat.group()))
print("----------------------------------------------------------
-----")



#regexprex25.py
import re

result=re.search("python","python is an oop lang. python is an pop
lang")
if result!=None:
```

```
     print("Search is successful")
else:
     print("Search not successful:")

#findall()-----finding all matches and returns in the form of list
object
#finditer()----finding all matches and returns in the form of
<Callable_Iterator> object

#serarch()--->finding first match only and it never search futher
matches if we have.
                #---->If not matching it return None


#mobilenovalid.py
import re
while True:
     mno=input("Enter Ur Mobile Number:")
     if (len(mno)==10):
          sres=re.search("\d{10}", mno)
          if sres!=None:
               print("Ur mobile Number Valid:")
               break
          else:
               print("Ur Mobile Number is Invalid-bcoz It should
not contain str/ special symbols")
     else:
          print("Mobile Must Contain 10 digits")



#namesmarks.py
import re
studinfo="Arindam got 90 marks, Gosling got 99 marks, Rossum got 99
marks, Ramesh got 66, Ram got 88 marks and Jay got 77 marks"
print("-------------------------------------------------")
print("Student Marks")
print("-------------------------------------------------")
markslist=re.findall("\d{2}", studinfo)
for marks in markslist:
     print("\t{}".format(marks))
print("-------------------------------------------------")
print("Student Names")
```

```
print("-------------------------------------------------")
names=re.finditer("[A-Z][a-z]+", studinfo)
for name in names:
     print("\t{}".format(name.group()))
print("-------------------------------------------------")
```

==========

Pre-Defined Character Classes in Regular Expression

==========

=>These classes are already defined in Regular expressions and they are used for preparing different search patterns.
=>Pre-Defined Character Classes in Regular Expression are given bellow.

1) \s ------>Search for space character only
2) \S------>Search for all except space character
3) \d------>Search for digit only
4) \D----->Search for all except digit
5) \w----->Search for word character [A-Za-z0-9]   (except special symbols)
6) \W--->Search for special symbols except word character [^A-Za-z0-9]
7) . ------>search for all characters

==========

==========

Quantifiers in Regular Expression

==========

=>Quantifiers in Regular Expression are used for finding number of occurences of special symobol / alphabet / digit.
=>We have the Quantifiers in Regular Expressions. They are

1) 'a'---->searching exactly for 'a'
2)   a+--->searching for either one 'a' or more 'a' s
3)   a*--->searching for either zero or one 'a' or more 'a' s

==========

Note:

---

1) \ddddddddddd   or   \d{10}
2) [A-Za-z]+
3) \d{2}.d+

==========

Python Data Base Communication (PDBC)

(OR)

Communication Between Python and Data Base Softwares

==========

=>As we know that we achieved data persistency through the concept of files and files are having some limitations.

1) Files of any Language are un-secured bcoz Files does not provide user Name and password.
2) Files concept does not allows us to store large volume of data.
3) The data of the file can be manipulated by any un-authrized users bcoz Files does not provide user Name and password.
4) Querying the data from multiple file is complex
5) Structure of the File may differ from One OS to another OS.

=>To overcome the above problems of files with python language, we prefer the achieve data persistency with any RDBMS(Relational Data Base Management System) products(Oracle,MySQL, SQL Server, DB2, PostGrey SQL, super sonic DB...etc)

==========
Adavtanges of Database Softwares(RDBMS):
---
1) Data Base software are fully Secured , bcoz Database software provides Security in the form of User Name and Password.
2) Data Base Softwares allows us to store large volume of data.
3) The data of the Data Base Softwares canot be manipulated by any un-authrized users bcoz Data Base Softwares provides user Name and password.
4) Querying the data from multiple tables of Data Base Softwares is simple.
5) Structure of the data base software will not differ from One OS to another OS. (or) Data base software are OS Independent.
==========
Data Base Software:

=>The data stored in data base softwares in the form tables.
=>First we need to create Tables to store in data base software.
==========
Pyhton association with Data Base software:
==========
=>If Python programming want to communicate with any data base software then we must INSTALL A SEPARATE MODULE of a perticular module of data base software.

=>To INSTALL A SEPARATE MODULE in python , we use tool called pip. the pip is present in C:\Users\Nareshit\AppData\Local\Programs\Python\Python39\Scripts

Syntax for using pip tool:

---
          pip install module name

=>If python Programming want to communicate Oracle Data base Software, we must install cx_Oracle module.

Example: pip   install cx_Oracle
Some times we get " 'pip' is not recognized as an internal or external command,"
To eliminate this OS error, we must set path

set path=C:\Users\Nareshit\AppData\Local\Programs\Python\Python39\Scripts

---

==========

Steps for Developing Python Program to Communicate with Data Base softwares(Oracle)

==========

1) import appropriate module of Data base Software and other modules if required.

Example: import cx_Oracle

2) Python program must obtain connection from the data base software.

3) Create an object of cursor in python program, which is used to carry the query from python program to data base

sodtware and Brings the result from data base software to the python program.

4) Design the Query, Place the Query in the Cursor object and send Query to data base software by using Cursor object and execute.

5) Python Program Process the Result , which came from Data Base Software.

6) Python Program Close the Connection.

==========

Explanation:

---

1) import appropriate module of Data base Software and other modules if required.

Example: import cx_Oracle

---

2) Python program must obtain connection from the data base software.

=> If python program want to communicate with any data base software, we use connect(), which is present cx_Oracle module

Syntax:- cx_Oracle.connect("username/password@DNS/serviceid")

=>User Name---->Data base user name-----My System(KVR)---scott

=>password-----> Data base password------My System (KVR)---tiger

=>DNS(Domain Naming Service)------->Name / IP address of the machine where

data base software installed.

---->Default name of machine is "localhost"

---->default IPaddress of every PC is 127.0.0.1 (loop back address)

=>service id---->On which name, data base software is available in our machine(alias to the original data base).

Q) How to find service-id of our data base

```
                    SQL> select *   from global_name;
                    ------------------------------------------
---------
                    output:
                    ------------
                             global name
                           --------------------
                                orcl
```
Example:- Get the connection Oracle data base

```
               con=cx_Oracle.connect("scott/tiger@localhost/orcl")
               print(type(con)------><class, cx_Oracle.Connection>
               print("Python program objatins connection Oracle
db")
```

==========
===
3) Create an object of cursor in python program, which is used to carry the query from python program to data base sodtware and Brings the result from data base software to the python program.

=>To implement step(3), we use a pre-defined function called cursor(), which is present in Connection class

Example:-

```
               cur=con.cursor()
               print("type of cur=", type(cur))-----<class
cx_Oracle.Cursor>
```

==========
=====
4) Design the Query, Place the Query in the Cursor object and send Query to data base software by using Cursor object and execute.
==========
DDL Queries(Data Definition Language):

=>create, drop, alter...etc---

DML Queries(Data Maniplulation Langauge):

```
====>insert,delete,update

DRL Queries(Data Retrival Langauge):
-------------------------------------------------------------
====>select
-------------------------------------------------------------------
-------
Example:
---------------
 DDL Operations
 --------------------------
qry="create table   student( sno number(3), sname  varchar2(10),
marks number(5,2) )"
cur.execute(qry)
qry="drop table student"

# add some additional column to the student table
     qry="alter table student add(cname varchar2(10))
# modify the size of sname of student table
     qry="alter table student modify(sname varchar2(15))

-------------------------------------------------------------------
-------------------------
DML Operations:
---------------------------------
=>These operations are used performing Operations on data or
records
=>we have 3 types of DML operations.
          1) insert    2. delete    3. update
=>Once we execute any DML operation from Python program , we must
commoit()
1. Insert:
---------------
qry="insert into student values (10,'KVR',94.25,'OUCET')"
cur.execute(qry)
con.commit()
print("One Record Inserted--successfully")
```

PROGRAMS

```
#altertab1.py
import cx_Oracle
try:
     con=cx_Oracle.connect("scott/tiger@localhost/orcl")
     print("Connection Obtained from Oracle DB")
     print("-------------------------------------------------------
--")
     cur=con.cursor()
     kvrqry="alter table student add(cname varchar2(10))"
     cur.execute(kvrqry)
     print("Student Table altered in Oracle Db--verify")
except cx_Oracle.DatabaseError as de:
     print(de)
finally:
     if con!=None:
          print("DB connection Closed:")
          con.close()


#altertab2.py
import cx_Oracle
try:
     con=cx_Oracle.connect("scott/tiger@localhost/orcl")
     print("Connection Obtained from Oracle DB")
     print("-------------------------------------------------------
--")
     cur=con.cursor()
     kvrqry="alter table student modify(sname varchar2(15))"
     cur.execute(kvrqry)
     print("Student Table altered in Oracle Db--verify")
except cx_Oracle.DatabaseError as de:
     print(de)
finally:
     if con!=None:
          print("DB connection Closed:")
```

```
          con.close()
```

```
#createtab.py
import cx_Oracle
try:
     con=cx_Oracle.connect("scott/tiger@localhost/orcl")
     print("Connection Obtained from Oracle DB")
     print("-----------------------------------------------------
--")
     cur=con.cursor()
     kvrqry="create table student( sno number(3), sname
varchar2(10), marks number(5,2) )"
     cur.execute(kvrqry)
     print("Student Table created in Oracle Db--verify")
except cx_Oracle.DatabaseError as de:
     print(de)
finally:
     if con!=None:
          print("DB connection Closed:")
          con.close()
```

```
#dbcontest.py
import cx_Oracle
try:
     kvrcon=cx_Oracle.connect("scott/tiger@localhost/kvrorcl")
except cx_Oracle.DatabaseError as da:
     print("Problem in Getting Connection from Oracle DB")
else:
     print("Python Program obtains connection from Oracle Data
base--Successfully:")
finally:
     if kvrcon!=None:
          print("Db connection Closed:");
          kvrcon.close()
```

```
#droptab.py
import cx_Oracle
try:
     con=cx_Oracle.connect("scott/tiger@localhost/orcl")
     print("Connection Obtained from Oracle DB")
     print("------------------------------------------------------
--")
     cur=con.cursor()
     kvrqry="drop table student"
     cur.execute(kvrqry)
     print("Student Table Dropped in Oracle Db--verify")
except cx_Oracle.DatabaseError as de:
     print(de)
finally:
     if con!=None:
          print("DB connection Closed:")
          con.close()



#recdelete1.py
#program for deleting record from table with static data
import cx_Oracle
try:
     con=cx_Oracle.connect("scott/tiger@localhost/orcl")
     print("Connection Obtained from Oracle DB")
     print("------------------------------------------------------
--")
     cur=con.cursor()
     kvrqry="delete from student where sno=30"
     cur.execute(kvrqry)
```

```
     con.commit()
     print("Student Record deleted from Student Table--verify")
except cx_Oracle.DatabaseError as de:
     print(de)
finally:
     if con!=None:
          print("DB connection Closed:")
          con.close()
```

```
#dynamicrecdelete1.py
#program for deleting record from table with dynamic  data
import cx_Oracle
try:
     sujcon=cx_Oracle.connect("scott/tiger@localhost/orcl")
     sujcur=sujcon.cursor()
     while(True):
          try:
               print("-----------------------------------------
------------")
               stno=int(input("Enter Student Number for deleting a
record:"))
               sujqry="delete from student where sno=%d "
               sujcur.execute(sujqry %stno)
               sujcon.commit()
               print("-----------------------------------------
------------")
               print("Student Record deleted:")
               print("-----------------------------------------
------------")
               ch=input("Do u want delete another record(yes/no):")
               if(ch=="no"):
                    break
          except ValueError:
```

```
                print("Don't enter strs/ special symbols/ alpha-
numeric values:")

except cx_Oracle.DatabaseError as de:
     print(de)
finally:
     if sujcon!=None:
          print("DB connection Closed:")
          sujcon.close()
```

```
#dynamicrecinsert1.py
#program for inserting record in table with dynamic  data
import cx_Oracle
try:
     con=cx_Oracle.connect("scott/tiger@localhost/orcl")
     cur=con.cursor()
     while(True):
          try:
                print("-----------------------------------------
-----")
                stno=int(input("Enter Student Number:"))
                stname=input("Enter Student Name:")
                marks=float(input("Enter Student Marks:"))
                colname=input("Enter College Name:")

                dqry="insert into student values(%d,'%s',%f, '%s') "
                cur.execute(dqry %(stno,stname,marks,colname) )
                con.commit()
                print("-----------------------------------------
-----")
                print("Student Record inserted Successfully...")
```

```
                print("-----------------------------------------------
-----")
                ch=input("Do U want to insert another Record (yes /
no):")
                if(ch=="no"):
                     break
           except ValueError:
                print("Don't enter strs / special symbols/
alphanumerics");

except cx_Oracle.DatabaseError as db:
     print("Data Base Problem: ",db)
```

```
#dynamicupdate.py
#program for updating a record in a  table with dynamic  data
import cx_Oracle
try:
     sujcon=cx_Oracle.connect("scott/tiger@localhost/orcl")
     sujcur=sujcon.cursor()
     while(True):
          try:
                print("-----------------------------------------------
------------")
                stno=int(input("Enter Student Number for updating a
record:"))
                stmarks=float(input("Enter Student Marks for
updating:"))
                sujqry="update student set marks=%f where sno=%d"
                sujcur.execute(sujqry %(stmarks, stno))
                sujcon.commit()
                print("-----------------------------------------------
```

```
------------")
                if(sujcur.rowcount>=1):
                     print("Student Record updated:")
                else:
                     print("Student Number and its record does not
exists:");
                print("-----------------------------------------
------------")
                ch=input("Do u want update another record(yes/no):")
                if(ch=="no"):
                     break
           except ValueError:
                print("Don't enter strs/ special symbols/ alpha-
numeric values:")

except cx_Oracle.DatabaseError as de:
     print(de)
finally:
     if sujcon!=None:
          print("DB connection Closed:")
          sujcon.close()



#recinsert1.py
#program for inserting record in table with static data
import cx_Oracle
try:
     con=cx_Oracle.connect("scott/tiger@localhost/orcl")
     print("Connection Obtained from Oracle DB")
     print("-----------------------------------------------------
--")
     cur=con.cursor()
     kvrqry="insert into student values(30,'MVR',39.25,'HCU')"
     cur.execute(kvrqry)
     con.commit()
     print("Student Record Inserted In Student Table--verify")
except cx_Oracle.DatabaseError as de:
     print(de)
finally:
     if con!=None:
          print("DB connection Closed:")
          con.close()
```

```
#selectex1.py
#program for reading the records from table
import cx_Oracle
try:
     con=cx_Oracle.connect("scott/tiger@localhost/orcl")
     cur=con.cursor()
     sq="select * from student"
     cur.execute(sq)
     print("-------------------------------------------------")
     print("Student Records")
     print("-------------------------------------------------")
     kvr=cur.fetchmany(2)
     for rec in kvr:
          for val in rec:
               print("{}".format(val), end=" ")
          print("\n")
     print("-------------------------------------------------")
except cx_Oracle.DatabaseError as db:
     print("Problem in Data base: ",db)
```

```
#testmy.py
import mysql.connector
con=mysql.connector.connect(host='localhost',

     user='root',

     passwd='root')
if con.is_connected():
     print("Python Program connected to My sql")
else:
     print("no")
```

==========
Random Module In Python
==========

=>"random" is one of the pre-defined module.
=>The purpose of "random" module is used for perfoming some random actions such as generating random number, selecting random element from any iterable object, suffle the elements randomly...etc.
=>Some of the essential functions present in random module are

1) random()
2) randint()
3) randrange()
4) choice()
5) shuffle()

1)random():

=> This function is used for generating any random number in the float between 0.0 to 1.0
Syntax: random.random()

Example:

```
#rex1.py
import random
print(random.random())
```

Example: print(random.random())
---
2) randint():

=>This function is used for generating random integer between the specified integers from start to stop.

Syntax:- random.randint(start,stop)

Example:

```
#rex2.py
import random
print(random.randint(1000,9999))
```
---
3) randrange():

=>this function returns a randomly selected element from range created by the start and stop. if we don't specify stop value then it starts from 0 to start-1

Syntax1:

```
     random.randrange(start)
```

=>This syntax generates random numbers from 0 to start-1
Example:

```
#rex3.py
import random
print(random.randrange(10))   # generates the random numbers from 0
```

```
to 10-1
-------------------------------------
Syntax2:-  random.randrange(start,stop)

=>This syntax generates random numbers from start to stop-1  values
---------------
Example:
---------------
#rex4.py
import random
print(random.randrange(10,20))   # generates the random numbers
from 10 to 20-1
-----------------------------------------------------------------
---------------------------------------
4) choice():
-----------------
=>This function is used for selecting an random element from
iterable object

Syntax:-
          random.choice(iterable object)

Example:-
          lst=[10,20,30,40,50]
          print(random.choice(lst))
Example:
----------------
import random
lst=[10,20,30,40,50]
print(random.choice(lst))
print("-------------------------------------")
print(random.choice("PYTHON"))
print("-------------------------------------")
tpl=(10,"KVR","OUCET",34.56)
print(random.choice( tpl))
=============================================================
5) shuffle():
------------------
=>this function is used for re-ordering the elements of any
iterable object
=>this function is applicabe for only mutable objects.
Syntax:-
          random.shuffle(iterable object)
```

==========

OOPS IN PYTHON

==========

=>In real time, to develop any project, we need a Language / Tech and it can be satisfied two principles. They are

1) Procedure Oriented Principles
2) Object Oriented Principles

=>Python is one of the Procedure Oriented and Object Programming Language. Even though python programming belongs to both, internally all the values are treated as "objects".

==========

" Benifits of Treating every thing is an Object in Python"

==========

=>The confidential data transfered in the form cipher text / encrypted format between client side application and server side application. So that Security is enhanced.

=>The Large Volume of data transfered between Client Side and Server Side application all at once (in the form object) and provides Effective Communication.

=>The data is available in the form of objects and we can apply functions / methods on the objects to perform operations.

==========

=>To Say a programming Language is object oriented , then it has to satisfy object oriented Priciples.

=>The Object Oriented Priciples in Python are:

1. Classes
2. Objects
3. Data Encapsulation
4. Data Abstraction
5. Inheritance
6. Polymorphism
7. Message Passing

==========

==========

Classes

==========

=>The purpose of Classes concept is that "To Develop Programmer-defined data Type + To Develop Real Time Applications ".
=>The pupose of Programmer-defined data Type is that to store multiple values either of same type or different type or the both types.
=>To develop programmer-defined data type with classes concept, we use a keyword called "class".
=>Each and Every Meanigful applications in Python must be developed by using classes concept.

=>Def. of Class:-

=>A class is a collection of Data Members(Instance Data memebrs and Class Level Data Members) and Methods(3).
=>Here "Data Members" of Class are used for Storing data in the objects(bcoz Data members of Class are available as it is as part of object)
=>"Methods" of class are used for performing Operations(If we write a Function inside class then those functions are called Methods).

Syntax for defining a class in python:

```
class   <clsname> :

     ClassLevel Data Members

     def    methodname(self, ........):
               ----------------------------------
            Block of statements--Object Operations
            ----------------------------------

     @classmethod
     def    methodname(cls, ........):
          ----------------------------------------
          Block of statements--Class Level Operations
          ----------------------------------------

     @staticmethod
     def    methodname(..............):
          ----------------------------------------------
```

Block of statements--Utility Operations

```
#studex1.py

class Student:pass  # here Student is called Class Name--
>Programmer-defined data type




#main program
so1=Student()
print("Id of so1=",id(so1))
print("content of so=",so1. __dict__) # { }
#add  Instance Data members to an object so1
so1.stno=100
so1.sname="KVR"
so1.marks=55.55
#display the data
print("{}\t{}\t{}".format(so1.stno,so1.sname,so1.marks))
print("------------------------------------------------------------
")
so2=Student()
print("Id of so2=",id(so2))
#add  Instance Data members to an object so1
so2.stno=101
so2.sname="Omprkash"
so2.marks=65.55
print("Id of so2=",id(so2))
print("{}\t{}\t{}".format(so2.stno,so2.sname,so2.marks))
```

```
#studex2.py

class Student:  # here Student is called Class Name-->Programmer-
defined data type
     crs="PYTHON"


#main program


so1=Student()
#add  Instance Data members to an object so1
so1.stno=100
so1.sname="KVR"
so1.marks=55.55
#display the data
print("{}\t{}\t{}\t{}".format(so1.stno,so1.sname,so1.marks,so1.crs
))
print("-----------------------------------------------------------
")
so2=Student()
#add  Instance Data members to an object so1
so2.stno=101
so2.sname="Omprkash"
so2.marks=65.55
print("{}\t{}\t{}\t{}".format(so2.stno,so2.sname,so2.marks,so2.crs)
)
```

==========
Types of Data Members in a Class of Python
==========

=>In python , we have 2 Types of Data Members in a Class. They are

1. Instance Data Members
2. Class Level Data Members

=>Differences between Instance Data Members and Class Level Data Members

1) Instance Data Members:

=>These data members are used for Storing specific / perticular values in the object.

=>The memory space is created for Instance Data Members every time, when an object is created and these data members are also called object level data members.

=>Instance Data Members must be accessed w.r.t object name (or) self

objectname.instance Data member

(OR)

self.instance Data member

=>One of the approach is that we can add / create Instance data members through an object name

=>another approach is that we can add / create Instance data members through an Instance method (with self)

Class Level Data Members

=>These data members are used for Storing common values .

=>The memory space is created for Class Level Data Members only

once irrespective of number of objects are created.
=>Class Level Data Members can be accessed w.r.t class name (or) object name (or) cls (or) self

```
classname. Class Level Data member
          (OR)
objectname.Class Level Data member
           (OR)
cls.Class Level Data member
           (OR)
self.Class Level Data member
```

=>Class Level Data Members can created in many ways.
a) Write within the class name
b) Out of Class definition w.r.t
Class name.class level Data member=val
c) through class level method

==========

==========
Types of Methods in a Class of Python
==========
=>In python programming, we have 3 types of methods. They are

1) Instance Methods
2) Class Level Methods
3) Static Methods

---
1) Instance Methods:

=>These methods are used for performing specific operations on objects.
=>These methods can do the operations only on objects and hence these methods are called Object Level Methods.
=>Instance methods Must written inside of calss definition by taking "self" as a first formal parameter.
=>Instance Methods must be accesses w.r.t object name (or) w.r.t self

```
objectname.Instancemethodname()
```

(or)

self.Instancemethodname()

=>Instance Methods can also call other type of methods.

==========

2) Class Level Methods

==========

=>These methods are used for performing Class Level / Common operations related the corresponding class

=>These methods can do the operations only on Class Level Data Members and hence these methods are called Class Level Methods.

=>The class level method must take "cls" as a First Formal Parameter and whose definition must be preceded with a pre-defined decorator called @classmethod

=>The class level method must be accessed w.r.t class name (OR) object name

classname.classlevel methodname()

(OR)

Object Name.classlevel methodname()

==========

3) Static Methods

==========

=>These methods are used for performing Utility operations / Independent Operations without considering the context corresponding class

=>These methods can do the operations on any class object and performs independent operations and they are called static methods.

=>The static methods never takes "cls" (or) "self" as a First Formal Parameter and whose definition must be preceded with a pre-defined decorator called @staticmethod

=>The static method must be accessed w.r.t class name (OR) object name

classname.static methodname()

(OR)

Object Name.static methodname()

==========

==========

Object

=>When we define a class, we don't get any memory for the data members(Instance Memebrs) and methods(Instance Methods) but whose memory space is created when we create an object.
=>To do any Data Processsing, We must create an object. In other words, without creating any object, we can't do any meaningful operations.
=>Programatically, to create an object , There must exists class definition otherwise we get Error.

Def. of Object:-- Instance of a class is called object (Instance is nothing but allocating memory space for Data members and Methods)

Creating an object:

=>To create an object, we use the following syntax

Syntax:

```
objectname=classname()
```

Example:- create an object of Student class

```
so=Student()
```

here so is called object name (or) reference vriable name

Example:- create an object of Employee class

```
eo=Employee()
```

here eo is called object name (or) reference vriable name

PROGRAM

```
#program for cal area and perimeter of Circle  using classes and objects
#Circledemo.py
```

```
class Circle:
     @classmethod
     def  setpi(cls):   # class Level method
          cls.pi=3.14    # class level data member

     def  circlearea(self):
          print("-------------------------------------------------
---------------")
          self.r=float(input("Enter Radious for cal Area of
Circle:"))
          self.ac=Circle.pi*self.r*self.r
          print("Area of Circle={}".format(self.ac))
          print("-------------------------------------------------
---------------")

     def  circleperi(self):
          print("-------------------------------------------------
---------------")
          self.r=float(input("Enter Radious for cal Peri of
Circle:"))
          self.pc=2*self.pi*self.r
          print("Area of Circle={}".format(self.pc))
          print("-------------------------------------------------
---------------")

#main program
Circle.setpi()
co=Circle()
co.circleperi()
co.circlearea()



#program for cal area and perimeter of Circle  using classes and
objects
#Circledemo1.py
class Circle:
     @classmethod
     def  setpi(cls):   # class Level method
          cls.pi=3.14    # class level data member
     def readradious(self, op):
          self.r=float(input("Enter Radious for {}:".format(op) ))
          return self.r
```

```
class Hyderabad:
     @staticmethod
     def  operations(c):
          rad=c.readradious("area")
          ac=3.14*rad*rad
          print("Area of Circle={}".format(ac))
          rad=c.readradious("peri")
          pc=2*3.14*rad
          print("\nperi of Circle={}".format(pc))

#main program
Circle.setpi()
co=Circle()
Hyderabad.operations(co)
```

```
class Employee:
     @classmethod
     def  appendcompname(cls):
          cls.compname="Wipro-HYD"

     def  appendempvalues(self):
          print("Id in method=", id(self))
```

```
            print("-------------------------------------------------
")
            self.empno=int(input("Enter Employee Number:"))
            self.ename=input("Enter Employee Name:")
            self.sal=float(input("Enter Employee Salary:"))
            self.dsg=input("Enter Employee Designation:")
            print("-------------------------------------------------
")

      def  dispempvalues(self):
            print("Id in method=", id(self))
            print("-------------------------------------------------
")
            print("Employee Number : {}".format(self.empno))
            print("Employee Name : {}".format(self.ename))
            print("Employee Designation : {}".format(self.dsg))
            print("Employee Salary : {}".format(self.sal))
            print("Employee Comp Name :
{}".format(Employee.compname))
            print("-------------------------------------------------
")

      @staticmethod
      def   operation(a,b,ops):
            if(ops=="+"):
                  print("{} {} {}={}".format(a,ops,b, a+b))
            elif(ops=="-"):
                  print("{} {} {}={}".format(a,ops,b, a-b))
            elif(ops=="*"):
                  print("{} {} {}={}".format(a,ops,b, a*b))
            elif(ops=="/"):
                  print("{} {} {}={}".format(a,ops,b, a/b))
            elif(ops=="//"):
                  print("{} {} {}={}".format(a,ops,b, a//b))
            elif(ops=="%"):
                  print("{} {} {}={}".format(a,ops,b, a%b))
            elif(ops=="**"):
                  print("{} {} {}={}".format(a,ops,b, a**b))
            else:
                  print("U Don't Arithmetic Operators--plz learn")

#main program
Employee.appendcompname()
```

```
eo1=Employee()
print("Id of eo1 in main program=",id(eo1))
eo1.appendempvalues()
eo2=Employee()
print("Id of eo2 in main program=",id(eo2))
eo2.appendempvalues()
print("==================================")
eo1.dispempvalues()
eo2.dispempvalues()
print("==========================================")
print("Utility Operation")
print("==========================================")
eo1.operation(10,3,"*")
eo2.operation(10,3,"**")


#studex3.py

class Student:pass  # here Student is called Class Name--
>Programmer-defined data type

#main program
Student.crs="DS-AI"
so1=Student()
#add  Instance Data members to an object so1
so1.stno=100
so1.sname="KVR"
so1.marks=55.55
#display the data
print("\n{}\t{}\t{}\t{}".format(so1.stno,so1.sname,so1.marks,so1.cr
s ))
print("------------------------------------------------------------
")
so2=Student()
#add  Instance Data members to an object so2
so2.stno=101
so2.sname="Omprkash"
so2.marks=65.55
print("{}\t{}\t{}\t{}".format(so2.stno,so2.sname,so2.marks,Student.
crs))
print("------------------------------------------------------------
")
so3=Student()
```

```
#add  Instance Data members to an object so3
so3.stno=102
so3.sname="dprasad"
so3.marks=66.55
print("{}\t{}\t{}\t{}".format(so3.stno,so3.sname,so3.marks,so3.crs)
)
```

==========

Introduction to Inheritance

==========

=>The main purpose of Inheritance principle is that to build Re-usable applications across the Class and Programs

Def.. of Inheritance:

=>The process of obtaining the data memebrs and methods(features) from one class into another class is called Inheritance

=>The Class which is giving data members and methods is called Base / super / parent class
=>The class which is taking data memebrs and Methods is called Derived / sub / child class

==========
=>Inheritance concept always follows Logical Memory Management.
=>This Memory Management Says "Neither we write Physical Source Code nor It takes Physical Memory Space"

==========
Advatanges of Inheritance:

If we develop any python application with the concept of inheritance, we get the following advantages.

1. Application Development time is Less
2. Application memory Space is Less
3. Application Execution time is Less
4. Application Performance is Enhanced (Improved)
5. Redundency (duplication / replication / repetation) of the Code is Minimized.

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

Inheriting the features of Base Class into Derived class

=>Features of a class are nothing but Data Members , Methods and Constructors.
=>To Inherit the features of Base class into derived Class, we use the following syntax.

```
class <clsname-1>:
      ------------------
      ------------------
class <clsname-2>:
      ------------------
      ------------------

class <clsname-n>:
      ------------------
      ------------------

class <clsname-n+1 > (clsname1,clsname-2......clsname-n):
          ----------------------------------
        ----------------------------------
```

Explnation:

=>here <clsname-1>,<clsname-2>....<clsname-n> reprsents Name of base class(es)
=>here <clsname-n+1 > reprsents Name of derived class.
=>In Python programing, one derived can inherit the features either from one base class or from more number of base classes.
=>When we develop any inheritance application, It is highly recommended to create an object Bottom derived class, bcoz it inherits the features of Base class and Intermediate Base classes .

=>For every class in python, there exists an implicit pre-defined super class called "object" bcoz it provides Garbage Collection facility and collected un-used memory space and improves the performance of Python Based Applications.

Program

```
#InhProg1.py
class Company:
     def   getcompdet(self):
          self.cname="Wipro"
          self.loc="HYD"

class Employee ( Company ) :
     def  getempdet(self):
          self.eno=100
          self.ename="Rosum"
          self.sal=23.45
          self.dsg="author"
     def dispempdet(self):
          print("-------------------------------------------------")
          print("Employee Details")
          print("-------------------------------------------------")
          print("Empoyee Number: {}".format(self.eno))
          print("Empoyee Name: {}".format(self.ename))
          print("Empoyee Salary: {}".format(self.sal))
          print("Empoyee Designation: {}".format(self.dsg))
          print("Company Name: {}".format(self.cname))
          print("Company Location: {}".format(self.loc))
          print("-------------------------------------------------")

#main program
eo=Employee()
eo.getempdet()
eo.getcompdet()
```

```
eo.dispempdet()
```

```
#InhProg2.py
class Company:
     def   getcompdet(self):
          self.cname="Wipro"
          self.loc="HYD"
class Food:
     def getfooddet(self):
          self.avfood="Biryani"
          self.drink="apple juice"

class Employee ( Company, Food ) :
     def  getempdet(self):
          self.eno=100
          self.ename="Rosum"
          self.sal=23.45
          self.dsg="author"
     def dispempdet(self):
          print("-------------------------------------------------")
          print("Employee Details")
          print("-------------------------------------------------")
          print("Empoyee Number: {}".format(self.eno))
          print("Empoyee Name: {}".format(self.ename))
          print("Empoyee Salary: {}".format(self.sal))
          print("Empoyee Designation: {}".format(self.dsg))
          print("Company Name: {}".format(self.cname))
          print("Company Location: {}".format(self.loc))
          print("-------------------------------------------------")
```

```
            print("Today Food details")
            print("-----------------------------------------------")
            print("Food in Company : {}".format(self.avfood))
            print("drink in Company : {}".format(self.drink))
            print("-----------------------------------------------")

#main program
eo=Employee()
eo.getempdet()
eo.getcompdet()
eo.getfooddet()
eo.dispempdet()
```

```
#InhProg3.py
class  Univ:
     def getunivdet(self):
          self.uname=input("Enter Univ Name:")
          self.uloc=input("Enter Univ Location:")
     def dispunivdet(self):
          print("-----------------------------------------------")
          print("University Details:")
          print("-----------------------------------------------")
          print("University Name:{}".format(self.uname))
          print("University Location:{}".format(self.uloc))
          print("-----------------------------------------------")

class College(Univ):
     def  getColldet(self):
          self.cname=input("Enter College Name:")
          self.cloc=input("Enter College Location:")
     def dispcolldet(self):
          print("College Details:")
          print("-----------------------------------------------")
          print("College Name:{}".format(self.cname))
          print("College Location:{}".format(self.cloc))
          print("-----------------------------------------------")

class Student(College):
```

```
     def  getstuddet(self):
          self.sno=int(input("Enter Student Number:"))
          self.sname=input("Enter Student Name:")
          self.crs=input("Enter Student Course:")

     def  dispstuddet(self):
          print("Studet Details:")
          print("-----------------------------------------------")
          print("Student Number:{}".format(self.sno))
          print("Student Name:{}".format(self.sname))
          print("Student Course:{}".format(self.crs))
          print("-----------------------------------------------")
#main program
so=Student()
so.getstuddet()
so.getColldet()
so.getunivdet()
so.dispunivdet()
so.dispcolldet()
so.dispstuddet()
#InhProg4.py
class  Univ:
     def getunivdet(self):
          self.uname=input("Enter Univ Name:")
          self.uloc=input("Enter Univ Location:")
     def dispunivdet(self):
          print("-----------------------------------------------")
          print("University Details:")
          print("-----------------------------------------------")
          print("University Name:{}".format(self.uname))
          print("University Location:{}".format(self.uloc))
          print("-----------------------------------------------")

class College:
     def  getColldet(self):
          self.cname=input("Enter College Name:")
          self.cloc=input("Enter College Location:")
     def dispcolldet(self):
          print("College Details:")
          print("-----------------------------------------------")
          print("College Name:{}".format(self.cname))
          print("College Location:{}".format(self.cloc))
          print("-----------------------------------------------")
```

```
class Student(College,Univ):
     def  getstuddet(self):
          self.sno=int(input("Enter Student Number:"))
          self.sname=input("Enter Student Name:")
          self.crs=input("Enter Student Course:")

     def  dispstuddet(self):
          print("Studet Details:")
          print("-----------------------------------------------")
          print("Student Number:{}".format(self.sno))
          print("Student Name:{}".format(self.sname))
          print("Student Course:{}".format(self.crs))
          print("-----------------------------------------------")

so=Student()
so.getstuddet()
so.getColldet()
so.getunivdet()
so.dispunivdet()
so.dispcolldet()
so.dispstuddet()
#InhProg5.py
class  Univ:
     def getunivdet(self):
          self.uname=input("Enter Univ Name:")
          self.uloc=input("Enter Univ Location:")
     def dispunivdet(self):
          print("-----------------------------------------------")
          print("University Details:")
          print("-----------------------------------------------")
          print("University Name:{}".format(self.uname))
          print("University Location:{}".format(self.uloc))
          print("-----------------------------------------------")

class College(Univ):
     def  getColldet(self):
          self.cname=input("Enter College Name:")
          self.cloc=input("Enter College Location:")
          so.getunivdet();
     def dispcolldet(self):
          so.dispunivdet()
          print("College Details:")
```

```
          print("-----------------------------------------------")
          print("College Name:{}".format(self.cname))
          print("College Location:{}".format(self.cloc))
          print("-----------------------------------------------")

class Student(College):
     def  getstuddet(self):
          self.sno=int(input("Enter Student Number:"))
          self.sname=input("Enter Student Name:")
          self.crs=input("Enter Student Course:")
          so.getColldet()

     def  dispstuddet(self):
          so.dispcolldet()
          print("Studet Details:")
          print("-----------------------------------------------")
          print("Student Number:{}".format(self.sno))
          print("Student Name:{}".format(self.sname))
          print("Student Course:{}".format(self.crs))
          print("-----------------------------------------------")

so=Student()
so.getstuddet()
so.dispstuddet()

#InhProg6.py
class  Univ:
     def getunivdet(self):
          self.uname=input("Enter Univ Name:")
          self.uloc=input("Enter Univ Location:")
     def dispunivdet(self):
          print("-----------------------------------------------")
          print("University Details:")
          print("-----------------------------------------------")
          print("University Name:{}".format(self.uname))
          print("University Location:{}".format(self.uloc))
          print("-----------------------------------------------")

class College(Univ):
     def  getColldet(self):
          self.cname=input("Enter College Name:")
          self.cloc=input("Enter College Location:")
          self.getunivdet();
```

```
     def dispcolldet(self):
          self.dispunivdet()
          print("College Details:")
          print("-----------------------------------------------")
          print("College Name:{}".format(self.cname))
          print("College Location:{}".format(self.cloc))
          print("-----------------------------------------------")

class Student(College):
     def  getstuddet(self):
          self.sno=int(input("Enter Student Number:"))
          self.sname=input("Enter Student Name:")
          self.crs=input("Enter Student Course:")
          self.getColldet()

     def  dispstuddet(self):
          self.dispcolldet()
          print("Studet Details:")
          print("-----------------------------------------------")
          print("Student Number:{}".format(self.sno))
          print("Student Name:{}".format(self.sname))
          print("Student Course:{}".format(self.crs))
          print("-----------------------------------------------")
so=Student()
so.getstuddet()
so.dispstuddet()
#university.py----->treated as module
class  Univ:
     def getunivdet(self):
          self.uname=input("Enter Univ Name:")
          self.uloc=input("Enter Univ Location:")
     def dispunivdet(self):
          print("-----------------------------------------------")
          print("University Details:")
          print("-----------------------------------------------")
          print("University Name:{}".format(self.uname))
          print("University Location:{}".format(self.uloc))
          print("-----------------------------------------------")


#student.py---treated as module
from college import College
class Student(College):
```

```
    def  getstuddet(self):
         self.sno=int(input("Enter Student Number:"))
         self.sname=input("Enter Student Name:")
         self.crs=input("Enter Student Course:")
         self.getcolldet()
         self.getunivdet()
    def  dispstuddet(self):
         self.dispunivdet()
         self.dispcolldet()
         print("Studet Details:")
         print("-----------------------------------------------")
         print("Student Number:{}".format(self.sno))
         print("Student Name:{}".format(self.sname))
         print("Student Course:{}".format(self.crs))
         print("-----------------------------------------------")


#studcolluniv.py
from student import Student
so=Student()
so.getstuddet()
so.dispstuddet()
```

==========
Polymorphism
==========

=>Polymorphism is one of the distinct principle of OOPs
=>The process of representing "one form in multiple multiple forms" is called Polymorphism

=>One form reprsents Original Method of base class
=>Multiple Forms reprsents overridden methods derived classes
=>form represents "State of existence of the method". if The method exists in base class then it is called Original Method(one form) and if the method exists in derived class then it is called Overridden method (multiple forms)
=>The advantage Polymorphism principle is that minimized amount of memory space is provided to the application.

=>Polymorphism is one best principle but it is not programming concept.
=>All OOP lang implementing the polymorphism principle by using their concepts.
=>In Python Programming, polymorphism principle is implemented by Method Overriding

=>Method Overridding = Method Heading is same + Method body is different

==========

super()

=>super() is used for calling Base class Original methods from the context of overridden method of derived class
=>super() is also used for calling Base class Constructor from the context of overridden Constructor of derived class.

Syntax1:(At method level)

```
super().method name()
        (or)
super().method name(list of values)
```

Syntax2(At constructor Level)

```
super(). __init__()
 (OR)
super(). __init__(list of values)
```

These syntaxex applicable for Types of all inheritances but not with multiple inheritance (or) Hybdrid Inheritance (diamond problem)
=>To over this problem, we use the following syntaxes.

Syntax3:(At method level)

a) classname.methodname(self)
(OR)
classname.methodname(self, list of values)

Syntax: 3 (At constructor Level)

a) classname.__init__(self)
(OR)
classname.__init__(self, list of values)

==========

Programs

```
#polyex1.py
class Circle(object):
     def   draw(self):#original method
          print("Drawing Circle--Circle class:")

class Rect(Circle):
     def  draw(self): # overridden method
          super().draw() # will call draw() of Circle class
          print("Drawing Rect--Rect class:")

class Square(Rect):
     def  draw(self): # overridden method
          super().draw() # will call draw() of Rect class
          print("Drawing Square--Squar class:")



#main program
so=Square()
so.draw();




#polyex2.py
class Circle(object):
     def   __init__(self):#original constructor
          print("Drawing Circle--Circle class:")

class Rect(Circle):
```

```
     def  __init__(self): # overridden constructor
          super().__init__() #calling    __int__() of Circle class
          print("Drawing Rect--__init__(self):")

class Square(Rect):
     def  __init__(self):  # overridden constructor
          super().super().__init__() #calling    __int__() of Rect
class
          print("Drawing Square-- __init__(self):")



#main program
so=Square()


#polyex3.py
class Circle(object):
     def   draw(self):#original method
          print("Drawing Circle--Circle class:")

class Rect:
     def  draw(self):
          print("Drawing Rect--Rect class:")

class Triangle:
     def  draw(self):
          print("Drawing Triangle--Triangle class:")

class Square(Triangle, Rect, Circle ):
     def  draw(self): # overridden method
          print("Drawing Square--Squar class:")
          Circle.draw(self)
          Triangle.draw(self)
          Rect.draw(self)

#main program
so=Square()
so.draw();

#polyex4.py
class Circle(object):
     def   __init__(self):#original constructor
```

```
          print("Drawing Circle--Circle class:")

class Rect:
     def  __init__(self):
          print("Drawing Rect--__init__(self):")
class Triangle:
     def  __init__(self):
          print("Drawing Triangle--__init__(self):")

class Square(Rect,Circle,Triangle):
     def  __init__(self):  # overridden constructor
          print("Drawing Square-- __init__(self):")
          Circle.__init__(self)
          Rect.__init__(self)
          Triangle.__init__(self)

#main program
so=Square()

#polyex5.py
class Circle:
     def  area(self,r):# original method
          ac=3.14*r*r
          print("Area of Circle=",ac)

class Rect (Circle):
     def area(self,l,b):# overridden method
          ar=l*b
          print("Area of Rect=",ar)

class Square(Rect):
     def area(self,s):  # overridden method
          as1=s*s
          print("Area of Square=",as1)
          print("----------------------------------------")
          super().area(float(input("Enter Length of
Rect:")),float(input("Enter breadth of Rect:")))
          print("----------------------------------------")
          Circle.area(self,float(input("Enter Radious of
Circle:")))
# main program
so=Square()
so.area(float(input("Enter Side of Square:")))
```

```
#polyex6.py
class Circle:
     def  area(self,r):# original method
          print("----------------------------------------")
          ac=3.14*r*r
          print("Area of Circle=",ac)

class Square:
     def area(self,s):  # overridden method
          print("----------------------------------------")
          as1=s*s
          print("Area of Square=",as1)

class Rect :
     def area(self,l,b=12):# overridden method
          print("----------------------------------------")
          ar=l*b
          print("Area of Rect=",ar)

#main program
co=Circle()
so=Square()
ro=Rect()
lst=[]
lst.append(co)
lst.append(so)
lst.append(ro)
for obj in lst:
     obj.area(3)
print("----------------------------------------")
for obj in (co,ro,so):
     obj.area(6)
```

```
#polyex7.py
class India:
     def capital(self):   # original method
          print("Delhi is the capital of India")

     def lang(self):    # original method
          print("Hindi / Mixed Lang Indian can speack")

     def  type(self):   # original method
          print("India is a developping.... Country")
class USA(India):
     def capital(self):   # overridden method
          print("WashingTon is the capital of USA")
          India.capital(self)
          print("-------------------------------------------------
--------")
     def lang(self):    # overridden method
          print("English Lang, Americans can speack")
          super().lang()
          print("-------------------------------------------------
--------")
     def  type(self):  # overridden method
          print("USA is a developped Country")
          super().type()
          print("-------------------------------------------------
--------")
#main program
uo=USA()
uo.capital()
uo.lang()
uo.type()
```

```
#polyex8.py
class India:
     def capital(self):   # original method
          print("Delhi is the capital of India")

     def lang(self):    # original method
          print("Hindi / Mixed Lang Indian can speack")

     def  type(self):   # original method
          print("India is a developping.... Country")
class USA:
     def capital(self):
          print("WashingTon is the capital of USA")
     def lang(self):
          print("English Lang, Americans can speack")
     def  type(self):
          print("USA is a developped Country")
#main program
uo=USA()
ind=India()
print("-----------------------------------------------------")
for obj in [uo,ind]:   # here we are using object level
polymorphism
     print("Ref of obj=",type(obj))
     obj.capital()
     obj.lang()
     obj.type()
     print("----------------------------------------")
```

```
#polyex9.py
class India:
     def capital(self):   # original method
          print("Delhi is the capital of India")
     def lang(self):    # original method
          print("Hindi / Mixed Lang Indian can speack")
     def  type(self):   # original method
          print("India is a developping.... Country")
class USA:
     def capital(self):
          print("WashingTon is the capital of USA")
     def lang(self):
          print("English Lang, Americans can speack")
     def  type(self):
          print("USA is a developped Country")

class Country:
     @staticmethod
     def dispCountryInfo(obj):
          print("-------------------------------------------------
--")
          obj.capital()
          obj.lang()
          obj.type()
          print("-------------------------------------------------
--")

#main program
uo=USA()
ind=India()
Country.dispCountryInfo(uo)
Country.dispCountryInfo(ind)
```

```
#polyex10.py
class India:
     def capital(self):   # original method
          print("Delhi is the capital of India")
     def lang(self):    # original method
          print("Hindi / Mixed Lang Indian can speack")
     def  type(self):   # original method
          print("India is a developping.... Country")
class USA:
     def capital(self):
          print("WashingTon is the capital of USA")
     def lang(self):
          print("English Lang, Americans can speack")
     def  type(self):
          print("USA is a developped Country")

class Country:
     @staticmethod
     def dispCountryInfo(obj):
          print("-------------------------------------------------
--")
          obj.capital()
          obj.lang()
          obj.type()
          print("-------------------------------------------------
--")

#main program
uo=USA()
ind=India()
for kvobj in (ind,uo):
     Country.dispCountryInfo(kvobj)
```

Data Encapsualtion and Data Abstraction

Data Encapsualtion:

=>The process hiding the confidential information(Data members / Methods) from external Programmers / Users is called Data Encapsualtion

=>Data Encapsulation can be implemented (like private in Java) in python for hinding the data members and methods.

=>the members can be hidden from external programmers by following the syntax

```
__ data member name   (like a private data member in java)

def __methodname(self,.....):
```

=>The __data member name (private data member) and __method name(..) (private method) can be accessed in the context of same class but not possible to access in the context of other classes.

Example: account.py

Data Abstraction:

=>The process of retrieving essential details without considering about hidden details is called Data Abstraction.

Example: others.py

```
#account.py----treated as module
class Account:
     def  __init__(self):
          self.__acno=10
          self.cname="Avinash"
          self.__bal=3.4
          self.__pin=1234
          self.bname="SBI"
     def __openPinCover(self):  # private method
          print("Ur pin is={}".format(self.__pin))


#others.py
from account import Account

ao=Account()
print("-----------------------------------------------------------
-----")
#print("Account Number=",ao.acno) can't access
print("Account Holder Name=",ao.cname)
#print("Account Balance=",ao.bal)  can't access
#print("Account Pin=",ao.pin)  can't access
print("Account Branch Name=",ao.bname)
print("-----------------------------------------------------------
-----")
#ao.openPinCover()    can't access
```

OOPs--->Constructors in Python

=>The purpose of Constructors in a class of Python is that "To Initlize the Object " (OR) Constructors are always used for placing our values in the object without empty

=>Def of Constructor:

=>A Constructor is one of the special method , which is automatically / implicitly called by PVM during object creation whose role is to place our own values in the object without empty.

Syntax for defining Constructor:

```
     def    __init__(self, list of formal params if any):
                -----------------------------------------------
            ---------------------------------------------
            Block of statemnts--Initlization
```


Rules for using Constructors:
---
1. A Constructor is automatically / implicitly called by PVM during object creation whose role is to place our own values in the object without empty.
2. Constructor Name must starts with _ _ init_ _(self, params if any)
3. A Constructor type must be always instance nature and it can also be class level method nature.

=>Differences between Constructors and Methods:

=>Constructors can initlize the objects and Methods can do the operations on object data.
=>Constructor name must be __init__(self,.....) where as method name can be programmer-defined.
=>Constructors called by PVM implicitly when an object is created where Methods must be explicitly when we need.

Types of Constructors in python:

=>In python, we have two types of Constructors. They are
  a) Default / Parameter Less Constructor
  b) Parameterized Constructor
---
a) Default / Parameter Less Constructor:

=>Syntax:- def __init__(self):
                  ----
                block of statements--Initlization

=>Def Default Constructor:-

=>A Default Constructor is one, which never takes any parameters. / values (except self)

=>The pupose of Default Constructor is that to initlize multiple

objects of same class with same values .

---

b) Parameterized Constructor:

=>Syntax:- def __init__(self, list of formal params):

                    ----

                block of statements--Initlization

=>Def Parameterized Constructor:-

=>A Parameterized Constructor is one, which always takes parameters. / values (along self)

=>The pupose of Parameterized Constructor is that to initlize multiple objects of same class with different values .

==========

=

Note:-

=>In perticular class of Python, we can't define a separate default and separate Parameterized constructor . But we can define only single parameterized constructor, which will satisfy default constructor functionality also by the help of default and key word arguments.

```
#defaulconsex1.py
class Test:
     def __init__(self):
          self.a=10
          self.b=20

t1=Test()
print("Content of t1=", t1.__dict__)
t2=Test()
print("Content of t2=", t2.__dict__)
t3=Test()
print("Content of t3=", t3.__dict__)
```

```
#emp.py
class Employee:
     def   getempvalues(self):
          self.eno=int(input("Enter Emp Number:"))
          self.ename=input("Enter Emp Name:")
          self.dsg=input("Enter Emp Designation:")

     def dispempvalues(self):
          print("Emp Number: {}".format(self.eno))
          print("Emp Name: {}".format(self.ename))
          print("Emp Designation: {}".format(self.dsg))


#main program
eo1=Employee()  # when we  create an object, it must initlized by
calling one special method implicitly by PVM--that special method
is called Constructor
print("content of eo1=", eo1.__dict__)

eo1.getempvalues()  # calling explicitly we are calling method
```

```
#emp1.py
class Employee:
     @classmethod
     def  __init__(self, a,b,c):
          print("--------------------------------------")
          self.eno=int(input("Enter Emp Number:"))
          self.ename=input("Enter Emp Name:")
          self.dsg=input("Enter Emp Designation:")
          print("--------------------------------------")
```

```
     def dispempvalues(self):
          print("-------------------------------------------------
-")
          print("Emp Number: {}".format(self.eno))
          print("Emp Name: {}".format(self.ename))
          print("Emp Designation: {}".format(self.dsg))
          print("-------------------------------------------------
-")
#main program
eo1=Employee()
eo2=Employee()
     eo3=Employee()
```

```
#paramconsex1.py
class Test:
     def __init__(self, a,b):
          self.a=a
          self.b=b

t1=Test(10,20)
print("Content of t1=", t1.__dict__)
t2=Test(100,200)
print("Content of t2=", t2.__dict__)
t3=Test(1000,2000)
print("Content of t3=", t3.__dict__)
```

```
#sample.py
class Sample:
     def __init__(self, a=100, b=200):
          print("-----------------------------------------")
          self.a=a
          self.b=b
          print("Val of a :{}".format(self.a))
```

```
            print("Val of b :{}".format(self.b))
            print("-----------------------------------------")
#main program
print("so1 values")
so1=Sample()   # default constructor
print("so2 values")
so2=Sample(1,2) # Parameterized  constructor
print("so3 values")
so3=Sample(10) # Parameterized  constructor
print("so4 values")
so4=Sample(b=55,a=65) # Parameterized  constructor
print("so5 values")
so5=Sample(b=555) # Parameterized  constructor
```

==========

Garbage Collection--gc module

and

Destructors in Python

==========

=>A destructor is a special method identified as __del__(self), which automatically / implicitly called by Garbage Collector .
=>Garbage Collector is in-built program, which is automatically calling destructor provided when the referenced object becomes un-referenced bcoz of two operations.

1) del objname
or
2) objname=None

=>Garbage Collector is in-built program, which is automatically calling destructor provided when PVM reaches end of the application. so that all memory space of objects of correspoding program will collected by GC

=>Syntax for Destructor:

```
def __del__(self):
```

PROGRAMS

```
#destex1.py
import time
class Sample:
     def __init__(self):
          print("Object Initliztion---Constructor:")

     def __del__(self):
          print("Memory Space pointed by object removed--Garbage Collector")


#main program
s1=Sample()
print("object created and initlized");
time.sleep(10)
s1=None
print("End of application")
#destex2.py
import time
```

```
class Sample:
     def __init__(self):
          print("Object Initliztion---Constructor:")

     def __del__(self):
          print("Memory Space pointed by object removed--Garbage
Collector")


#main program
s1=Sample()
print("object created and initlized");
time.sleep(10)
print("End of application")


#destex3.py
import time
class Sample:
     def __init__(self):
          print("Object Initliztion---Constructor:")

     def __del__(self):
          print("Memory Space pointed by object removed--Garbage
Collector")


#main program
s1=Sample()
print("object created and initlized");
s2=s1
s3=s2
print("Object s3 is removed")
s3=None
time.sleep(10)
print("Object s2  is removed")
s2=None
time.sleep(10)
print("Object s1   memory space is  going be removed")
s1=None
time.sleep(10)
print("End of Application")
```

```
#destex4.py
import time
class Sample:
     def __init__(self):
          print("Object Initliztion---Constructor:")

     def __del__(self):
          print("Memory Space pointed by object removed--Garbage
Collector")


#main program
s1=Sample()
print("object created and initlized");
s2=s1
s3=s2
print("Object s3 is removed")
time.sleep(10)
print("Object s2  is removed")
time.sleep(10)
print("Object s1   memory space is  going be removed")
time.sleep(10)
print("End of Application")


#destex5.py
import time
class Sample:
     def __init__(self):
          print("Object Initliztion---Constructor:")

     def __del__(self):
          print("Memory Space pointed by object removed--Garbage
Collector")


#main program
lst=[ Sample(),Sample(),Sample() ]
print("we created 3 sample objects and placed in lst")
time.sleep(10)
del lst
time.sleep(10)
print("end of application")
```

```
#destex6.py
import time
class Sample:
     def __init__(self):
          print("Object Initliztion---Constructor:")

     def __del__(self):
          print("Memory Space pointed by object removed--Garbage
Collector")


#main program
s1=Sample()
s2=Sample()
s3=Sample()
print("we created 3 sample objects and placed in lst")
time.sleep(10)
print("object s1--memory space")
del s1    # gc is calling __del__(self)
time.sleep(10)
print("object s2--memory space")
del s2   # gc is calling __del__(self)
time.sleep(10)
print("object s3--memory space")
del s3   # gc is calling __del__(self)
time.sleep(10)
print("end of application")


#destex7.py
import time,sys
class Sample:
     def __init__(self):
          print("Object Initliztion---Constructor:")

     def __del__(self):
          print("Memory Space pointed by object removed--Garbage
Collector")

#main program
s1=Sample()
```

```
s2=s1
s3=s2
print("Refs of s1=", sys.getrefcount(s1)) #4
print("end of application")
```

==========

Garbage Collection--gc module

==========

=>We know that garbage Collector is python in-built program, which is running by default in background our python application for collecting / removing Un-Used memory space and improves the performnace of Python Based applications.

=>To deal with garbage Collection Facility , we use a pre-defned module called "gc"

=>The module gc contains the following functions.

a) gc.disable()------>used for stopping the exection of garbage collector

b) gc.enable()------>used to start the exection of garbage collector

c) gc.isenabled()--> It returns True provided garbage collector program is running

It returns False provided garbage collector program is not running

=>Garbage Collector is internally calling destructor for de-allocating the memory space which is allocated for an object

---

Example:

#gcdemo.py

```
import gc
print("By default Garbage Collector Program is running ..")
#gc module-----  isenabled()--->True(running)      Flase (not running)
print("is Garbage Colletcor Running by default=", gc.isenabled()) # True
#gc module-----  disable()--->Garbage Collector program stops running
gc.disable()
print("is Garbage Colletcor Running afer disable()=", gc.isenabled()) # False
```

```
#gc module-----  enable()--->Garbage Collector program starts
running
gc.enable()
print("is Garbage Colletcor Running afer enable()=",
gc.isenabled()) # True
-------------------------------------------------------------------
----------------------------------
```

```
#gcdemo.py
import gc
print("By default Garbage Collector Program is running ..")
#gc module-----  isenabled()--->True(running)      Flase (not
running)
print("is Garbage Colletcor Running by default=", gc.isenabled()) #
True
#gc module-----  disable()--->Garbage Collector program stops
running
gc.disable()
print("is Garbage Colletcor Running afer disable()=",
gc.isenabled()) # False
#gc module-----  enable()--->Garbage Collector program starts
running
gc.enable()
print("is Garbage Colletcor Running afer enable()=",
gc.isenabled()) # True
```

==========
Network Programming In Python
==========

=>The purpose of Network Programming is that "To share the data / info across the network".

=>Def of network:

=>"Collection of interconnected computers connected each other"

=>We are we dealing network programming, we must ensure that Physical network must established.

=>In Network programming any language , we develop, two types of applications / programs. They are

1) Server Side Program
2) Client Side Program

1) Server Side Program:

=>This program Accepts Client Request, Process the client request by reading Requested data and Gives Response back client Side Program.

Steps for developing Server Side Application

-
1) write the server side program
2) Every server side program be available in a machine(DNS--
>localhost or IP Adress->127.0.0.1) and it
must run at unique Port number

3) The Server Side Program must ACCEPT Client Request
4) The Server Side Program Must Read Client Requested Data and
Process
5) The Server Side Program must send Response back to client Side
Program

Note:- As long as client Makes request(2) , Server side Program
performs (3),(4) and (5) steps.

---
2) Client Side Program

=>This Program makes a request to server side program and the
response

Steps for developing Client Side Application

---
1) Write the Client side program

2) Client Side Program must obtain connection from Server Side
Program(DNS / IPAddress , portno)

3) The Client Side Program must send a request to the server side.

4) The Client Side Program must receive the response from Server
Side Program and display.

5) Client Side Program closes its connection

=>In Python, to develop any networking application / Client-Server
Application, we must use a pre-defined module called "socket"
---

socket module:
---
1) socket():

=>This function is used creating object of socket, which is bi-directional communication object.

Syntax:-

varname=socket.socket()

here varname is an object of socket

Example:-

s=socket.socket()

---
2) bind(dns,port)

=>This function is used for making server side program to run at certain DNS and Port number. The params dns and port must be always passed in the tuple

host=socketobj.gethostname()

Syntax:- socketobj.bind(("localhost",8888))

(OR)

socketobj.bind(("127.0.0.1",8888))

(OR)

socketobj.bind((host,8888))

(OR)

socketobj.bind((socket.gethostname(),8888))

---
3) connect(dns,port):

=>This function is used for obtaning connection from server side program by passing dns and port in the tuple by client side program.

Syntax:- socketobj.connect((host,portno))

Example: socketobj.connect(("localhost",8888))

(or)

socketobj.connect((socket.gethotsname() ,8888))

---
4) listen(no. of client request):

=>This function is used for configuring the server side program, how many clients can communicate with Server Side program simultaneously.
Syntax:- socketobj.listen(2)
---
5) accept( ) --->[we get addr of client, connection]

=>This function accepts the request from client side program and returns in the form of connection and address of client

Syntax: varname1,varname2=socket.accept()
varname1 represents connection object
varname2 represents addr of client at server side
Example: kvrcon, kvradd=socketobj.accept()
---
6) send():-

=>This function is used for sending the data from client side to server side program and from server side program to client side program with encode()

syntax: socketobj.send(strdata.encode())

Example:-socketobj.send("Hello Rossum".encode())
---
7) recv()---> [1024 2048 4096......] :

=>This function is used for recevinging the data from client side to server side program and from server side program to client side program with decode() by specifing size of data by mensioning either 1024 or 2048 or 4096...etc

syntax: varname=socketobj.recv(1024).decode()
varname is an object, which is holding server data or

client

Program

```
#client1.py
import socket
s=socket.socket()
s.connect(("localhost",8989) )
print("-----------------------------------------------------")
```

```
print("Client Side Program connected to server side program");
print("-------------------------------------------------")
cdata=input("Enter a Message:")
s.send(cdata.encode())
print("-------------------------------------------------")
sdata=s.recv(1024).decode()
print("Data from Server at Client side={}".format(sdata))
print("-------------------------------------------------")
```

```
#server1.py
import socket
s=socket.socket()
s.bind( ("localhost",8989) )
s.listen(4)
print("-------------------------------------------------------------
---")
print("Server Side Program is ready to accept client Request...");
print("-------------------------------------------------------------
---")
while(True):
     con,addr=s.accept()
     print("Type of con=",type(con))
     print("Type of addr=",type(addr))
     print("Server program connected to client at:
{}".format(addr))
     cdata=con.recv(1024).decode()
     print("Data from Client at Server side={}".format(cdata))
     con.send("Hi, This is KVR from server Side".encode())
```

```
#client2.py----> This program accept a number from kbd at client,
send to server and get its  square.
import socket
s=socket.socket()
s.connect(("localhost",9999) )
print("-----------------------------------------------------")
print("Client Side Program connected to server side program");
print("-----------------------------------------------------")
val=input("\nEnter a number:")
s.send(val.encode())
sdata=s.recv(1024).decode()
print("-----------------------------------------------------")
print("Square of {} at Client Side:{}".format(val,sdata))
print("-----------------------------------------------------")
```

```
#server2.py---------> Server program accept the number from client
program and square it send the result to client program
import socket
s=socket.socket()
s.bind( ("localhost",9999) )
s.listen(4)
print("-------------------------------------------------------------
---")
print("Server Side Program is ready to accept client Request...");
print("-------------------------------------------------------------
---")
while(True):
     conn,addr=s.accept()
     cval=conn.recv(1024).decode()
     print("Value from client at server side: {}".format(cval))
     n=int(cval)
     res=n*n
     conn.send(str(res).encode())
```

History of NUMPY:

=>NUMPY stands for NUMerical PYthon
=>Before NUMPY , there was a library called "Numeric Library"
=>"Numeric Library" developed by JIM HUNGUNIAN
=>NUMPY developed by TRAVIS OLIPHANT in the year 2005 with help of other contributors
=>NUMPY module developed in the languages C,PYTHON

=>To use NUMPY as part of our python program, we must install numpy module by using pip tool.

Syntax:- pip install numpy
==========
Uses of numpy module:

=> The numpy module is used to organize the elements in the form of arrays easily.
=> The numpy module computes complex mathematical operations easily.
=>The numpy module can be used in AI, DS , ML , DL in data analysis.
==========

==========
Storing the data in numpy programming
==========
=>In numpy programming, to store the data, we must create an object of ndarray class.

Creating an object of ndarray:

=>we have the following essential approaches to create an object of ndarray.
a) array()

```
          b) arange()
          c) linspace()
          d) zeros()
          e) ones()
          f) full()
          g) eye()
          h) identity()
-----------------------------------------------

a)array():
------------
=>It is used for creating an object of ndarray by passsing an
object which is containing multiple values (OR) any function
returns an of array of elements.
Syntax:-
-----------
          varname=np.array(object, dtype)

=>here varname is an object of <class,'ndarray'>
=>array() converts any type object values into an object of type
ndarray

>>>a=np.array(10)
>>> a
array(10)
>>> type(a)-----------<class 'numpy.ndarray'>
>>> a.ndim---0
-------------------------------------
>>> l1=[10,20,30,40]
>>> a=np.array(l1)
>>> a-----------array([10, 20, 30, 40])
>>> a.ndim--------------1
-------------------------------------------------------------
>>> l1=[[10,20,30],[30,40,50],[60,70,80]]
>>> a=np.array(l1)
>>> a
array([[10, 20, 30],
       [30, 40, 50],
       [60, 70, 80]])
>>> a.ndim--------------2
>>> print(type(a))-----------<class 'numpy.ndarray'>
-------------------------------------------------------------
--------
```

```
>>> def  fun():
...    l1=[[10,20],[30,40]]
...    return l1
...
>>> a=np.array(fun())
>>> type(a)
<class 'numpy.ndarray'>
>>> a
array([[10, 20],
       [30, 40]])
>>> a.ndim---------2
============================================================
special examples:
---------------------------
>>> a=np.array([10,20,30.2])
>>> a------------>array([10. , 20. , 30.2])
>>> a.dtype---------->dtype('float64')

>>> a=np.array([10,20,40,True])
>>> a.dtype----------dtype('int32')
>>> a------------array([10, 20, 40,  1])
>>> a=np.array([10,20,40,True,2.3])
>>> a----------array([10. , 20. , 40. ,  1. ,  2.3])
>>> a.dtype------------dtype('float64')

>>> a=np.array([10,20,40,True,2.3,0], dtype='bool')
>>> a-----------array([ True,  True,  True,  True,  True, False])
>>> a.dtype-----------dtype('bool')

------------------------------------------------------------------------
----------------------
>> a=np.array([10,20,40,True])
>>> a.dtype
dtype('int32')
>>> a------------array([10, 20, 40,  1])
>>> a=np.array([10,20,40,True,2.3])
>>> a-----------------array([10. , 20. , 40. ,  1. ,  2.3])
>>> a.dtype---------------dtype('float64')
>>> a=np.array([10,20,40,True,2.3,0], dtype='bool')
>>> a-----------array([ True,  True,  True,  True,  True, False])
>>> a.dtype-----------dtype('bool')
>>> a=np.array([10,20,30,'kvr'])
>>> a---array(['10', '20', '30', 'kvr'], dtype='<U11')
```

```
>>> a=np.array(['kvr','pyt','jav'])
>>> a----array(['kvr', 'pyt', 'jav'], dtype='<U3')
>>> a=np.array(['kr','pt','jav'])
>>> a-------array(['kr', 'pt', 'jav'], dtype='<U3')
>>> a=np.array(['kr','pt','jv'])
>>> a------------array(['kr', 'pt', 'jv'], dtype='<U2')
>>> a=np.array([10,20,2+3j])
>>> a------array([10.+0.j, 20.+0.j,  2.+3.j])
>>> a=np.array([10,20,2+3j], dtype='int')-->TypeError: can't
convert complex to int
>>> a=np.array([10,20,2+3j], dtype='complex')
>>> a-----------array([10.+0.j, 20.+0.j,  2.+3.j])
>>> a=np.array([10,20,2+3j], dtype='str')
>>> a------------array(['10', '20', '(2+3j)'], dtype='<U6')
>>> a.dtype----------dtype('<U6')
>>> a=np.array([10,20,2+3j], dtype='float')----TypeError: can't
convert complex to float
>>> a=np.array([10,'KVR',12.34,True,2+4j])
>>> a-----array(['10', 'KVR', '12.34', 'True', '(2+4j)'],
dtype='<U64')
>>> a=np.array([10,'KVR',12.34,True,2+4j],dtype='object')
>>> a----------array([10, 'KVR', 12.34, True, (2+4j)],
dtype=object)
>>> a.dtype-------dtype('O')
```

---

2)arange():

=>arange() is used generating seqence of numerical values in the form 1-Dimensional array only.[if we want to view then in matrix format, we must use reshape())

Syntax:- arange(start, [stop,step, dtype])

Examples:

```
import numpy as np
>>> a=np.array([10,20,30,40])
>>> a
array([10, 20, 30, 40])
>>> a.ndim
1
```

```
>>> a=np.array([ [10,20],[30,40]])
>>> a.ndim
2
>>> a
array([[10, 20],
       [30, 40]])
>>> a=np.arange(10)
>>> type(a)
<class 'numpy.ndarray'>
>>> a
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
>>> a=np.arange(10, dtype='float')
>>> a
array([0., 1., 2., 3., 4., 5., 6., 7., 8., 9.])
>>> a=np.arange(10)
>>> type(a)
<class 'numpy.ndarray'>
>>> a
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
>>> a.ndim
1
>>> a.shape
(10,)
>>> a.size
10
>>> a.reshape(5,2)
array([[0, 1],
       [2, 3],
       [4, 5],
       [6, 7],
       [8, 9]])
>>> a
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
>>> a.reshape(2,5)
array([[0, 1, 2, 3, 4],
       [5, 6, 7, 8, 9]])
>>> a.reshape(10,1)
array([[0],
       [1],
       [2],
       [3],
       [4],
       [5],
```

```
       [6],
       [7],
       [8],
       [9]])
>>> a.reshape(1,10)
array([[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]])

a=np.arange(10,20)
>>> a
array([10, 11, 12, 13, 14, 15, 16, 17, 18, 19])
>>> a.size
10
>>> a.reshape(2,5)
array([[10, 11, 12, 13, 14],
       [15, 16, 17, 18, 19]])

a=np.arange(10,20,3)
>>> a
array([10, 13, 16, 19])
>>> a.reshape(2,2)
array([[10, 13],
       [16, 19]])
-------------------------------------------------------------------
-----------------------------------------
3)  linspace():
-----------------------
=>This function is used for generating equally spaced numbers will
be generated.
                        (OR)
=> This function returns exactly spaced numbers over a specified
interval
=>this function returns the values in the form float
=>the default value for linspace is 50.

Syntax:
varname=np.linspace(start,stop, lin_num)

Example:-
----------------
a=np.linspace(1,10, 5)
>>> a
array([ 1.  ,  3.25,  5.5 ,  7.75, 10.  ])
>>> a=np.linspace(1,50, 10)
```

```
>>> a
array([ 1.        ,  6.44444444, 11.88888889, 17.33333333,
22.77777778,
       28.22222222, 33.66666667, 39.11111111, 44.55555556, 50.
])

-----------------------------------------------------------------
------------------------------
4) zeros():
-------------
=>This function is used for creating ndarray matrix with zeros
depends on type of shape we specify.
=>This function fills the value '0' (zero)

=>syntax:     np.zeros(shape, dtype)
-------------------------------------------------------
     shape can be  1-d , 2-d   or 3-nd

     defualt data type   is   float

Example:
-------------------------
>>> a=np.zeros(10)
>>> a.ndim
1
>>> a.shape
(10,)
>>> a.size
10
>>> a
array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0.])
>>> a.reshape(5,2)
array([[0., 0.],
       [0., 0.],
       [0., 0.],
       [0., 0.],
       [0., 0.]])
>>> a.reshape(2,5)
array([[0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.]])
>>> a.reshape(3,3)--------->-ValueError: cannot reshape array of
size 10 into shape (3,3)
-----------------------------------------------------------------
```

```
-----------------------------------------------------
 a=np.zeros((3,3))
>>> a
array([[0., 0., 0.],
       [0., 0., 0.],
       [0., 0., 0.]])
>>> a=np.zeros((3,3), dtype='int')
>>> a
array([[0, 0, 0],
       [0, 0, 0],
       [0, 0, 0]])
>>> a=np.zeros((4,3), dtype='int')
>>> a
array([[0, 0, 0],
       [0, 0, 0],
       [0, 0, 0],
       [0, 0, 0]])
-----------------------------------------------------------------------
-----------------------------------------------------------
5) ones():
---------------
>This function is used for  for creating ndarray matrix with ones
depends on type of shape we specify.
=>This function fills the value '1' (one )

=>syntax:     np.ones(shape, dtype)
-----------------------------------------------------------
     shape can be  1-d , 2-d   or 3-nd

     defualt data type   is   float


Examples:
-----------------
a=np.ones(6)
>>> type(a)
<class 'numpy.ndarray'>
>>> a
array([1., 1., 1., 1., 1., 1.])
>>> a=np.ones(6,dtype='int')
>>> a
array([1, 1, 1, 1, 1, 1])
>>> a.reshape(2,3)
```

```
array([[1, 1, 1],
       [1, 1, 1]])
>>> a.reshape(3,2)
array([[1, 1],
       [1, 1],
       [1, 1]])
-------------------------------------------------------------------
------
a=np.ones((3,3),dtype='int')
>>> a
array([[1, 1, 1],
       [1, 1, 1],
       [1, 1, 1]])
>>> a=np.ones((3,4),dtype='int')
>>> a
array([[1, 1, 1, 1],
       [1, 1, 1, 1],
       [1, 1, 1, 1]])
>>> a=np.ones((2,6),dtype='int')
>>> a
array([[1, 1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1, 1]])
-------------------------------------------------------------------
----------------------------
6) full():
------------
The purpose of full() is that to fill the ndarray object with
programmer specified value along with its dtype.

Syntax:-         varname=np.full(shape, fill_value, dtype)


           shape can be  1-d , 2-d   or 3-nd
           fill_value can be programmer musr specify (Ex:- 2,4,5...
           defualt data type   is   int


Examples:
------------------
>> a=np.full(5,fill_value=6)
>>> a
array([6, 6, 6, 6, 6])
>>> a=np.full(shape=(5,), fill_value=8)
```

```
>>> a
array([8, 8, 8, 8, 8])
>>> a=np.full(fill_value=9,shape=(10,))
>>> a
array([9, 9, 9, 9, 9, 9, 9, 9, 9, 9])
>>> a.reshape(5,2)
array([[9, 9],
       [9, 9],
       [9, 9],
       [9, 9],
       [9, 9]])
>>> a.reshape(2,5)
array([[9, 9, 9, 9, 9],
       [9, 9, 9, 9, 9]])
>>> a=np.full(4,2)
>>> a
array([2, 2, 2, 2])
>>> a.reshape(2,2)
array([[2, 2],
       [2, 2]])
>>> type(a)
<class 'numpy.ndarray'>
------------------------------------------------------------------------
---------------------------------------
>>>a=np.full((3,3),5)
>>> a
array([[5, 5, 5],
       [5, 5, 5],
       [5, 5, 5]])
       >>> a=np.full((3,3),1)
>>> a
array([[1, 1, 1],
       [1, 1, 1],
       [1, 1, 1]])
>>> a=np.full((3,4),0)
>>> a
array([[0, 0, 0, 0],
       [0, 0, 0, 0],
       [0, 0, 0, 0]])
------------------------------------------------------------------------
-----------------------------------
7) eye()
-----------------------------------------------------------
```

=>This function is used for generating identity matrix (ie. 1 in principle diagonal elements and 0 in other places)

syntax:- np.eye(n,m=None, k=0,dtype)

Explanation:

=>here 'n' represents number of rows
'm' represents number of columns
k represents principle diagonal (k=0)
if k=-1,-2... represents bellow principle diagonal made as 1
if k=1 2,.....represents above principle diagonal made as 1
=>If we don't pass 'm' value then by default 'n' value will be considered as 'm' value.
Examples:

```
>>a=np.eye(3)
>>> a
array([[1., 0., 0.],
       [0., 1., 0.],
       [0., 0., 1.]])
>>> a=np.eye(3,dtype='int')
>>> a
array([[1, 0, 0],
       [0, 1, 0],
       [0, 0, 1]])
>>> a=np.eye(3,2,dtype='int')
>>> a
array([[1, 0],
       [0, 1],
       [0, 0]])
>>> a=np.eye(3,4,dtype='int')
>>> a
array([[1, 0, 0, 0],
       [0, 1, 0, 0],
       [0, 0, 1, 0]])
>>> a=np.eye(5,5,dtype='int')
>>> a
array([[1, 0, 0, 0, 0],
       [0, 1, 0, 0, 0],
```

```
       [0, 0, 1, 0, 0],
       [0, 0, 0, 1, 0],
       [0, 0, 0, 0, 1]])
>>> a=np.eye(5,5, k=1,dtype='int')
>>> a
array([[0, 1, 0, 0, 0],
       [0, 0, 1, 0, 0],
       [0, 0, 0, 1, 0],
       [0, 0, 0, 0, 1],
       [0, 0, 0, 0, 0]])
>>> a=np.eye(5,5, k=-1,dtype='int')
>>> a
array([[0, 0, 0, 0, 0],
       [1, 0, 0, 0, 0],
       [0, 1, 0, 0, 0],
       [0, 0, 1, 0, 0],
       [0, 0, 0, 1, 0]])
>>> a=np.eye(5,5, k=-2,dtype='int')
>>> a
array([[0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0],
       [1, 0, 0, 0, 0],
       [0, 1, 0, 0, 0],
       [0, 0, 1, 0, 0]])
>>> a=np.eye(5,5, k=-3,dtype='int')
>>> a
array([[0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0],
       [1, 0, 0, 0, 0],
       [0, 1, 0, 0, 0]])
>>> a=np.eye(5,5, k=1,dtype='int')
>>> a
array([[0, 1, 0, 0, 0],
       [0, 0, 1, 0, 0],
       [0, 0, 0, 1, 0],
       [0, 0, 0, 0, 1],
       [0, 0, 0, 0, 0]])
>>> a=np.eye(5,5, k=2,dtype='int')
>>> a
array([[0, 0, 1, 0, 0],
       [0, 0, 0, 1, 0],
       [0, 0, 0, 0, 1],
```

```
       [0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0]])
-------------------------
8)identity()
-------------------------
=>This function is used for generating identity or unit matix
(always square matrix)

syntax:-  np.identity(n,dtype)
=>here 'n' repersents (n,n), i.e n rows and n columns.

Example:
------------------
>>> a1=np.identity(3)
>>> a1
array([[1., 0., 0.],
       [0., 1., 0.],
       [0., 0., 1.]])
>>> a1=np.identity(3, dtype='int')
>>> a1
array([[1, 0, 0],
       [0, 1, 0],
       [0, 0, 1]])
============================================================

                 ============================
                      ndarray
                ============================
=>In numpy module programming, the data  is always stored in the
form of object whose type is 'ndarray'
=>here 'ndarray' is a pre-defined class in numpy module and whose
objects allows to store array of values.
=>an object of ndarray takes less memory space fast in data
analysis.
------------------------------------------------------------------
-----------------------------------------------
Differences between ndarray and tradtional list of python lang
------------------------------------------------------------------
---------------------------
=>an object of ndarray takes less memory space fast in data
analysis and list object takes more memory space.
=>on the object of ndarray, we can perform Vector operations. where
as  on the object of list we can't perform any vector operations.
```

==========

Attrubutes of ndarray :

---

a) ndim---->gives dimesion of an array.
b) shape--->returns shape of an array in the form of tuple ()
c) size------>returns no. of values in the array
d) dtype--->returns data type of ndarray
c) itemsize--->returns the memory space occupied by every element of an array

---

==========

1) Numpy Indexing

==========

=>In indexing Process, we find or obtain a single value from given array by passing valid existing position otherwise we IndexError

=>Syntax for index on 1-D Array:

---

ndarrayobj[index]

Example:

```
>>> a=np.array([10,20,30,40,50,60,70])
>>> a
array([10, 20, 30, 40, 50, 60, 70])
>>> print(a[0])
10
>>> print(a[4])
50
>>> print(a[7])------------IndexError: index 7 is out of bounds for axis 0 with size 7
>>> print(a[6])
70
```

=>Syntax for index on 2-D Array:

2darrayobj[Rowindex, ColIndex]

-
Example:
```
------------------
>>>a=np.array([10,20,30,40,50,60,70,80,90])
>>>b=a.reshape(3,3)
>>> type(b)---------------<class 'numpy.ndarray'>
>>> b
array([[10, 20, 30],
       [40, 50, 60],
       [70, 80, 90]])

>>> a.ndim---------------1
>>> b.ndim---------------2
--------------------------------
>>> b[0,0]
10
>>> b[1,1]
50
>>> b[2,2]
90
-----------------------------
>>> b[-3][-3]
10
>>> b[-3,-3]
10
-----------------------------
>>> b[-13,-13]----IndexError: index -13 is out of bounds for axis 0
with size 3
>>> b[-3,-13]----IndexError: index -13 is out of bounds for axis 1
with size 3
```

==========

Numpy Slicing Operations

==========

=>The purpose of Numpy Slicing Operations is that to extract some range values from the given nd array object.
=>We can perform Numpy Slicing Operations on two things. The are

1). I-D Array
2). 2-D Array

Numpy Slicing Operations on I-D Array:

Syntax1:- ndarrayobject[start:stop]

Examples:

```
>>> a=np.array([10,20,30,40,50,60,70])
>>> a
array([10, 20, 30, 40, 50, 60, 70])
>>> a[0:6]
array([10, 20, 30, 40, 50, 60])
>>> a[2:6]
array([30, 40, 50, 60])
>>> a[2:]
array([30, 40, 50, 60, 70])
>>> a[:6]
array([10, 20, 30, 40, 50, 60])
>>> a[:]
array([10, 20, 30, 40, 50, 60, 70])
```

---

syntax2:- ndarrayobject[start:stop:step]

Rules:

---

1) For 'start' and 'stop' values , we can pass both Possitive and Negative Values.

2) If STEP Value is Possitive then we have to take the elements

from start to stop-1 in Forward direction (left to Right) provided startvalue<stop value.

3) If STEP Value is Negative then we have to take the elements from start to stop+1 in backward direction (Right to left) provided startvalue>stop value.

4) In forward direction, if stop value is '0' then the result is always empty

5) In backward direction, if stop value is '-1' then the result is always empty

---

Forward direction

```
>>> a=np.array([10,20,30,40,50,60,70])
>>> a
array([10, 20, 30, 40, 50, 60, 70])
>>> a[0:7:3]
array([10, 40, 70])
>>> a[0:7:2]
array([10, 30, 50, 70])
>>> a[1:6:2]
array([20, 40, 60])
>>> a[::2]
array([10, 30, 50, 70])
```

Backward direction

```
>>> a=np.array([10,20,30,40,50,60,70])
>>> a
array([10, 20, 30, 40, 50, 60, 70])
>>> a[5:1:-2]
array([60, 40])
>>> a[6:1:-3]
array([70, 40])
>>> a[::-2]
array([70, 50, 30, 10])
```

```
>>> a[2:0:1]
array([], dtype=int32)
```

```
>>> a[4:-1:-1]
array([], dtype=int32)
-----------------------------------------------------------------
---------------------------------------
          Numpy Slicing Operations on  2-D Array:
-----------------------------------------------------------------
---------------------------------------
=>we know 2D array is a collection of rows and columns.

Syntax for 2-D array indexing
-----------------------------------------

                        ndarrayobj[rowindex,colindex]

-----------------------------------------
Syntax for 2-D array slicing
-----------------------------------------
          ndarrayobj[rowindex,colindex]

=>   The above syntax can be represented as follows.

          ndarrayobj[begin:end:step, begin:end:step      ]
                       ----------------------  -------------------
--
                            slice1                 slice2
=>here slice1 represents Rows
=>here slice2 represents Columns

Examples:
-----------------
 a[0:1,0:3]
array([[10, 20, 30]])
>>> a[0:1,:]
array([[10, 20, 30]])
>>> a[0::2,:]
array([[10, 20, 30],
       [70, 80, 90]])
>>> a[0:3:2,:]
array([[10, 20, 30],
       [70, 80, 90]])
>>> a[1:2, : ]
array([[40, 50, 60]])
>>> a[1:2, 0:2 ]
```

```
array([[40, 50]])
>>> a[1:2, :2 ]
array([[40, 50]])
>>> a[ :, 2:]
array([[30],
       [60],
       [90]])
>>> a[1:3,0:3]
array([[40, 50, 60],
       [70, 80, 90]])
>>> a[1:3, :]
array([[40, 50, 60],
       [70, 80, 90]])
>>> a[1:, :]
array([[40, 50, 60],
       [70, 80, 90]])
>>> a[0:2,0:2]
array([[10, 20],
       [40, 50]])
>>> a[:2,:2]
array([[10, 20],
       [40, 50]])
>>> a[0:2,1: ]
array([[20, 30],
       [50, 60]])
>>> a[0:2,1:3]
array([[20, 30],
       [50, 60]])
>>> a[0:2,1:]
array([[20, 30],
       [50, 60]])
>>> a[:,1:2]
array([[20],
       [50],
       [80]])
>>> a[:3,1:2]
array([[20],
       [50],
       [80]])
>>> a[:3,1:2]
array([[20],
       [50],
       [80]])
```

```
>>> a[:3:2,1:2]
array([[20],
       [80]])
>>> a[:3:2,0:1]
array([[10],
       [70]])
>>> a[::2,0:1]
array([[10],
       [70]])
>>> a[::2,1: ]
array([[20, 30],
       [80, 90]])
>>> a[0:3:2,1:3 ]
array([[20, 30],
       [80, 90]])
>>> a[: ,1:]
array([[20, 30],
       [50, 60],
       [80, 90]])
>>> a[::2, ::2]
array([[10, 30],
       [70, 90]])
>>> a[0:3:2, 0:3:2]
array([[10, 30],
       [70, 90]])
```

==========

==========

Numpy Advanced Indexing and Slicing

=>If we want access multiple elements which are not in order / sequence (random elements or arbitrary elements) then we must advanced indexing.

Numpy Advanced Indexing and Slicing on 2D--Array

=>Syntax:- ndarrayobject[[row indices],[column indices]]

Examples:

```
a=np.array([[10,20,30],[40,50,60],[70,80,90]])
```

```
>>> a
array([[10, 20, 30],
       [40, 50, 60],
       [70, 80, 90]])


>>> a[[0,2],[0,2]]
array([10, 90])
>>> a[[0,1,2],[0,1,2]]
array([10, 50, 90])
>>> a[ [0,1,2],[2,1,0]]
array([30, 50, 70])
>>> a[[1,2,2],[2,1,2]]
array([60, 80, 90])
------------------------------------------------------------------------
------------------------
Numpy Advanced Indexing and Slicing on 1D--Array
------------------------------------------------------------------------
------------
Syntax:-     ndarryobj[ k ]

=>here  k represents either list object or boolean list

Examples:
-----------------
>>> a=np.array([100,200,300,400,500,600])
>>> print(a)-------------[100 200 300 400 500 600]

Example:
--------------
>>> print(a)
[100 200 300 400 500 600]
--------------
Way-1
--------------
>>> lst=[0,5]
>>> a[lst]
array([100, 600])
>>> a[ [0,5] ]
array([100, 600])
--------------------------
Way-2
--------------------------
```

```
>>> a[[True,False,False,False,False,True]]
array([100, 600])

>>> a[[True,False,False,True,False,True]]
array([100, 400, 600])
```

```
==================================================
                3-D Arrays in Numpy
==================================================
=>We know that 1-D array Contains Some Sequence of elements in 1-
Direction / row

=>Example:-         >>>a=np.array([10,20,30,40]
                   >>>print(a)---->array([10,20,30,40]---- 1-D Array

=>We know that 2-D array Contains data in the form of Rows and cols
=>2-D array contains combination of 1-D arrays.
Examples:
-------------------
>>> a=np.array([ [10,20,30],[40,50,60],[70,80,90]])
>>> a
array([[10, 20, 30],
       [40, 50, 60],
       [70, 80, 90]])
>>> a.ndim--------------2
------------------------------------------------------------------
---
=>3-D array contains combination of 2-D arrays with rows and
columns.
Examples:
----------------
```

```
>>>
a=np.array([[[10,20,30],[40,50,60],[70,80,90]],[[1,2,3],[4,5,6],[7,
8,9]]])
>>> a.ndim
3
>>> a.shape
(2, 3, 3)
>>> a
array([[[10, 20, 30],
        [40, 50, 60],
        [70, 80, 90]],

       [[ 1,  2,  3],
        [ 4,  5,  6],
        [ 7,  8,  9]]])
>>>
a=np.array([[[1,2,3,4],[5,6,7,8],[9,10,11,12]],[[13,14,15,16],[17,1
8,19,20],[21,22,23,24]] ] )
>>> a
array([[[ 1,  2,  3,  4],
        [ 5,  6,  7,  8],
        [ 9, 10, 11, 12]],

       [[13, 14, 15, 16],
        [17, 18, 19, 20],
        [21, 22, 23, 24]]])
>>> a.ndim
3
>>> a.shape
(2, 3, 4)
-----------------------------------------------------------------
---------------------------------
Slicing Operation on 3-D Arrays:
---------------------------------------------------
Syntax for 1-D:    1darray[begin:end:step]

Syntax for 2-D:    2darray[ROW,COLUMN]
                          (or)
           2darray[begin:end:step ,begin:end:step]


Syntax for 3-D:    3darray[Dimension,ROW,COLUMN]
                          (or)
```

```
     3darray[begin:end:step,begin:end:step,begin:end:step]

Examples:-
------------------
>>> a[ : , : , 0:1]
array([[[ 1],
        [ 5],
        [ 9]],

       [[13],
        [17],
        [21]]])
>>> a[0:2, 0:3, :1]
array([[[ 1],
        [ 5],
        [ 9]],

       [[13],
        [17],
        [21]]])
>>> a[:,0:2,1:3]
-------------------------------------------------------------------
-----------------
Advanced Indexing on 3-Darrays:
-----------------------------------------------

Syntax:-
---------------

np.array([indexes of 2darray of random elements],[indexes of row of
random elements ],[indexex of columns of random elements] )

Example:
>>> a[[0,0,0],[1,0,1],[1,2,3]]
array([6, 3, 8])
>>> a[[0,1,1],[0,0,2],[0,1,3]]
array([ 1, 14, 24])
>>> a[ [0,0,0,1,1,1],[0,1,2,0,1,2],[2,1,0,2,1,0] ]
array([ 3,  6,  9, 15, 18, 21])
===================================================================
```

==========

Numpy--BroadCasting /Arithmetic Operations

==========

=>To perform Arithmetic Operations on numpy arrays, we use the following functions.

1) add()
2) subtract()
3) multiply()
4) divide()

Syntax:-

```
np.add(array1,array2)
np.subtract(array1,array2)
np.multiply(array1,array2)
np.divide(array1,array2)
```

=>Instead of using these functions, we cal also use all arithmetic Operators of python.

Examples:
==========
```
>>> import numpy as np
>>> a=np.array([[10,20],[30,40]])
>>> b=np.array([[1,2],[3,4]])
>>> a
array([[10, 20],
       [30, 40]])
>>> b
array([[1, 2],
       [3, 4]])
>>> print(a+b)
[[11 22]
 [33 44]]
>>> print(a-b)
[[ 9 18]
 [27 36]]
>>> print(a*b)
[[ 10  40]
 [ 90 160]]
>>> print(b-a)
[[ -9 -18]
 [-27 -36]]
>>> print(a/b)
[[10. 10.]
 [10. 10.]]
>>> print(a//b)
[[10 10]
 [10 10]]
>>> print(a%b)
[[0 0]
 [0 0]]
>>> print(a**b)
[[     10     400]
 [  27000 2560000]]
>>> np.add(a,b)
array([[11, 22],
       [33, 44]])
>>> np.subtract(a,b)
```

```
array([[ 9, 18],
       [27, 36]])
>>> np.multiply(a,b)
array([[ 10,  40],
       [ 90, 160]])
>>> np.divide(a,b)
array([[10., 10.],
       [10., 10.]])
================================================================
Examples : Arithmetic Operations with Different Dimensions
(BroadCasting)
>>> a=np.array([[10,20,30],[40,50,60]])
>>> a
array([[10, 20, 30],
       [40, 50, 60]])
>>> b=np.array([2,5,6])
>>> b
array([2, 5, 6])
>>> a.ndim
2
>>> b.ndim
1
>>> print(a+b)
[[12 25 36]
 [42 55 66]]
>>> print(a-b)
[[ 8 15 24]
 [38 45 54]]
>>> print(a*b)
[[ 20 100 180]
 [ 80 250 360]]
>>> print(a/b)
[[ 5.  4.  5.]
 [20. 10. 10.]]
>>> print(a//b)
[[ 5  4  5]
 [20 10 10]]
>>> print(a%b)
[[0 0 0]
 [0 0 0]]
 ----------------------------------------------------------------
----------------------------------------------------------
 >>> np.add(a,b)
```

```
array([[12, 25, 36],
       [42, 55, 66]])
>>> np.subtract(a,b)
array([[ 8, 15, 24],
       [38, 45, 54]])
>>> np.multiply(a,b)
array([[ 20, 100, 180],
       [ 80, 250, 360]])
>>> np.divide(a,b)
array([[ 5.,  4.,  5.],
       [20., 10., 10.]])
```

==========

Numpy--Statical Functions

==========

=The most essential statical functions are

1) amax
2) amin
3)mean
4) median
5) variance
6) standard deviation

1) amax() and amin():

=>These functions are used for finding max and min from given array.

Syntax:-

```
np.amax(array)
np.amin(array)

np.amax(array, axis=0)   # here 0 rep columns
np.amin(array, axis=1)   # here 1 rep rows
```

Examples:

```
>>> b=np.array([[2,6],[4,8]])
>>> np.amax(b)
8
>>> np.amin(b)
2
>>> np.amin(b,axis=0)
array([2, 6])
>>> np.amin(b,axis=1)
array([2, 4])
```
---
2) mean():

=>This is used for finding mean of the given array

Syntax:- np.mean(array)

```
np.mean(array, axis=0)
np.mean(array, axis=1)
```

Examples:
```
>>> b=np.array([[2,6],[4,8]])
>>> np.mean(b)
5.0
>>> np.mean(b,axis=0)
array([3., 7.])
>>> np.mean(b,axis=1)
array([4., 6.])
```

---
3) median()

=>This is used for finding median of the given array

```
Syntax:-      np.median(array)

                  np.median(array, axis=0)
                  np.median(array, axis=1)

Examples:
>>> b=np.array([[2,6],[4,8]])
>>> np.median(b)
5.0
>>> np.median(b,axis=0)
array([3., 7.])
>>> np.median(b,axis=1)
array([4., 6.])
-------------------------------------------------------------------
--------
5) variance:
--------------------
=>used for finding variance of an array

Syntax:-      np.var(array)

                  np.var(array, axis=0)
                  np.var(array, axis=1)

Examples:
-----------------
>>> b=np.array([[2,6],[4,8]])
>>> np.var(b)
5.0
>>> np.var(b,axis=0)
array([1., 1.])
>>> np.var(b,axis=1)
array([4., 4.])
-----------------------------------------------------------
6) standard deviation
---------------------------------
=>used for finding standard deviation of an array
Syntax:-      np.std(array)

                  np.std(array, axis=0)
                  np.std(array, axis=1)
Examples:
```

```
------------------------
>>> b=np.array([[2,6],[4,8]])
>>> np.std(b)
2.23606797749979
>>> np.std(b,axis=0)
array([1., 1.])
>>> np.std(b,axis=1)
array([2., 2.])
=======================================================================
=
```

==========

Introduction to PANDAS

=>PANDAS is an open Source Python Library(C+PYTHON lang)
=>PANDAS provides High Performnce for doing Data Analysis. In otherwords PANDAS is one of the Data Analysis tool.
=>The word PANDAS is derived a word "PANel DAta
=>The PANDAS Module / Library developed by WES McKinney in the year 2008

=>Using PANDAS, we can accomplish five steps in processing and analysis of data.

1) load
2) Preapre
3) Manipulate
4) Model
5) Analyze.

=>Python with PANDAS can be used in wide orange of fileds like Commercial Domains like Finacial Sectors, stastical Sectors and Economical Sectors...etc

---

Instaling Pandas module:

=>Standard python software does not come with pandas module. So that we have to install pandas explicitly by using PIP tool (Python Package Installer)

Syntax:- pip install pandas

Data Structures in Pandas

=>Data Structures makes us to understand how effectively to store the data.

=>In pandas, we have 3 types of Data Structures. They are

a) Series
b) DataFrame
c) Panel

Dimensions and Descriptions

=>The best way to think of these data structures is that , The High dimensional data structure is a container of its lower dimensional data structure.

Example:- DataFrame is a container of Series
Panel is a container of DataFrame

==========

Series

==========

=>It is one of the one dimensional labelled array capable of holding homogeous data of any type(Integer, String, float,python object)
=>The axis labels are collectively called Index (rows)
=>Pandas Series is nothing but a column in an excel sheet
=>Pandas Contains Homogeneous Data
=>Pandas size is immutable
=>Pandas Series values are mutable

Creating a Series

=>We know that Series of the one dimensional labelled array capable of holding homogeous data of any type(Integer, String, float,python object).
=>To create a Series, we use a pre-defined function Series()

=>Syntax:-

```
varname=pandas.Series(data, index,dtype)
```

Explanation:-

=>here varname is an object of <class, 'pandas.core.series.Series'>
=>Data represents ndarray,list,constants
=>index represents rows . In otherwords Index value must be Unique and they must be equal to number of values. By default index starts from 0 to n-1. Programatically we can assign our own index names.
=>dtype represents data type (int32,int64, float32,float64...etc)

=>We have two types of Series. They are

a) empty series
b) non-empty series

```
a) empty series:
----------------------------
=>An empty series does not contain any values.
Syntax / example:        s=pd.Series()
                         print(s)-------------Series([], dtype:
float64)
-------------------------------------------------------------------
----------------------
b)b) non-empty series:
--------------------------------
=>An non-empty series contains many values.

Syntax        s=pd.Series(data)

===================================================================
Example:    create a series for 10 20 30 40
--------------
>>s=pd.Series([10,20,30,40])
>>> print(s)
0    10
1    20
2    30
3    40
dtype: int64
-------------------------------------------------------------------
---
>>> s=pd.Series([10,20,30,40],dtype='float')
>>> print(s)
0    10.0
1    20.0
2    30.0
3    40.0
dtype: float64
-------------------------------------------------------------------
----
create a series object w.r.t nd array object.
-------------------------------------------------------------------
-----
>>> a=np.array([100,200,300,400])
>>> a
array([100, 200, 300, 400])
>>> s=pd.Series(a)
>>> print(s)
```

```
0    100
1    200
2    300
3    400
dtype: int32
------------------------------------------------------------------
-------------
create a series object by using  list object
------------------------------------------------
>>> lst=[10,"KVR",23.45,True]
>>> s=pd.Series(lst)
>>> print(s)
0       10
1      KVR
2    23.45
3     True
dtype: object
>>> s=pd.Series([10,"KVR",23.45,True],dtype='object')
>>> print(s)
0       10
1      KVR
2    23.45
3     True
dtype: object
------------------------------------------------------------------
------------------
The above examples, uses default index . ie. 0 t0 n-1 values
==================================================================
Creating a Series object with Programmer-defined Index
------------------------------------------------------------------
-----------------------------------------
Examples:
---------------------------
>>>
s=pd.Series(["Ramu","Karan","DD","Rossum","Brucelee"],[100,101,102,
103,104])
>>> print(s)
100        Ramu
101       Karan
102          DD
103      Rossum
104    Brucelee
dtype: object
```

```
>>>
s=pd.Series(data=["Ramu","Karan","DD","Rossum","Brucelee"],index=[1
00,101,102,103,104])
>>> print(s)
100        Ramu
101       Karan
102          DD
103      Rossum
104    Brucelee
dtype: object
------------------------------------------------------------------
-------------------------------------
>>> s=pd.Series(["Rama
Krishna","Raja","Rani"],["Father","Son","Daughter"])
>>> print(s)
Father      Rama Krishna
Son                 Raja
Daughter            Rani
dtype: object
==================================================================
=>Creating a Series from dict:
==================================================================
=>dict object organizes the data in the form of (key,value)
=>If we use dict object in a series() then keys of dict object can
be taken as Indices automatically and corresponding values of dict
can be takes data of the Series()

Example:
-----------------
>>> d1={100:"Rossum",101:"Gosling",200:"Ritche"}
>>> print(d1)
{100: 'Rossum', 101: 'Gosling', 200: 'Ritche'}
>>> s=pd.Series(d1)
>>> print(s)
100     Rossum
101    Gosling
200     Ritche
dtype: object


IMP:-  If we pass Index Values explicitly through Series() then
Values in dict will be pull out  or taken out by matching
programmer-defined indexes with key of Dict object.
```

If the Programmer-defined indexes are not found then we get NaN(Not a Number) in the result.

Example:

```
-----------------
>>> s=pd.Series(d1,[150,250,100,200])
>>> print(s)
150       NaN
250       NaN
100    Rossum
200    Ritche
dtype: object
------------------------------------------------------------------------
------------------------------
>>> s=pd.Series(10)
>>> print(s)
0    10
dtype: int64
------------------------------------------------------------------------
------------------------------
>>> s=pd.Series(10,["Apple","Mango","Kiwi"])
>>> print(s)
Apple    10
Mango    10
Kiwi     10
dtype: int64
```

Accessing the data from Series

=>To access the data from Series object, we have two approaches. They are

a) By using Position / Index
b) By using Label

---

a) By using Position / Index:

Syntax:-

```
seriesobj[Index]
seriesobj[begin:end]
seriesobj[begin:end:step]
```

Examples:

```
>>> s=pd.Series([1,2,3,4,5],['a','b','c','d','e'])
>>> print(s)
a    1
```

```
b    2
c    3
d    4
e    5
dtype: int64
>>> print(s[3])
4
>>> s=pd.Series([1,2,3,4,5],['a','b','c','d','e'])
>>> print(s)
a    1
b    2
c    3
d    4
e    5
dtype: int64
>>> print(s[0:3])
a    1
b    2
c    3
dtype: int64
>>> print(s[0:3:2])
a    1
c    3
dtype: int64
>>> print(s[::2])
a    1
c    3
e    5
dtype: int64
-------------------------------------------------------------------
------------------------------
2) By using Label:
---------------------------------
=>A series is a like a fiexed size dict object, in that we can set
and get values by index label.
Syntax:-
               seriesobj['labelname']   ----------->getting the
value

               seriesobj['labelname']= Value---->setting the value

Examples:
------------------
```

```
>>> s=pd.Series([1,2,3,4,5],['a','b','c','d','e'])
>>> s
a    1
b    2
c    3
d    4
e    5
dtype: int64
>>> print(s['a'])
1
>>> s['a']=100
>>> s
a    100
b      2
c      3
d      4
e      5
dtype: int64
>>> print(s['b'])
2
>>> print(s[['b','c','e']])
b    2
c    3
e    5
dtype: int64
```

```
=======================================
                    DataFrame in PANDAS
          ======================================
=>A DataFrame is one of the Two dimensional Labelled Data Structure to
organize the data in the form rows and columns.
=>In Otherwords, DataFrame stores the data in the from Rows and columns.

Features of DataFrame:
---------------------------------
1) The column names of DataFrame can be different or same data type (int,
str,float..etc)
2) The size of DataFrame is mutable
3) DataFrame contains two axis(Row and Column)

=>No of approaches to create Data Frame:
-------------------------------------------------------------------------
=>We have the following  approaches to create DataFrame.
     a) List                      d) with CSV file
     b) dict                e) ndarray
     c) Series
-------------------------------------------------------------------------
--------------------------------
Syntax for creating a DataFrame:
-----------------------------------------------------
     varname=pandas.DataFrame(data, index, columns,dtype)

Explanation:
-------------------
1) here varname reprsents an object of <class
'pandas.core.frame.DataFrame'>
2) DataFrame() is a function used for preparing Two-dimension Labelled
Data Structure.
3) data represents list ,dict,series,CSV file and ndarray
d) index represents row labels. By default row labels starts from 0 to
n-1. Programtically we can give our own row label names
e) columns represents Column Labels. By default column labels starts from
0 to n-1.Programtically we can give our own column label names
f) dtype represents data type of column names.
=====================================================================
Example1:      creating an empty data frame
------------------
>>>import pandas as pd
>>>df=pd.DataFrame()
>>> print(df)
           Empty DataFrame
           Columns: []
           Index: []
```

```
--------------------------------------------------------------------------------
---------------------------------
Example2:      creating an non-empty data frame with list
------------------
>>>lst=[10,20,30,40,50]
>>>df=pd.DataFrame(lst)
>>>print(df)
                            0<-----Column
                       0  10
                       1  20
                       2  30
                       3  40
                       4  50

>>> df=pd.DataFrame(["apple","kiwi","mango","Guava"])
>>> print(df)
                              0
                       0  apple
                       1   kiwi
                       2  mango
                       3  Guava

Example2:      creating an non-empty data frame with inner list
----------------
>>> lst=[ [10,"KVR"],[20,"Rossum"],[30,"Ritche"],[40,"Gosling"] ]
>>> df=pd.DataFrame(lst)
>>> print(df)
                        0        1
                  0  10      KVR
                  1  20   Rossum
                  2  30   Ritche
                  3  40  Gosling
>>> lst=[ [10,20,30,40],["abc","pqr","klm","zzz"] ]
>>> df=pd.DataFrame(lst)
>>> print(df)
                         0     1      2       3
                  0   10   20    30     40
                  1  abc  pqr  klm  zzz
--------------------------------------------------------------------------------
---------------------------------------------
Example:- Create a DataFrame with Inner list with Programmer defined
Labels for rows and    columns.

>>> lst=[ ["Trump",60],["Jobiden",76],["Obamma",60] ]
>>> print(lst)
[['Trump', 60], ['Jobiden', 76], ['Obamma', 60]]
>>> df=pd.DataFrame(lst,index=[1,2,3],columns=['Name','Age'])
```

```
>>> print(df)
                      Name  Age
                1    Trump   60
                2  Jobiden   76
                3   Obamma   60
-----------------------------------------------------------------
Example:
-----------------------------------------------------------------
>>> df=pd.DataFrame(lst,columns=['Name','Age'])
>>> print(df)
      Name      Age
0    Trump     60
1  Jobiden     76
2   Obamma  60
-----------------------------------------------------------------------------
--------------
Example:
-----------------------------------------------------------------------------
-------------
>>>
df=pd.DataFrame(lst,index=[1,2,3],columns=['Name','Age'],dtype='float')
>>> print(df)
      Name   Age
1    Trump  60.0
2  Jobiden  76.0
3   Obamma  60.0
-----------------------------------------------------------------------------
----------------------
Example:- Create a DataFrame with ndarray with Programmer defined Labels
for rows and                      columns.
-----------------------------------------------------------------------------
-------------------------------------------------
>>> lst=[ ["Trump",60],["Jobiden",76],["Obamma",60] ]
>>> a=np.array(lst, dtype='object')
>>> df=pd.DataFrame(a,index=[1,2,3],columns=['Name','Age'],dtype='object')
>>> print(df)
                      Name     Age
                1    Trump     60
                2   Jobiden    76
                3   Obamma  60
-----------------------------------------------------------------------------
-------------------------------------------------
Example:- Create a DataFrame with dict with list with Programmer defined
Labels for rows and columns.
-----------------------------------------------------------------------------
---------------------------------
Examples:
```

```
------------------------------------------------------------------------
----------------------
>>> studs={"Names":["Divya","Kushrga","Sharada","Sravanthi"],
...         "sub1":["PYTHON","Java","Data Sc","AI"],
...         "sub2":["PYTHON","Java","Data Sc","AI"]}
>>> print(studs)
{'Names': ['Divya', 'Kushrga', 'Sharada', 'Sravanthi'], 'sub1': ['PYTHON',
'Java', 'Data Sc', 'AI'], 'sub2': ['PYTHON', 'Java', 'Data Sc', 'AI']}
>>> type(studs)
<class 'dict'>
>>> df=pd.DataFrame(studs,index=["stud1","stud2","stud3","stud4"])
>>> print(df)

            Names     sub1     sub2
stud1      Divya   PYTHON   PYTHON
stud2    Kushrga     Java     Java
stud3    Sharada  Data Sc  Data Sc
stud4  Sravanthi       AI       AI
========================================================================
Example:- Create a DataFrame Series object
------------------------------------------------------------------------
-------
>>> s=pd.Series(["Divya","Kushrga","Sharada","Sravanthi"])
>>> s
0        Divya
1      Kushrga
2      Sharada
3    Sravanthi
dtype: object
>>> df=pd.DataFrame(s)
>>> df
           0
0      Divya
1    Kushrga
2    Sharada
3  Sravanthi
>>> df=pd.DataFrame(s, columns=['Names'])
>>> df
       Names
0      Divya
1    Kushrga
2    Sharada
3  Sravanthi
------------------------------------------------------------------------
--------------------------------------
Example:- Create a DataFrame with Dict of Series objects
```

```
-------------------------------------------------------------------------
-------
>>>d={"Names": pd.Series(["Divya","Kushrga","Sharada","Sravanthi"]),
      "sub1"   : pd.Series(["PYTHON","Java","Data Sc","AI"]),
      "sub2"   : pd.Series(["Oracle","MySQL","SQLite3","DB2"] )   }

>>> df=pd.DataFrame(d)
>>> print(df)

                     Names     sub1     sub2
                0      Divya   PYTHON   Oracle
                1    Kushrga     Java    MySQL
                2    Sharada  Data Sc  SQLite3
                3  Sravanthi       AI      DB2

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


>>> studs={"Names":["Divya","Kushrga","Sharada","Sravanthi"],
...         "sub1":["PYTHON","Java","Data Sc","AI"],
...         "sub2":["Oracle","MYSQL","SQLITE3","DB2"]}
>>> print(studs)
{'Names': ['Divya', 'Kushrga', 'Sharada', 'Sravanthi'], 'sub1': ['PYTHON',
'Java', 'Data Sc', 'AI'], 'sub2': ["Oracle","MYSQL","SQLITE3","DB2"]}
>>> type(studs)
<class 'dict'>
>>> df=pd.DataFrame(studs,index=[100,101,102,103])
>>> print(df)
>>> print(df)
                       Names     sub1     sub2
                100      Divya   PYTHON   Oracle
                101    Kushrga     Java    MYSQL
                102    Sharada  Data Sc  SQLITE3
                103  Sravanthi       AI      DB2
-------------------------------------------------------------------------
---------------------------------
crkt={"player names:" : pd.Series([........]),
       "ages:" :pd.Series([.....]),
       "rating" :pd.Series([.......]) }
df=pd.DataFrame(crkt)
----------------
Examples:
----------------
d ={'Player
Name':pd.Series(['Tom','James','Ricky','Vin','Steve','Smith','Jack','Lee',
'David','Gasper','Betina','Andres']),
   'Age':pd.Series([25,26,25,23,30,29,23,34,40,30,51,46]),
```

```
'Rating':pd.Series([4.23,3.24,3.98,2.56,3.20,4.6,3.8,3.78,2.98,4.80,4.10,3
.65])}

>>> df=pd.DataFrame(d)
>>> print(df)
                   Player Name  Age  Rating
                0          Tom   25    4.23
                1        James   26    3.24
                2        Ricky   25    3.98
                3          Vin   23    2.56
                4        Steve   30    3.20
                5        Smith   29    4.60
                6         Jack   23    3.80
                7          Lee   34    3.78
                8        David   40    2.98
                9       Gasper   30    4.80
                10      Betina   51    4.10
                11      Andres   46    3.65
-------------------------------------------------------------------------
---------------------------------------
>>> cart = {'Product': ['Mobile', 'AC', 'Laptop', 'TV', 'Football'],
...         'Type': ['Electronic', 'HomeAppliances', 'Electronic',
...                  'HomeAppliances', 'Sports'],
...         'Price': [10000, 35000, 50000, 30000, 799]
...        }
>>> type(cart)
<class 'dict'>
>>> df=pd.DataFrame(cart)
>>> print(df)
               Product            Type                 Price
           0    Mobile      Electronic              10000
           1        AC         HomeAppliances  35000
           2    Laptop      Electronic              50000
           3        TV         HomeAppliances   30000
           4  Football          Sports                799
-------------------------------------------------------------------------
-------------------------------
=>To select the data from DataFrame object we use

           dataframeobject['column name']

>>> print(df['Product'])
                            0      Mobile
                            1          AC
                            2      Laptop
                            3          TV
```

```
                            4    Football
                            Name: Product, dtype: object
>>> print(df['Type'])
                            0        Electronic
                            1    HomeAppliances
                            2        Electronic
                            3    HomeAppliances
                            4            Sports
>>> print(df['Price'])
                            0    10000
                            1    35000
                            2    50000
                            3    30000
                            4      799
                            Name: Price, dtype: int64
------------------------------------------------------------------------------
--------------------
Querying the data from DataFrame object with loc()
------------------------------------------------------------------------------
--------------------
Syntax:-          varname=dataframeobj.loc[dataframeobj['columnname'] cond
stmt]
------------------------------------------------------------------------------
--------------------
Examples:
------------------------------------------------------------------------------
-------------------
>>> data=df.loc[df['Price']>500]
>>> print(data)
    Product            Type  Price
0    Mobile      Electronic  10000
1        AC  HomeAppliances  35000
2    Laptop      Electronic  50000
3        TV  HomeAppliances  30000
4  Football          Sports    799

>>> data=df.loc[df['Price']<1000]
>>> print(data)
    Product    Type  Price
4  Football  Sports    799


>>> data=df.loc[df['Type']=='Electronic']
>>> print(data)
  Product        Type  Price
0  Mobile  Electronic  10000
2  Laptop  Electronic  50000
```

Opening(Reading /writting) csv without pandas

1)READING

```
#readcsvdata.py
#in csv modulle  reader()---> syntax:
result=csv.reader(filepointer)
import csv
with open("stud.csv") as csvrp:
     records=csv.reader(csvrp)
     for rec in records:
          print(rec)
```

2)WRITTING

```
import csv
rec=[63, "viswa",66.66]
with open("G:\\KVR-PYTHON-4PM\\PANDAS\\stud.csv","a") as csvwp:
     wpen=csv.writer(csvwp)
     wpen.writerow(rec)
     print("\nData written to the CSV File")
```

Opening(Reading) csv with pandas

```
#pandascsv.py
import pandas as pd
recs=pd.read_csv("G:\\KVR-PYTHON-4PM\\PANDAS\\stud.csv")
print("The Data Available :\n",recs)
```