
Getting Started with Python

Index

- =>History of python
 - =>Versions of Python
 - =>Downloading and Installation Process of Python Software(www.python.org)
 - =>Python Inspired From
-

History of python

- =>Python Programming Language Conceived in the Year 1980
 - =>Python Programming Language Implementation(Bring into action) was started in the year 1989
 - =>Python Programming Language Officially Released in the Year 1991 Feb 20th
 - =>Python Programming Language Developed by "GUIDO VAN ROSSUM".
 - =>Python Programming Language Developed at Centrum Wiskunde Informatica(CWI) in Nether Lands and Executed on Ameba OS.
 - =>ABC Programming Language is the predecessor of Python Programming Language.
 - =>Python Programming Language Maintained by a Non-Commercial Organization called "Python Software Foundation (PSF) " and whose official website is www.python.org
-
-

Versions in Python

- =>Python Programming contains 3 Types of Versions. They are
 - 1) Python 1.x Here 1 is called Major Version and Here x Represents 0 1 2 3etc(outdated)
 - 2) Python 2.x---Here 2 is called Major Version and Here x Represents 0 1 2 3 4 5
 - 3) Python 3.x--->Here 3 is called Major Version and Here x Represents 0 1 2 3 4 5 6 7 8 9 10 11 (Preview Version) 12 (Future Version)
- 6 7 (ou
-
-

=>Python Software does not provide Backward Compatability.

Python Inspired From

- =>Functional Programming from C
- =>Object Oriented Programming Principles from C++ OOPs
- =>Modular programming Language from Modulo3
- =>Scripting Programming Language from PERL

Features of Python Programming

=>Features of a language are nothing but services or facilities provided by Language Developers and available in Language which are used by Programmers for developing Real Time Applications.

=>Python programming Provides 11 Features. They are

1. Simple
 2. Freeware and Open Source
 3. Dynamically Typed
 4. Platform Independent
 5. Interpreted
 6. High Level
 7. Robust (Strong)
 8. Both Procedure and Object Oriented Programming Language
 9. Extensible
 10. Embedded
 11. Support for Third Party APIs such as Numpy, Pandas, matplotlib, seaborn, NLP, keras , scipy and scikit
-

1. Simple

=>Python is one of the simple Programming language, bcoz of 3 important Technical Features. They are

1. Python Programming Provides "Rich set of Modules (Libraries). So that Python Programmer can re-use the pre-defined modules and develop real time application easily.

Def. of Module:

A module is a collection Functions, Variables and Class names

Examples: math, cmath, calendar, random.....etc

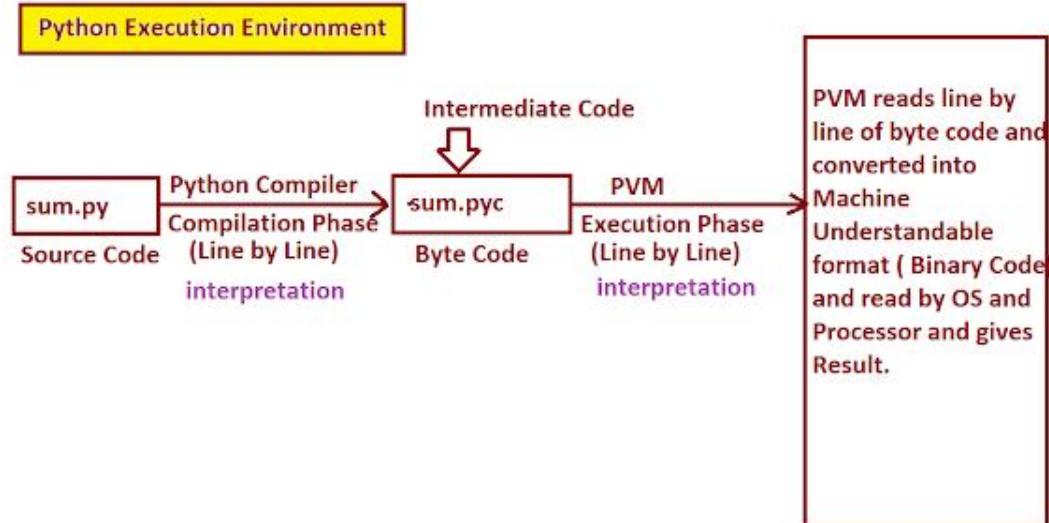
2. Python Programming Provides In-built facility "Garbage Collector". So that Garbage Collector Collects Un-used Memory space and Improves performance of python Real Time Applications.

Definition of Garbage Collector:

=>A Garbage Collector is one of the Python In-built background Program, which is running behind of Every Python Regular program and whose Role is to Collect or Remove Un-used Memory space and improves performance of python Real Time Applications.

Hence Garbage Collector takes care about automatic memory management.

3. Python Programming Provides User-Friendly Syntaxes. So that we can develop Error-Free Program in Limited Span of time.



3. Dynamically Typed

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

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

1. Static Typed Programming Languages

=>In This Programming Languages, It is mandatory to declare Variables by Using variable Declaration where it contains Data Types and Variables Names.

=>Without Variable Declaration, we cant store the data.

Examples: C,C++,JAVA,C#.Net...etc

```
int a,b,c // Variable Declaration--Mandatory.  
a=10  
b=20  
c=a+b
```

2. Dynamically Typed Programming Languages

=>In This Programming Languages,It is not necessary to use Variable Declaration.

=>Internally, depends on type of Value we store or assign to a variable, automatically Python Execution Environment will allocate memory space by using Appropriate Data Types.

Examples Software: Python

Examples:

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

=>In Python Programming, All Values are stored in the form of OBJECTS and Behind of object there exist CLASS.

4. Platform Independet

=>In IT, "Platform" is nothing but type of OS Being Used to run the application.

=>In IT, we have two Types of Programming Languages. They are

1. Platform Dependent languages.
2. Platform Independet languages.

1. Platform Dependent languages.

=>A Language is said to Platform Dependent iff whose data types differs their memory space from One OS another OS.

Examples: C,CPP...etc

2. Platform Independent languages.

=>A Language is said to Platform Independent iff whose data types takes Same memory space on All Types OSes. (Java Slogan)

=>A Language is said to Platform Independent iff whose OBJECTS takes Same memory space on All Types OSes and There Restriction on size of Data which is present in object(Behind of objects there exist class). (Python Slogan)

Example: Java , PYTHON

6. High Level Programming

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

- a) Low Level Programming Languages.
 - b) High Level Programming Languages.
-

a) Low Level Programming Languages:

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

Examples: - a=0b11111000011101010---binary data

 b=0o23----octal

 c=0xface---Hexa Decimal

b) High Level Programming Languages.

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

Example : Python

=====X=====

===== Robust (Strong) =====

=>Python is one of the Robust bcoz of "Exception Handling".

Definition of Exception: Every RuntimeError(Invalid Input) is called Exception

Every Exception in any language by default gives Technical Error Messages. Which are understandable by Programmer but not by End-Users. This Process is Not recommended in Industry.

=>Industry alway recommends to generate User-Friendly Error Messages by Using Exception Handling.

Definition of Exception Handling:- The Process of Converting Technical Error Messages into User-Friendly Error Messages is called Exception Handling.

2. Freeware and Open Source

=>FreeWare

=>If any software is downloaded freely from Official Websites then that software is called FreeWare.

Examples: Python, Java...etc

=>OpenSource:

=>The standard Python name is CPYTHON.

=>Many Software Company vendors came forward and Customized the CPYTHON and the customized versions of CPYTHON used in their companies as In-house tools.

=>The Customized Versions of CPYTHON are called "Python Distributions".

=>Some of the Python Distributions are

1. JPython or Jython---->Used for Running JAVA Based Applications
2. IronPython or IPython--->Used for Running C#.Net Applications.
3. Micro Python----->Used developing Micro Controller Applications.
4. Stackless Python---->Used Developing Concurrency Applications.
5. Anaconda Python---->Used for Dealing BigData / Hadoop Applications
6. Ruby Python----->Used to Ruby on Rails Based Applications
.....etc

===== 9. Extensible =====

=>Extensible feature in Python refers that we can write some of Python Code in Other languages Like C, CPP, Java, HTML....etc.

=>It means that it can be extended to other languages and makes other languages programmer easy in writing and Re-using code of Python and Hence Python Extensible Language.

===== 10.Embedded =====

=>Embedded feature of Python refers, Python Program can also call Other language codes.

=>For Example. Inside Python Program, we can use C, C++ and Java Code .

Note: Python is one of the comfortable Programming Language and not a Fastest Programming language.

===== 11. Support for Third Party APIs such as Numpy ,Pandas , matplotlib, seaborn, NLP, keras , scipy and scikit =====

=>Python Programming Uses Third Party Modules for

Complex Mathematical Operations(numpy--Travis),
Data Analysis and Data Analytics--(Pandas--WES MCKINNEY--
www.pandas.org)

Data Visualization-----Matplot lib...etc

Literals and Its Types

=>A Literal is nothing but a value passing as input to the program.

=>In Python Programming, Primarily, Literals are classified into 5 types. They are

1. Integer Literals-----Example--> 234 567 23
2. String Literals-----Examples-->"Python", "Rossum", "Ram"
3. Float Literals-----Examples--> 34.56 4.5 99.99 0.999
4. Boolean Literals-----Examples----> True False
5. Date Literals-----Examples:----> 29-08-2022, 17-08-

2022...etc

Identifiers or Variables

=>We know that all types of Literals are stored in main memory by allocating Sufficient amount of Memory with help of Data Types. To Process the Data / Literals stored in main memory, we must give distinct names to the created memory space and these distinct names makes us to identify the values and they are also called IDENTIFIERS.

=>During Program Execution IDENTIFIER Values can be changed / Varying and hence IDENTIFIERS are called VARIABLES.

=>Hence All Types of LITERALS Must Be stored in the form of VARIABLES.

=>In Python Programming All Variables are called Objects.

Definition of Variable

=>A Variable is one of the Identifier whose value(s) can be changed during Program execution.

=>In Programming Language to do any data processing, we must use Variables / Objects (Python).

Rules for Using Variables or Identifiers in Python Program

=>To Use Variables in Python Programming, We use the following Rules.

1. A Variable Name is a combination of
Alphabets, Digits and a Special Symbol Under Score(_).
2. First Letter of the Variable Names must start with Alphabet or Special Symbol Under

Score (_)

Examples:

```
-----  
sal=23-----valid  
$sal=45----Invalid  
@name="python"---Invalid  
-sal=45----Invalid  
2sal=56---Invalid  
456=3.4---Invalid  
_sal_=45--valid  
_=56---valid  
__=5.6--valid  
--=45---invalid
```

3. Within the variable name, No special symbols are allowed except Under Score (_)

Examples:

```
tot sal=45----Invalid  
tot_marks=456--valid  
tot#sal=56----NameError
```

4. No Keywords to be used as Variable Names (bcoz Keywords are the Reserved Words and they give special Meaning to the compilers) .

Example:

```
-----  
if=45-----Invalid  
else=67---invalid  
for=4.5---Invalid  
if1=56--Valid  
_else=67--valid  
_for_=5.6--valid
```

Note:All Class Name can be used as Variable Names bcoz Class Names are not Keywords

5. All Variable Name are Case Sensitive

Examples:

```
-----  
>>> age=99-----Valid  
>>> AGE=98-----Valid  
>>> Age=97-----Valid  
>>> aGe=96-----Valid  
>>> print(age,AGE,Age,aGe)---- 99 98 97 96  
>>> a=12  
>>> A=13  
>>> print(a,A)----- 12 13
```

Data Types in Python

=>The Purpose of Data Types in Python is that " To allocate Sufficient amount of memory space for storing inputs in main memory of computer".

=>In Python Programming, We have 14 Data Types and They are Classified into 6 types.

I. Fundamental Category Data Types

1. int
 2. float
 3. bool
 4. complex
-

II.Sequence Category Data Types

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

III. List Category Data Types (Collections Data Types or Data Structures)

1. list
 2. tuple
-

IV. Set Category Data Types (Collections Data Types or Data Structures)

1. set
 2. frozenset
-

VI. Dict Category Data Types (Collections Data Types or Data Structures)

1. dict
-

VI. NoneType Category Data Types

1. NoneType
-

I. Fundamental Category Data Types

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

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

1. int

-
- 2. float
 - 3. bool
 - 4. complex
-

=====

1. int

=====

Properties

=>'int' is one of the pre-defined class and treated as Fundamental Data Type.
=>The purpose of int data type is that " To store Integer Data or Whole Numbers or Integral Values(Numbers or digits without decimal Places) and Different Number System data".

Examples:

Python Instructions	Output
>>> a=100	
>>> b=123	
>>> c=a+b	
>>> print(a,type(a))-----	100 <class 'int'>
>>> print(b,type(b))-----	123 <class 'int'>
>>> print(c,type(c))-----	223 <class 'int'>

=>with int data type we can also Store Different types of Number Systems Values.
=>In Programming languages, we have 4 Types of Number Systems. They are

- 1. Decimal Number System (default)
 - 2. Binary Number System
 - 3. Octal Number System
 - 4. Hexa Decimal Number System
-

1. Decimal Number System (default)

=>This is one of the default number System.

=>This Number System Contains

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

=>All Base 10 Literals are called Integer Data.

=>By default python Execution Environment always displays the result in the form decimal number System.

2. Binary Number System

=>This Number System Contains

Digits: 0 1 -----Total Digits =2
Base : 2

=>All Base 2 Literals are called Binary Data.

=>To Store Binary Data in python environment, The Binary Data Must be preceded by a letter 'b' or 'B'

=>Syntax: varname=0b Binary data

(OR)

varname=0B Binary data

=>When we store the Binary data in python environment,python Execution Environment converts automatically into decimal number System data.

Examples:

```
>>> a=0b1010
>>> print(a,type(a))-----10 <class 'int'>
>>> bin(10)-----'0b1010'
>>> a=0B1111
>>> print(a,type(a))-----15 <class 'int'>
>>> a=0b10120-----SyntaxError: invalid digit '2' in binary literal
```

3. Octal Number System

=>This Number System Contains

Digits: 0 1 2 3 4 5 6 7 -----Total Digits =8
Base : 8

=>All Base 8 Literals are called Octal Data.

=>To Store Octal Data in python environment, The Octal Data Must be preceded by a letter 'o' or 'O'

=>Syntax: varname=0o Octal data

(OR)

varname=0O Octal data

=>When we store the Octal data in python environment,python Execution Environment converts automatically into decimal number System data.

Examples:

```
>>> a=0o27
>>> print(a,type(a))-----23 <class 'int'>
>>> oct(a)-----'0o27'
>>> a=0O123
>>> print(a,type(a))-----83 <class 'int'>
>>> oct(83)-----'0o123'
>>> a=0o148-----SyntaxError: invalid digit '8' in octal literal
```

4. Hexa Decimal Number System

=>This Number System Contains

Digits: 0 1 2 3 4 5 6 7 8 9
 A(10) B(11) C(12) D(13) E(14) F(15) -----Total Digits
 =16

Base : 8

=>All Base 16 Literals are called Hexa Decimal Data.

=>To Store Hexa Decimal Data in python environment, The Hexa Decimal Data Must be preceded by a letter 'x' or 'X'

=>Syntax: varname=0x Hexa Decimal data

(OR)

varname=0X Hexa Decimal data

=>When we store the Hexa Decimal data in python environment, python Execution Environment converts automatically into decimal number System data.

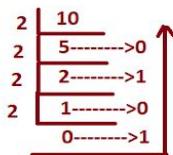
Examples:

```
>>> a=0xAC
>>> print(a,type(a))-----172 <class 'int'>
>>> hex(172)-----'0xac'
>>> a=0xBEE
>>> print(a,type(a))-----3054 <class 'int'>
>>> hex(3054)-----'0xbbe'
>>> a=0xFacE
>>> print(a,type(a))-----64206 <class 'int'>
>>> a=0xBEER-----SyntaxError: invalid hexadecimal literal
```

Conversion from Decimal Data Binary Data

Q1) Convert $(10)_{10} \rightarrow (x)_2$ here $x=1010$

Sol:-



hence $(10)_{10} \rightarrow (1010)_2$

Conversion from Binary to Decimal Data

Q1) Convert $(1010)_2 \rightarrow (x)_{10}$ $x=10$

Sol:-

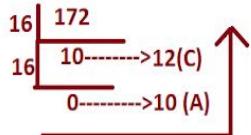
$$\begin{array}{r}
 1 \quad 0 \quad 1 \quad 0 \\
 3 \quad 2 \quad 1 \quad 2 \\
 \Rightarrow 2 \quad 2 \quad 2 \quad 2 \\
 \Rightarrow 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 \\
 \Rightarrow 8 + 0 + 2 + 0 \\
 \Rightarrow 10
 \end{array}$$

hence $(1010)_2 \rightarrow (10)_{10}$

Conversion from Decimal to Hexa Decimal

Q1) Convert $(172)_{10} \rightarrow (x)_{16}$ $x=AC$

Sol:



Hence $(172)_{10} \rightarrow (AC)_{16}$

Conversion from Hexa Decimal to Decimal

Q1) Convert $(AC)_{16} \rightarrow (x)_{10}$

Sol:

A	C
1	0
16	16

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

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

$\Rightarrow 160 + 12$

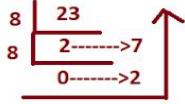
$\Rightarrow 172$

hence $(AC)_{16} \rightarrow (172)_{10}$

Conversion from Decimal Data into Octal Data

Q1) Convert $(23)_{10} \rightarrow (x)_{8}$ here $x=27$

Sol:-



Hence $(23)_{10} \rightarrow (27)_{8}$

Conversion from Octal Data into Decimal Data

Q1) Convert $(27)_{8} \rightarrow (x)_{10}$ $x=23$

Sol:

2	7
1	0
8	8

 $= 2 \times 8 + 7 \times 1$

$= 16 + 7$

$= 23$

Hence $(27)_{8} \rightarrow (23)_{10}$

2. float

Properties:

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

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

=>Example: Percentage of Marks, Taxable income for Financial year 22-23..etc

=>float data type can stores the data which belongs to Scientific Notation.

=>The Advantage of Scientific Notation is that " It Takes Less Memory Space for Extremly Large Floting Point Values."

=>float data type does not support to store directly the values of Binary, Ocral and Hexa Decimal Number Systems. But it allows to store only Deciaml Number System Values (Default)

Examples:

```
>>> a=12.34
>>> print(a,type(a))-----12.34 <class 'float'>
>>> a=10
>>> b=2.3
>>> c=a+b
>>> print(a,type(a))-----10 <class 'int'>
>>> print(b,type(b))-----2.3 <class 'float'>
>>> print(c,type(c))-----12.3 <class 'float'>
```

```
>>> a=0b1010.0b1010-----SyntaxError: invalid decimal literal  
>>> a=0b1010.34-----SyntaxError: invalid syntax  
>>> a=0xACC.0b1010-----SyntaxError: invalid decimal literal  
>>> a=0o23.0o45-----SyntaxError: invalid decimal literal
```

3. bool

Properties

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

=>The purpose of bool data type is that "To Store True and False Values".
=>In Python Programming , True and Fase are of the KeyWords and They are the values for
bool data type.
=>Internally, The value of True is 1 and the value of False is 0

Examples:

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

>>> a=True
>>> print(a,type(a))-----True <class 'bool'>
>>> b=False
>>> print(b,type(b))-----False <class 'bool'>
>>> print(a+b)-----1
>>> print(False+False)-----0
>>> print(False-True)-----1
>>> print(True+True+False)-----2
>>> print(2*True-4+False)-----2
>>> print(0b1010+True+1)-----12
>>> print(0b1111*False+2*True)----2
>>> print(0b100*2+3*True)-----11

>>> print(True>False)-----True
>>> print(True>1)-----False
>>> print(True>=0b00000001)-----True
>>> print(False>=0.0)-----True
>>> print(True*False>True)-----False
```

4. complex

=>Properties

=>'complex' is one of the pre-defined class and treated as Fundamental Data Type.
=>The purpose of complex data type is that " To Store and Process Complex Data".
=>The Generale Notation of Complex Data Type is shown bellow.

$$a+bj \text{ or } a-bj$$

=>here 'a' is called Real Part

=>here 'b' is called Imginary Part

=>Here 'j' represents $\sqrt{-1}$

=>Internally , The real and imginary parts are by default belongs to float.

=>To Extract or get Real Part from Complex object, we use a pre-defined attribute called "real"
Syntax:- Complexobj.real

=>To Extract or get Imaginary Part from Complex object, we use a pre-defined attribute called "imag"
Syntax:- Complexobj.imag

Examples:

```
>>> a=2+3j
>>> print(a,type(a))-----(2+3j) <class 'complex'>
>>> b=4-5j
>>> print(b,type(b))-----(-4+5j) <class 'complex'>
>>> c=-2-4j
>>> print(c,type(c))-----(-2-4j) <class 'complex'>
-----
>>> a=1.4+3.4j
>>> print(a,type(a))-----(1.4+3.4j) <class 'complex'>
>>> b=-3.5-5.6j
>>> print(b,type(b))-----(-3.5-5.6j) <class 'complex'>
>>> c=10+2.3j
>>> print(c,type(c))-----(10+2.3j) <class 'complex'>
-----
>>> a=0+2j
>>> print(a,type(a))-----2j <class 'complex'>
>>> b=9.5j
>>> print(b,type(b))-----9.5j <class 'complex'>
-----
>>> a=10.4+3j
>>> print(a,type(a))-----(10.4+3j) <class 'complex'>
>>> a.real-----10.4
>>> a.imag-----3.0
>>> a=9.5j
>>> print(a,type(a))-----9.5j <class 'complex'>
>>> a.real-----0.0
>>> a.imag-----9.5
>>> a.imaginary-----AttributeError: 'complex' object has no attribute 'imagiary'
-----
>>> a=-3.4-4.5j
>>> print(a,type(a))-----(-3.4-4.5j) <class 'complex'>
>>> a.real----- -3.4
>>> a.imag----- -4.5
-----
>>> (12+4j).real-----12.0
>>> (12+4j).imag-----4.0
>>> (-0-2.3j).real-----0.0
>>> (-0-2.3j).imag-----2.3
```

```
>>> (0b1111+0b1010j).real-----SyntaxError: invalid binary literal  
>>> (0b1111+0b1010j).imag-----SyntaxError: invalid binary literal  
=====X=====
```

II.Sequence Category Data Types

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

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

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

1. str (Part-1)

=>"str" is one of the pre-defined class and treated as Sequence Data Type,

=>The purpose of str data type is that " To store Text Data or Numeric Data or Alpha-numeric Data and special symbols enclosed within Single or Double Quotes or Tripple Single or Tripple Double Quotes."

=>Def. of str (String):

=>A String is sequence or Collection of Characters or Numbers or Alpha-numeric Data and special symbols enclosed within Single or Double Quotes or Triple Single or Triple Double Quotes.

Types of str data

=>In Python Programming, We have 2 types of str data. They are

1. Single Line String Data
 2. Multi Line String Data

1. Single Line String Data

=>Syntax: " String Data "

(OR)
'String data'

=>Single Line string data always enclosed within Double or Single Quotes.

=>Double or Single Quotes are not useful for organizing Multi Line String Data.

2. Multi Line String Data

```
=>Syntax:     " " "
                  String Line 1
                  String Line 2
-----
                  String Line n " "
                  (OR)
      ''''      String Line 1
                  String Line 2
-----
                  String Line n '''
```

=>With Tripple Double Quotes or Tripple Single Quotes we can organize Multi Line String data and also we can organize Single Line String data.

Examples:

```
>>> s1="Guido Van Rossum"
>>> print(s1,type(s1))-----Guido Van Rossum <class 'str'>
>>> s2="123456"
>>> print(s2,type(s2))-----123456 <class 'str'>
>>> s2="Python3.10.6"
>>> print(s2,type(s2))-----Python3.10.6 <class 'str'>
>>> s3='Travis Oliphant'
>>> print(s3,type(s3))-----Travis Oliphant <class 'str'>
>>> s4='1234python%$'
>>> print(s4,type(s4))-----1234python%$ <class 'str'>
>>> s5='A'
>>> print(s5,type(s5))-----A <class 'str'>
>>> s6='6'
>>> print(s6,type(s6))-----6 <class 'str'>
>>> s7='$%^&@'
>>> print(s7,type(s7))-----$%^&@ <class 'str'>
-----
>>> s1="Python Programming"
>>> print(s1,type(s1))-----Python Programming <class 'str'>
>>> s2='Python Programming'
>>> print(s2,type(s2))-----Python Programming <class 'str'>
-----
>>> addr1="Guido van Rossum
                                         SyntaxError: unterminated string literal
(detected at line 1)
```

```
>>> addr1='Guido van Rossum'                                     SyntaxError: unterminated string literal  
(detected at line 1)  
-----  
>>> addr1="""Guido Van Rossum  
... FNO:3-4, Red Sea Side  
... Python Software Foundation  
... Nether Lands  
... Pin-57 """  
>>> print(addr1,type(addr1))  
                                         Guido Van Rossum  
                                         FNO:3-4, Red Sea Side  
                                         Python Software Foundation  
                                         Nether Lands  
                                         Pin-57 <class 'str'>  
-----  
>>> addr2="""Travis Oliphant  
... Numpy Organization  
... FNO-34-56 Nether lands  
... PIN-45 ""  
>>> print(addr2,type(addr2))  
                                         Travis Oliphant  
                                         Numpy Organization  
                                         FNO-34-56 Nether lands  
                                         PIN-45 <class 'str'>  
-----  
>>> s1="""Python Programming"""  
>>> print(s1,type(s1))-----Python Programming <class 'str'>  
>>> s1="Python Programming"  
>>> print(s1,type(s1))-----Python Programming <class 'str'>  
>>> s2="""K"""  
>>> print(s2,type(s2))-----K <class 'str'>  
>>> s2="K"  
>>> print(s2,type(s2))-----K <class 'str'>
```

Operations on str data (Part-1)

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

1. Indexing Operation
2. Slicing Operations

1. Indexing Operation

=>The Process of Obtaining Single Character from given str object by passing valid Index is called Indexing.

=>Syntax:

----- strobj [Index]

=>Here strobj is an object of <class, 'str'>

=>Index can be either +Ve Indexing or -Ve Indexing

=>If we enter valid Index value then we get Corresponding Character from strobj.

=>If we enter invalid Index value then we get IndexError.

----- Examples:

```
>>> s="PYTHON"
>>> print(s[3])-----H
>>> print(s[-2])-----O
>>> print(s[4])-----O
>>> print(s[-6])-----P
>>> print(s[0])-----P
>>> print(s[-5])-----Y
>>> print(s[-1])-----N
>>> print(s[3])-----H
>>> print(s[-3])-----H
>>> print(s[-13])-----IndexError: string index out of range
```

```
>>> s="PYTHON"
>>> print(s,type(s))-----PYTHON <class 'str'>
>>> len(s)-----6
>>> s="Python Prog"
>>> len(s)----- 11
>>> print(s[34])-----IndexError: string index out of range
```

2. Slicing Operations

=>The Process of obtaining Range of Characters or Sub String from given Str object is called Slicing .

=>Slicing Operation can performed by using 5 Syntaxes.-

Syntax-1: Strobject[Begin Index : End Index]

This Syntax obtains range of characters from BeginIndex to EndIndex-1 provided Begin Index<End Index Otherwise we never get any output (' ')

----- Examples

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

```

>>> print(s,type(s))-----PYTHON <class 'str'>
>>> print( s[0:4] )-----PYTH
>>> print( s[4:0] )----- Empty
>>> s[4:0]----- ''
>>> print( s[2:5] )----- THO
>>> print( s[0:6] )-----PYTHON
>>> print( s )-----PYTHON
>>> print(s[-6:-3])-----PYT
>>> print(s[-4:-1])-----THO
>>> print(s[-1:-6])----- Empty
>>> print(s[2:6])-----THON
>>> print(s[2:-2])----- TH ( Most Imp )
>>> print(s[1:-1])----- YTHO
>>> print(s[-1:-6])----- empty
>>> s[-1:-6]----- ''
>>> s[2:-1]-----'THO'
>>> s[-6:4]-----'PYTH'
>>> s[2:-4]----- ' ' ( Empty String )

```

Syntax-2: StrObj[BeginIndex :]

=>In This Syntax We specified Begin Index and Did't not specify End Index.
=>If we don't Specify End Index then PVM always takes End Character Index as End Index OR
len(strobj)-1

Examples:

```

>>> s="PYTHON"
>>> print(s,type(s))-----PYTHON <class 'str'>
>>> s[2:]-----'THON'
>>> s[1: ]-----'YTHON'
>>> s[0: ]-----'PYTHON'
>>> s[-4: ]-----'THON'
>>> s[-6: ]-----'PYTHON'
>>> s[-3: ]-----'HON'

```

Syntax-3: StrObj[: EndIndex]

=>In This Syntax We specified End Index and Did't not specify Begin Index.
=>If we don't Specify Begin Index then PVM always takes First Character Index as Begin Index.

Examples:

```

>>> s="PYTHON"
>>> print(s,type(s))-----PYTHON <class 'str'>

```

```
>>> s[:4]-----'PYTH'  
>>> s[:3]-----'PYT'  
>>> s[:6]-----'PYTHON'  
>>> s[:-4]-----'PY'  
>>> s[:-5]-----'P'  
>>> s[:-3]-----'PYT'  
>>> s[:0]-----' empty  
>>> s[:-6]-----' empty
```

Syntax-4: StrObj[:]

=> In This Syntax We Didn't not specify Begin Index and End Index.

=> If we don't Specify Begin Index then PVM always takes First Character Index as Begin Index and If we don't Specify End Index then PVM always takes Last Character Index as End Index (OR) len(strobj)-1 as End Index.

Examples:

```
>>> s="PYTHON"  
>>> print(s,type(s))-----PYTHON <class 'str'>  
>>> s[:]-----'PYTHON'  
>>> s[0:]-----'PYTHON'  
>>> s[:-6]-----' ' Empty  
>>> s[:6]-----'PYTHON'  
>>> s[-6:]-----'PYTHON'  
>>> s[:-5]-----'P'  
>>> s[:-4]-----'PY'  
>>> s[-3:]-----'HON'  
>>> s[-6:6]-----'PYTHON'
```

Most IMP:

```
--  
>>> s="PYTHON"  
>>> print(s,type(s))-----PYTHON <class 'str'>  
>>> s[-13:-6]-----'  
>>> s[-13:6]-----'PYTHON'  
>>> s[0:123]-----'PYTHON'  
>>> s[-123:345]-----'PYTHON'
```

NOTE:- All the Above Syntaxes are obtaining Range of Characters In Forward Direction.

Syntax-5 : Strobj[BeginIndex :End Index :Step]

Rules:

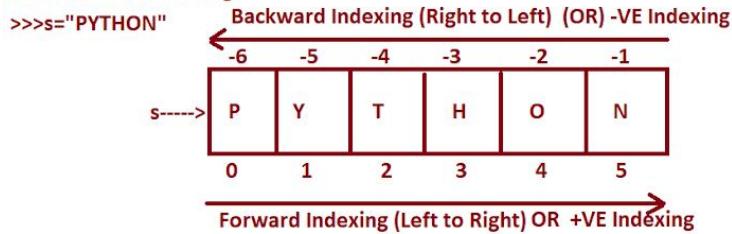
- 1) Here BeginIndex , End Index and Step can either +VE INDEX and -VE INDEX
- 2) If the value of STEP is +VE then PVM takes the Range of Characters from Begin Index to End Index-1 in Forward Direction provided Begin Index<End Index otherwise we get empty String(')
- 3) if the value of STEP is -VE then PVM Takes Range of Characters from BeginIndex to End Index+1 in Backward Direction provided Begin Index > End Index
- 4) When we are retrieving the data in forward Direction if the EndIndex Value is 0 then we never get any result / outout.
- 5) When we are retrieving the data in backward Direction if the EndIndex Value is -1 then we never get any result / outout.

Examples:

```
-----
>>> s="PYTHON"
>>> print(s,type(s))-----PYTHON <class 'str'>
>>> s[0:6:1]-----'PYTHON'
>>> s[0:6:2]-----'PTO'
>>> s[2:4:1]-----'TH'
>>> s[-6: :1]-----'PYTHON'
>>> s[:6:1]-----'PYTHON'
>>> s[:-2:2]-----'PT'
-----
>>> s[6:2:2]-----''
-----
>>> s="PYTHON"
>>> print(s,type(s))-----PYTHON <class 'str'>
>>> s[0:6:2]-----'PTO'
>>> s[0:6:-2]-----' '
>>> s[5:0:-1]-----'NOHTY'
>>> s[5: :-1]-----'NOHTYP'
>>> s[-1:-7:-1]-----'NOHTYP'
>>> s[-1:-7:-2]-----'NHY'
>>> s[::-1]-----'NOHTYP'
-----
>>> s="MADAM"
>>> s==s[::-1]-----True
>>> s="LIRIL"
>>> s[::-1]==s[::1]-----True
>>> "MALAYALAM"=="MALAYALAM"[::-1] -----True
>>> "RACECAR"[::-1]== "RACECAR"[::-1] -----True
>>> "PYTHON"=="PYTHON"[::-1]-----False
```

```
>>> print("KVR"[::3])-----K
>>> "KVR"[::3]=="KVR"[:-1][-1]-----True
-----
>>> s="PYTHON"
>>> print(s)-----PYTHON
>>> s="PYTHON PROG"
>>> s[::-1]-----'GORP NOHTYP'
>>> s="121"
>>> s==s[::-1]-----True
>>> "8558"=="8558"[:-1]-----True
-----
>>> s="PYTHON"
>>> print(s)
PYTHON
>>> s[2:-1:1]-----'THO'
>>> s[2:0:1]-----' ' (Rule-5)
>>> s[1:0:2]-----' '
>>> s[-6:-1:-1]----- ' ' (Rule-6)
>>> s[-3:-1:-2]-----' '
-----
-----
>>> s="PYTHON"
>>> print(s)-----PYTHON
>>> s[-6:6:-2]-----''
>>> s[2:-1:-2]-----' '
>>> s[1:-1:3]-----'YO'
>>> s[1:-1:3]-----' '
>>> s[1::-3]-----'Y'
>>> s[-2::-2]-----OTP'
>>> s[-2::-2][::-1]-----'PTO'
```

Consider the following



Type Casting Techniques in Python

=>The Process of Converting One Type of Possible Value into Another Type of Value is called Type Casting.

=>Fundamentally, we have 5 types of Type Casting Techniques. They are

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

1. int()

=>int() is used converting any Possible Type of Value into int type Value

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

Examples: float into int-->Possible

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

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

Examples: bool into int-->Possible

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

Examples:complex into int-->Not Possible

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

Examples:

Case-1: Str int-->int-->Possible

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

Case-2: Str float-->int-->Not Possible

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

Case-3: Str bool--> int-->Not Possible

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

Case-4: str complex-->int-->Not Possible

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

Case-5---Pure Str-->int-->Not Possible

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

2. float()

=>float() is used converting any Possible Type of Value into float type Value

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

Example: int---->float-->Possible

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

Example: bool---->float-->Possible

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

Example: complex---->float-->Not Possible

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

```
>>> print(b,type(b))-----4.5 <class 'float'>
```

Example:

Case-1 str int---->float -->Possible

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

Case-2 str float---->float -->Possible

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

Case-3 str bool---->float -->Not Possible

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

Case-4 str complex---->float -->Not Possible

```
>>> a="2+3.5j"  
>>> print(a,type(a))-----2+3.5j <class 'str'>  
>>> b=float(a)-----ValueError: could not convert string to float:  
'2+3.5j'
```

Case-5 Pure str ---->float -->Not Possible

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

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

=>bool() is used converting any Possible Type of Value into bool type Value

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

=>ALL NON-ZERO VALUES ARE TREATED AS TRUE

=>ALL ZERO VALUES ARE TREATED AS FALSE

Example: int---->bool---->Possible

```
>>> a=10
>>> print(a,type(a))-----10 <class 'int'>
>>> b=bool(a)
>>> print(b,type(b))-----True <class 'bool'>
>>> a=-123
>>> print(a,type(a))-----123 <class 'int'>
>>> b=bool(a)
>>> print(b,type(b))-----True <class 'bool'>
>>> a=0
>>> print(a,type(a))-----0 <class 'int'>
>>> b=bool(a)
>>> print(b,type(b))-----False <class 'bool'>
```

Example: float---->bool---->Possible

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

Example: complex---->bool--->Possible

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

Example: Str int,float,complex,bool and pure str are possible to convert into bool type
Here bool type True provided len(strobj) is >0
Here bool type False provided len(strobj) is ==0

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

>>> a="False"
>>> print(a,type(a))
False <class 'str'>
>>> b=bool(a)
>>> print(b,type(b))
True <class 'bool'>
>>> a="12.34"
```

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

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

complex()

=>complex() is used converting any Possible Type of Value into complex type Value

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

Examples: int---->complex---->Possible

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

Examples: float---->complex---->Possible

```
>>> a=2.3
>>> print(a,type(a))-----2.3 <class 'float'>
>>> b=complex(a)
```

```
>>> print(b,type(b))-----(2.3+0j) <class 'complex'>
```

Examples: bool---->complex--->Possible

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

Examples:

```
>>> a="12" # str int----complex--Possible  
>>> print(a,type(a))-----12 <class 'str'>  
>>> b=complex(a)  
>>> print(b,type(b))-----(12+0j) <class 'complex'>  
  
>>> a="12.45" #str float----complex---->Possible  
>>> print(a,type(a))-----12.45 <class 'str'>  
>>> b=complex(a)  
>>> print(b,type(b))-----(12.45+0j) <class 'complex'>  
  
>>> a="True" # str bool---->complex---Not Possible  
>>> print(a,type(a))-----True <class 'str'>  
>>> b=complex(a)-----ValueError: complex() arg is a malformed string  
  
>>> a="KVR-PYTHON" # Pure Str----->Complex--Not Possible  
>>> print(a,type(a))-----KVR-PYTHON <class 'str'>  
>>> b=complex(a)-----ValueError: complex() arg is a malformed string
```

===== 5. str() =====

=>str() is used converting all types of values into str type value.

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

Examples:

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

```

>>> print(b,type(b))-----123 <class 'str'>
>>> b-----'123'
>>> a=12.34
>>> print(a,type(a))-----12.34 <class 'float'>
>>> b=str(a)
>>> print(b,type(b))-----12.34 <class 'str'>
>>> b-----'12.34'

>>> a=True
>>> print(a,type(a))-----True <class 'bool'>
>>> b=str(a)
>>> print(b,type(b))-----True <class 'str'>
>>> b-----'True'
>>> a=2+3.5j
>>> print(a,type(a))-----(2+3.5j) <class 'complex'>
>>> b=str(a)
>>> print(b,type(b))-----(2+3.5j) <class 'str'>
>>> b-----'(2+3.5j)'
=====X=====
==

=====2. bytes
=====
```

Properties:

- =>"bytes" if one of the pre-defined class and treated as Sequence Data Type.
- =>Cipher Text
- =>The Internal Implementation of bytes data type is that "End-to-End Encryption (OR) Cipher Text (OR) Encrypted Data" of Normal Text.
- =>The bytes data stores the data in the range of 0 to 256 (It stores from 0 to 255 (256-1) only)
- =>bytes data type does not contains any symbolic notation but we can convert other type of values into bytes type values by using bytes().
- =>Syntax: varname=bytes(object)
- =>An object of bytes belongs to Immutable bcoz bytes' object does not support item assignment
- =>An object of bytes data type supports Both Indexing and Slicing Operations.
- =>An object of bytes maintains Insertion Order (i.e Which ever order we insert the data in the same order Value will display)

Examples:

```

>>> l1=[10,20,30,40,256]
>>> print(l1,type(l1))-----[10, 20, 30, 40, 256] <class 'list'>
>>> b=bytes(l1)-----ValueError: bytes must be in range(0, 256)
>>> l1=[10,0,-20,30,40,255]
>>> print(l1,type(l1))-----[10, 0, -20, 30, 40, 255] <class 'list'>
```

```
>>> b=bytes(11)-----ValueError: bytes must be in range(0, 256)

>>> l1=[10,0,30,40,255]
>>> print(l1,type(l1))-----[10, 0, 30, 40, 255] <class 'list'>
>>> b=bytes(l1)
>>> print(b, type(b),id(b))-----b'\n\x00\x1e(\xff <class 'bytes'> 2043775384912
>>> b[-1]-----255
>>> b[0]-----10
>>> b[0]=123-----TypeError: 'bytes' object does not support item assignment
-----
>>> l1=[10,0,30,40,255]
>>> print(l1,type(l1))-----[10, 0, 30, 40, 255] <class 'list'>
>>> b=bytes(l1)
>>> print(b, type(b),id(b))-----b'\n\x00\x1e(\xff <class 'bytes'> 2043775382752
>>> for kvr in b:
...     print(kvr)
...
10
0
30
40
255

>>> t1=(10,20,30,10,255,45)
>>> print(t1,type(t1))-----(10, 20, 30, 10, 255, 45) <class 'tuple'>
>>> b=bytes(t1)
>>> print(b,type(b),id(b))-----b'\n\x14\x1e\n\xff-' <class 'bytes'> 2043775382800
>>> for v in b:
...     print(v)
...
10
20
30
10
255
45

>>> b[0:4]-----b'\n\x14\x1e\n'
>>> for v in b[0:4]:
...     print(v)
...
10
20
30
10

>>> for v in b[::-1]:
...     print(v)
...
...
```

```
45  
255  
10  
30  
20  
10
```

=====X=====

===== Mutable and Immutable =====

=>A Mutable object is one, whose content can be changed at Same Memory Address.

=>Examples: list, bytearray, set, dict

=>An immutable object is one, which will satisfy the following Properties

a) The value immutable object can't be changed at Same Memory Address (OR) In otherwords, Value of Immutable object can be changed and place the modified Value in New Memory Address by eliminating Old Memory Address by Garbage Collector.

b) Immutable objects does not support Item Assignment.

Examples: int, float, bool, complex, str, bytes, range, tuple, set, frozenset, etc

=====X=====

=

===== 3. bytearray =====

Properties:

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

=>The Internal Implementation of bytearray data type is that "End-to-End Encryption (OR) Cipher Text (OR) Encrypted Data" of Normal Text.

=>The bytearray data stores the data in the range of 0 to 256 (It stores from 0 to 255 (256-1) only)

=>bytearray data type does not contain any symbolic notation but we can convert other type of values into bytearray type values by using bytearray().

=>Syntax: varname=bytearray(object)

=>An object of bytearray belongs to Mutable bcoz bytearray object supports item assignment

=>An object of bytearray data type supports Both Indexing and Slicing Operations.

=>An object of bytearray maintains Insertion Order (i.e Which ever order we insert the data in the same order Value will display)

=>NOTE:- The Functionality of bytearray is exactly similar to bytes but an object of bytes belongs to immutable where as an object of bytearray is mutable.

=====

Examples:

```
>>> l1=[10,20,30,40,0,256]
>>> print(l1,type(l1))-----[10, 20, 30, 40, 0, 256] <class 'list'>
>>> b=bytearray(l1)-----ValueError: byte must be in range(0, 256)
>>> l1=[10,-20,30,40,0,255]
>>> print(l1,type(l1))-----[10, -20, 30, 40, 0, 255] <class 'list'>
>>> b=bytearray(l1)-----ValueError: byte must be in range(0, 256)

>>> l1=[10,20,30,40,0,255]
>>> print(l1,type(l1))-----[10, 20, 30, 40, 0, 255] <class 'list'>
>>> b=bytearray(l1)
>>> print(b,type(b),id(b))---bytearray(b'\n\x14\x1e(\x00\xff') <class 'bytearray'>
                                         2376795361136
>>> for k in b:
...     print(k)
...
10
20
30
40
0
255
>>> b[0]=120 # Item Assignment--Possible--Mutable
>>> for k in b:
...     print(k)
...
120
20
30
40
0
255
>>> print(b,type(b),id(b))---bytearray(b'\x14\x1e(\x00\xff') <class 'bytearray'> 2376795361136
>>> for k in b:
...     print(k)
...
120
20
30
40
0
255
>>> b[1]-----20
>>> b[1:4]-----bytearray(b'\x14\x1e(')
>>> for k in b[1:4]:
```

```

...     print(k)
...
20
30
40
>>> for k in b[::-1]:
...     print(k)
...
255
0
40
30
20
120
=====
X=====
=
=====
4. range
=====
```

Properties

- =>"range" is one of the pre-defined class and treated as Sequence Data Type
 - =>The purpose of range data type is that "To store or generate Sequence of Numerical Integer Values by maintaining Equal Interval of Value."
 - =>On the object of range data type, we can perform Both Indexing and Slicing Operations
 - =>An object of range belongs to immutable.
 - =>An object of range maintains Insertion Order.
 - =>To store or generate Sequence of Numerical Integer Values by maintaining Equal Interval of Value, range data type provides 3 Syntaxes. They are.
-

- =>Syntax-1: varname=range(Value)
 - =>This Syntax generates Range of Values from 0 to Value-1
- Examples:
-

```

>>> r=range(10)
>>> print(r,type(r))-----range(0, 10) <class 'range'>
>>> for v in r:
...     print(v)
...
0
1
2
3
4
5
6
```

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

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

=>This generates Range of Values from Begin to End-1

Examples:

```
>>> r=range(10,16)  
>>> print(r,type(r))-----range(10, 16) <class 'range'>  
>>> for v in r:  
...     print(v)  
...  
10  
11  
12  
13  
14  
15
```

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

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

=>NOTE: In the above Two Syntaxes, the default STEP is 1

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

=>This generates Range of Values from Begin to End-1 by maintaining Step as Equal Interval.

Examples:

```
>>> r=range(10,21,3)
>>> print(r,type(r))-----range(10, 21, 3) <class 'range'>
>>> for v in r:
...     print(v)
...
10
13
16
19
>>> for v in range(2,21,2):
...     print(v)
...
2
4
6
8
10
12
14
16
18
20
>>> for v in range(1,21,2):
...     print(v)
...
1
3
5
7
9
11
13
15
17
19
```

Programming Examples:

Q1) 0 1 2 3 4 5 6 7 8 9 -----range(10)

```
>>> for v in range(10):
...     print(v)
...
0
1
2
```

```
3  
4  
5  
6  
7  
8  
9
```

Q2) 10 11 12 13 14 15---range(10,16)

```
>>> for v in range(10,16):
```

```
...     print(v)
```

```
...  
10  
11  
12  
13  
14  
15
```

Q3) 300 301 302 303 304 305---range(300,306)

```
>>> for v in range(300,306):
```

```
...     print(v)
```

```
...  
300  
301  
302  
303  
304  
305
```

Q4) 10 9 8 7 6 5 4 3 2 1----range(10,0,-1)

```
>>> for v in range(10,0,-1):
```

```
...     print(v)
```

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

Q5) -10 -11 -12 -13 -14 -15----range(-10,-16,-1)

```
>>> for v in range(-10,-16,-1):  
...     print(v)  
...
```

```
-10  
-11  
-12  
-13  
-14  
-15
```

Q6) 100 110 120 130 140 150--range(100,151,10)

```
>>> for k in range(100,151,10):  
...     print(k)
```

```
...  
100  
110  
120  
130  
140  
150
```

```
>>>
```

Q7) 1000 900 800 700 600 500----range(1000,499,-100)

```
>>> for v in range(1000,499,-100):  
...     print(v)
```

```
...  
1000  
900  
800  
700  
600  
500
```

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

```
>>> for v in range(-5,6,1):  
...     print(v)
```

```
...  
-5  
-4  
-3  
-2  
-1
```

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

```
>>> r=range(500,601,50)  
>>> r[0]  
500  
>>> r[1]  
550  
>>> r[-1]  
600  
>>> r[2]  
600  
>>> r=range(500,601,10)  
>>> r[-1]  
600  
>>> for v in r:  
...     print(v)  
...  
500  
510  
520  
530  
540  
550  
560  
570  
580  
590
```

```
600
>>> for v in r[5:]:
...     print(v)
...
550
560
570
580
590
600
>>> for v in r[5:][::-1]:
...     print(v)
...
600
590
580
570
560
550
-----
>>> r=range(500,601,10)
>>> print(r,type(r))
range(500, 601, 10) <class 'range'>
>>> r[0]
500
>>> r[1]
510
>>> r[2]
520
>>> r[1]=700-----TypeError: 'range' object does not support item assignment
-----
>>> print(range(50,60)[5])-----55
>>> for v in range(50,60)[5:7]:
...     print(v)
...
55
56
>>> for v in range(50,60)[::-2]:
...     print(v)
...
59
57
55
53
51
```

=====X=====

List Category Data Types (Collections Data Types or Data Structures)

=>The purpose of List Category Data Types in python is that " To Store Multiple Values either of Same Type or Different Type or Both the Types with Unique and Duplicate in single object."

=>We have two data type in List Category. They are

1. list (Mutable)
2. tuple (Immutable)

list

Index

=>Purpose of list

=>Operations on list

- 1) Indexing
- 2) slicing

=>Pre-Defined Functions in list

- 1) append()
- 2) insert()
- 3) remove()
- 4) pop(index)
- 5) pop()

Note: del operator

- 6) count()
- 7) index()
- 8) reverse()
- 9) sort()
- 10) extend()
- 11) copy()---- Shallow and Deep copy

=>Inner List / Nested List

=>Pre-defined Function in inner / nested list

Properties of list

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

=>The purpose of list data type is that "To Store Multiple Values either of Same Type or

Different Type or Both the Types with Unique and Duplicate in single object.
=>The Elements of list must written or Organized or stored within Square Brackets and the elements separated by Comma.
=>An object of list maintains Insertion Order.
=>On the object of list, we can perform both Indexing and Slicing Operations.
=>An object of list belongs to Mutable bcoz it allows us to update the values of list at same address.
=>We can convert any type of value into list type value by using list()
Syntax: listobj=list(object)
=>by using list data type, we can create two types of list objects. They are

- 1) empty list
 - 2) non-empty list
-

1) empty list

Syntax: varname=[]
(OR)
varname=list()

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

2) non-empty list

Syntax: varname=[Val1,Val2...Val-n]

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

Examples:

```
>>> l1=[10,20,30,10,40]
>>> print(l1,type(l1))-----[10, 20, 30, 10, 40] <class 'list'>
>>> l1=[111,"Rossum",34.56,True,"Python"]
>>> print(l1,type(l1))-----[111, 'Rossum', 34.56, True, 'Python'] <class 'list'>
>>> l1[0]-----111
>>> l1[-1]-----'Python'
>>> l1[0:3]-----[111, 'Rossum', 34.56]

>>> print(l1,type(l1))-----[111, 'Rossum', 34.56, True, 'Python'] <class 'list'>
>>> print(l1,type(l1),id(l1))---[111, 'Rossum', 34.56, True, 'Python'] <class 'list'>
2902431303872
>>> l1[0]=222
>>> print(l1,type(l1),id(l1))---[222, 'Rossum', 34.56, True, 'Python'] <class 'list'>
2902431303872
```

```

>>> l1=[]
>>> print(l1,type(l1))-----[] <class 'list'>
>>> len(l1)-----0
>>> l2=list()
>>> print(l2,type(l2))-----[] <class 'list'>
>>> len(l2)-----0
>>> l3=[10,"Rossum","PSF",3.4,True]
>>> print(l3,type(l3))-----[10, 'Rossum', 'PSF', 3.4, True] <class 'list'>
>>> len(l3)-----5
-----
>>> s="PYTHON"
>>> print(s,type(s))-----PYTHON <class 'str'>
>>> l1=list(s)
>>> print(l1,type(l1))-----['P', 'Y', 'T', 'H', 'O', 'N'] <class 'list'>
-----
>>> l1=[10,20,30,40]
>>> b=bytes(l1)
>>> print(b,type(b))-----b'\n\x14\x1e' <class 'bytes'>
>>> l2=list(b)
>>> print(l2,type(l2))-----[10, 20, 30, 40] <class 'list'>
-----
>>> l1=[10,20,30,40]
>>> l1[2]-----30
>>> l1[-2]-----30
>>> l1[::2]-----[10, 30]
>>> l1[::-1]-----[40, 30, 20, 10]
>>> l1[:]-----[10, 20, 30, 40]
=====X=====
=====
```

Pre-Defined Functions in list

=>Along with the operations on list like Indexing and Slicing, we can perform many more operations by using pre-defined function of list object.
=>The pre-defined functions of list are given bellow.

1) append():

=>Syntax: listobj.append(Value)
=>This Function is used for adding Value at the end of existing elements of list(known as appending)

Examples:

```

>>> l1=[10,"Rossum"]
>>> print(l1,type(l1),id(l1))-----[10, 'Rossum'] <class 'list'> 2902435500480
-----
```

```
>>> len(l1)-----2
>>> l1.append(23.45)
>>> print(l1,type(l1),id(l1))---[10, 'Rossum', 23.45] <class 'list'> 2902435500480
>>> l1.append("KVR")
>>> print(l1,type(l1),id(l1))---[10, 'Rossum', 23.45, 'KVR'] <class 'list'> 2902435500480
>>> l1.append(True)
>>> l1.append(2+3.5j)
>>> print(l1,type(l1),id(l1))---[10, 'Rossum', 23.45, 'KVR', True, (2+3.5j)] <class 'list'>
>>> len(l1)---6
```

```
>>> l1=[]
>>> print(l1,len(l1), id(l1))-----[] 0 2902435500544
>>> l1.append(10)
>>> l1.append("Raj")
>>> l1.append(10.34)
>>> l1.append("Hyd")
>>> print(l1,len(l1), id(l1))-----[10, 'Raj', 10.34, 'Hyd'] 4 2902435500544
```

2) insert()

=>Syntax:- listobj.insert(Index, Value)
=>Here Index can be either +Ve or -Ve
=>Value can be any type.
=>This Function is used for Inserting the Specific Value at specified index.

Examples:

```
>>> l1=[10,20,30,"Python","DJango",34.56]
>>> print(l1,id(l1))-----[10, 20, 30, 'Python', 'DJango', 34.56] 2902431529344
>>> l1.insert(3,"Rossum")
>>> print(l1,id(l1))-----[10, 20, 30, 'Rossum', 'Python', 'DJango', 34.56] 2902431529344
>>> l1[-3]="PYTH"
>>> print(l1,id(l1))-----[10, 20, 30, 'Rossum', 'PYTH', 'DJango', 34.56] 2902431529344
>>> l1.insert(1,234.99)
>>> print(l1,id(l1))-----[10, 234.99, 20, 30, 'Rossum', 'PYTH', 'DJango', 34.56] 2902431529344
```

```
>>> l1=list()
>>> print(l1,id(l1))-----[] 2902435501056
>>> l1.insert(0,"KVR")
>>> print(l1,id(l1))-----['KVR'] 2902435501056
>>> l1.insert(0,1111)
>>> print(l1,id(l1))-----[1111, 'KVR'] 2902435501056
>>> l1.insert(2,"HYD")
>>> print(l1,id(l1))-----[1111, 'KVR', 'HYD'] 2902435501056
```

```
>>> l1=[10,20,30]
>>> print(l1,id(l1))
[10, 20, 30] 2902435496128
>>> l1.append("Python")
>>> print(l1,id(l1))
[10, 20, 30, 'Python'] 2902435496128
>>> l1.insert(30,"Rossum") # Most IMP
>>> print(l1,id(l1))-----[10, 20, 30, 'Python', 'Rossum'] 2902435496128
```

3) remove() Based on Value

=>Syntax: listobj.remove(Value)
=>This Function is used for removing First Occurrence of The specific value from list object.
=>If the specific value does not exist in list object then we get ValueError
Examples:

```
>>> l1=[10,20,30,10,40,50,60]
>>> print(l1,id(l1))-----[10, 20, 30, 10, 40, 50, 60] 2902431529344
>>> l1.remove(20)
>>> print(l1,id(l1))-----[10, 30, 10, 40, 50, 60] 2902431529344
>>> l1.remove(10)
>>> print(l1,id(l1))-----[30, 10, 40, 50, 60] 2902431529344
>>> l1.remove(50)
>>> print(l1,id(l1))-----[30, 10, 40, 60] 2902431529344
>>> l1.remove(100)-----ValueError: list.remove(x): x not in list
```

```
>>> l1=[]
>>> l1.remove(3)-----ValueError: list.remove(x): x not in list
>>> list().remove(100)----ValueError: list.remove(x): x not in list
```

4) pop(index): Based Index

Syntax: listobj.pop(Index)
=>This Function is used for removing the element of listobj based Index.
=>If index value is invalid then we get IndexError

Examples:

```
>>> l1=[10,20,10,30,40,50,60,30]
>>> print(l1,id(l1))-----[10, 20, 10, 30, 40, 50, 60, 30] 2902435496128
>>> l1.pop(2)-----10
>>> print(l1,id(l1))-----[10, 20, 30, 40, 50, 60, 30] 2902435496128
>>> l1.pop(-1)-----30
```

```
>>> print(l1,id(l1))-----[10, 20, 30, 40, 50, 60] 2902435496128
>>> l1.pop(2)-----30
>>> print(l1,id(l1))-----[10, 20, 40, 50, 60] 2902435496128
-----
>>> list().pop(4)-----IndexError: pop from empty list
>>> [].pop(3)-----IndexError: pop from empty list
```

5) pop() :

=>Syntax:- list.pop()
=>This Function is used for Removing Last Element of List object
=>When we call pop() on empty list then we get IndexError

Examples:

```
>>> lst=[10,"Rossum",45.67,True,2+3j]
>>> print(lst,type(lst))-----[10, 'Rossum', 45.67, True, (2+3j)] <class 'list'>
>>> lst.pop()-----(2+3j)
>>> print(lst,type(lst))-----[10, 'Rossum', 45.67, True] <class 'list'>
>>> lst.pop()-----True
>>> print(lst,type(lst))-----[10, 'Rossum', 45.67] <class 'list'>
>>> lst.pop()-----45.67
>>> print(lst,type(lst))-----[10, 'Rossum'] <class 'list'>
>>> lst.pop()-----'Rossum'
>>> print(lst,type(lst))-----[10] <class 'list'>
>>> lst.pop()-----10
>>> print(lst,type(lst))-----[] <class 'list'>
>>> lst.pop()-----IndexError: pop from empty list
>>> list().pop()-----IndexError: pop from empty list
-----
>>> lst=[10,20,30,40,50]
>>> print(lst)-----[10, 20, 30, 40, 50]
>>> lst.insert(2,300)
>>> print(lst)-----[10, 20, 300, 30, 40, 50]
>>> lst.pop()-----50
```

NOTE: del operator

=>del operator is used for deleting Elements of any mutable object either based on Index or Based on Slicing or Total Object.

=>Syntax1: del object[Index]
 del object[Begin:End:Step]
 del object

=>With "del" operator we can't delete Immutable Content But we can delete complete Immutable Object.

Examples:

```
>>> lst=[10,"Rossum",45.67,True,2+3j,"Python"]
>>> print(lst)-----[10, 'Rossum', 45.67, True, (2+3j), 'Python']
>>> del lst[3] # Deleting Based on Index
>>> print(lst)-----[10, 'Rossum', 45.67, (2+3j), 'Python']
>>> del lst[2:4] # Deleting Based on Slicing
>>> print(lst)-----[10, 'Rossum', 'Python']
>>> del lst # Deleting Entire Object
>>> print(lst)-----NameError: name 'lst' is not defined. Did you mean: 'list'?
```

```
>>> s="PYTHON"
>>> print(s,type(s),id(s))-----PYTHON <class 'str'> 2073554063472
>>> s=s+"Prog"
>>> print(s,type(s),id(s))-----PYTHONProg <class 'str'> 2073554063280
>>> del s[0]-----TypeError: 'str' object doesn't support item deletion
>>> del s[0:3]-----TypeError: 'str' object does not support item deletion
>>> del s # Deleting Immutable object
>>> s-----NameError: name 's' is not defined
```

6) copy()

=>Syntax: object2=object1.copy()

=>This Function is used for Copying the content of one object into another object (Implementation of Shallow Copy)

Example:

Examples:

```
>>> l1=[10,"Rossum"]
>>> print(l1,id(l1))-----[10, 'Rossum'] 2073549864512
>>> l2=l1.copy() # Shallow Copy
>>> print(l2,id(l2))-----[10, 'Rossum'] 2073554063744
>>> l1.append("Python")
>>> l1.append("Python")
>>> l2.insert(1,"PSF")
>>> print(l1,id(l1))-----[10, 'Rossum', 'Python', 'Python'] 2073549864512
>>> print(l2,id(l2))-----[10, 'PSF', 'Rossum'] 2073554063744
```

Examples:---Deep Copy

```
>>> l1=[10,"Rossum"]
>>> print(l1,id(l1))-----[10, 'Rossum'] 2073554059392
>>> l2=l1 # Deep Copy
>>> print(l2,id(l2))-----[10, 'Rossum'] 2073554059392
>>> l1.append("Python")
>>> print(l1,id(l1))-----[10, 'Rossum', 'Python'] 2073554059392
>>> print(l2,id(l2))-----[10, 'Rossum', 'Python'] 2073554059392
>>> l2.insert(2,"PSF")
>>> print(l1,id(l1))-----[10, 'Rossum', 'PSF', 'Python'] 2073554059392
>>> print(l2,id(l2))-----[10, 'Rossum', 'PSF', 'Python'] 2073554059392
```

NOTE:- Slice Based Copy

```
>>> lst1=[10,20,30,40,50,60]
>>> print(lst1,id(lst1))-----[10, 20, 30, 40, 50, 60] 2073692289216
>>> lst2=lst1[0:3] # Slice Based Copy
>>> print(lst2,id(lst2))-----[10, 20, 30] 2073692289792
>>> lst2.append(12.34)
>>> lst1.append(70)
>>> print(lst1,id(lst1))-----[10, 20, 30, 40, 50, 60, 70] 2073692289216
>>> print(lst2,id(lst2))-----[10, 20, 30, 12.34] 2073692289792
>>>
>>> lst3=lst1[:] # Slice Based Copy
>>> print(lst3,id(lst3))-----[10, 20, 30, 40, 50, 60, 70] 2073686948288
>>> lst3.insert(1,"KVR")
>>> lst1.append(80)
>>> print(lst1,id(lst1))-----[10, 20, 30, 40, 50, 60, 70, 80] 2073692289216
>>> print(lst3,id(lst3))-----[10, 'KVR', 20, 30, 40, 50, 60, 70] 2073686948288
```

7) count():

Syntax:- listobj.count(Value)

=>This Function is used for Counting Number of Occurrences of a Specified Element.

=>If the Specified Element does not exist in list object then we get 0

Examples:

```
>>> lst=[10,20,30,40,10,20,10,60]
>>> print(lst)
[10, 20, 30, 40, 10, 20, 10, 60]
>>> lst.count(10)-----3
>>> len(lst)-----8
```

```
>>> lst.count(20)-----2
>>> lst.count(30)-----1
>>> lst.count(300)-----0
>>> lst.count("H")-----0
-----
>>> list().count(10)-----0
>>> [].count("")-----0
```

7) index()

=>Syntax:- listobj.index(Value)
=>This Function is used for finding Index of First Occurrence of Specified Element.
=>If the Specified Element not existing in list object then we get ValueError.

Examples:

```
>>> lst=[10,20,30,10,60,70,80,20,45]
>>> print(lst)-----[10, 20, 30, 10, 60, 70, 80, 20, 45]
>>> lst.index(10)-----0
>>> lst.index(20)-----1
>>> lst.index(60)-----4
>>> lst.index(45)-----8
>>> lst.index(145)-----ValueError: 145 is not in list
>>> list().index("KVR")-----ValueError: 'KVR' is not in list
>>> [10,20,30].index(10)-----0
>>> [10,20,30].index(100)-----ValueError: 100 is not in list
>>> [10,20,30].index("10")-----ValueError: '10' is not in list
```

8) reverse()

=>Syntax: listobj.reverse()
=>This Function is used for obtaining reverse the content of listobject (nothing but front to back and back to front)

Examples:

```
>>> l1=[10,20,30,-4,-5,100,12,45]
>>> print(l1,id(l1))-----[10, 20, 30, -4, -5, 100, 12, 45] 2670070726208
>>> l1.reverse()
>>> print(l1,id(l1))-----[45, 12, 100, -5, -4, 30, 20, 10] 2670070726208
>>> l1=["Python","java","R","DS"]
>>> print(l1,id(l1))-----['Python', 'java', 'R', 'DS'] 2670074921088
>>> l1.reverse()
>>> print(l1,id(l1))-----['DS', 'R', 'java', 'Python'] 2670074921088
```

9) sort()

-
- =>This function is used for sorting the Homogeneous (Similar) data either in Ascending Order (reverse = False) or in Descending Order (reverse=True)
- =>When we call sort() on list object where it contains Hetrogeneous (different) data then we get TypeError.
- =>Syntax: listobj.sort() ---- Display the data in Ascending Order
- =>Syntax: listobj.sort(reverse=False)---Display the data in Ascending Order
(default value of reverse is False)
- =>Syntax: listobj.sort(reverse=True)---Display the data in Descending Order

Examples:

```
>>> l1=[10,-4,23,15,56,3,-5,34,0]
>>> print(l1,id(l1))-----[10, -4, 23, 15, 56, 3, -5, 34, 0] 2670070726208
>>> l1.sort()
>>> print(l1,id(l1))-----[-5, -4, 0, 3, 10, 15, 23, 34, 56] 2670070726208
>>> l2=["Travis","Kinney","Rossum","Trump","Biden","Dennis","Anil"]
>>> print(l2)---['Travis', 'Kinney', 'Rossum', 'Trump', 'Biden', 'Dennis', 'Anil']
>>> l2.sort()
>>> print(l2)---['Anil', 'Biden', 'Dennis', 'Kinney', 'Rossum', 'Travis', 'Trump']

>>> l3=[10,"Rossum",34.56,True]
>>> l3.sort()-----TypeError: '<' not supported between instances of 'str' and 'int'

>>> l2=["Travis","Kinney","Rossum","Trump","Biden","Dennis","Anil"]
>>> print(l2)-----['Travis', 'Kinney', 'Rossum', 'Trump', 'Biden', 'Dennis', 'Anil']
>>> l2.sort()
>>> print(l2)-----['Anil', 'Biden', 'Dennis', 'Kinney', 'Rossum', 'Travis', 'Trump']
>>> l2.reverse()
>>> print(l2)-----['Trump', 'Travis', 'Rossum', 'Kinney', 'Dennis', 'Biden', 'Anil']

>>> l1=[10,-4,23,15,56,3,-5,34,0]
>>> print(l1,id(l1))-----[10, -4, 23, 15, 56, 3, -5, 34, 0] 2670074921088
>>> l1.sort()
>>> print(l1,id(l1))-----[-5, -4, 0, 3, 10, 15, 23, 34, 56] 2670074921088
>>> l1.reverse()
>>> print(l1,id(l1))-----[56, 34, 23, 15, 10, 3, 0, -4, -5] 2670074921088

>>> l1=[10,-4,23,15,56,3,-5,34,0]
>>> print(l1,id(l1))-----[10, -4, 23, 15, 56, 3, -5, 34, 0] 2670070726208
>>> l1.sort(reverse=True)
>>> print(l1,id(l1))-----[56, 34, 23, 15, 10, 3, 0, -4, -5] 2670070726208
```

```
-----  
>>> l1=[10,-4,23,15,56,3,-5,34,0]  
>>> print(l1,id(l1))  
[10, -4, 23, 15, 56, 3, -5, 34, 0] 2670070726208  
>>> l1.sort(reverse=False) # OR l1.sort()  
>>> print(l1,id(l1))-----[-5, -4, 0, 3, 10, 15, 23, 34, 56] 2670070726208
```

```
-----  
>>> l1=[10,-4,23,15,56,3,-5,34,0]  
>>> print(l1,id(l1))-----[10, -4, 23, 15, 56, 3, -5, 34, 0] 2670074921088  
>>> l1.sort()  
>>> print(l1,id(l1))-----[-5, -4, 0, 3, 10, 15, 23, 34, 56] 2670074921088
```

10) extend()

=> Syntax: listobj1.extend(listobj2)

=> This Function is used for extending the functionality of listobj1 with the values of listobj2.
=> At any point time, extend() takes one list object as argument
=> If we want extend the functionality of one list object with multiple objects then we can use + operator.
=> Syntax:- listobj1=listobj1+listobj2+.....listobj-n

Examples:

```
>>> l1=[10,20,30]  
>>> l2=["RS","TR","SD"]  
>>> l1.extend(l2)  
>>> print(l1)-----[10, 20, 30, 'RS', 'TR', 'SD']  
>>> print(l2)-----['RS', 'TR', 'SD']
```

```
-----  
>>> l1=[10,20,30]  
>>> l2=["RS","TR","SD"]  
>>> l2.extend(l1)  
>>> print(l1)-----[10, 20, 30]  
>>> print(l2)-----['RS', 'TR', 'SD', 10, 20, 30]
```

```
-----  
>>> l1=[10,20,30]  
>>> l2=["RS","TR","SD"]  
>>> l3=["Python","R"]  
>>> l1.extend(l2,l3)-----TypeError: list.extend() takes exactly one argument (2 given)  
NOTE:
```

```
>>> l1=l1+l2+l3  
>>> print(l1)-----[10, 20, 30, 'RS', 'TR', 'SD', 'Python', 'R']
```

11) clear()

=> Syntax: listobj.clear()

=>This Function is used removing all the elements of non-empty list

=>

Examples:

```
>>> l1=[10,-4,23,15,56,3,-5,34,0]
>>> print(l1,id(l1))-----[10, -4, 23, 15, 56, 3, -5, 34, 0] 2670074921088
>>> len(l1)-----9
>>> l1.clear()
>>> print(l1,id(l1))-----[] 2670074921088
>>> len(l1)-----0
-----  
>>> print([] .clear())-----None
>>> print(list().clear())-----None
```

===== Copy Techniques in Python =====

=>In Python Programming, we have 2 types of Copy Techniques. They are

1. Shallow Copy
 2. Deep Copy
-

1. Shallow Copy

=>The Properties of Shallow Copy are

- a) Initial Content of Both the Objects are Same.
- b) Both the Objects Memory Address are Different
- c) Modifications are Independent (Whatever the modifications we do on any one object they are not reflecting another object)

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

=>Syntax: object2=object1.copy()

Examples:

```
>>> l1=[10,"Rossum"]
>>> print(l1,id(l1))-----[10, 'Rossum'] 2073549864512
>>> l2=l1.copy() # Shallow Copy
>>> print(l2,id(l2))-----[10, 'Rossum'] 2073554063744
>>> l1.append("Python")
>>> l1.append("Python")
>>> l2.insert(1,"PSF")
```

```
>>> print(l1,id(l1))-----[10, 'Rossum', 'Python', 'Python'] 2073549864512  
>>> print(l2,id(l2))-----[10, 'PSF', 'Rossum'] 2073554063744
```

2. Deep Copy

=>The Properties of Deep Copy are

- a) Initial Content of Both the Objects are Same.
- b) Both the Objects Memory Address are Same
- c) Modifications are Dependent (Whatever the modifications we do

on any one

object they are reflecting to another object)

=>To Implement Deep Copy, we use Assignment Operator (=)

=>Syntax: object2 = object1

Examples:

```
>>> l1=[10,"Rossum"]  
>>> print(l1,id(l1))-----[10, 'Rossum'] 2073554059392  
>>> l2=l1 # Deep Copy  
>>> print(l2,id(l2))-----[10, 'Rossum'] 2073554059392  
>>> l1.append("Python")  
>>> print(l1,id(l1))-----[10, 'Rossum', 'Python'] 2073554059392  
>>> print(l2,id(l2))-----[10, 'Rossum', 'Python'] 2073554059392  
>>> l2.insert(2,"PSF")  
>>> print(l1,id(l1))-----[10, 'Rossum', 'PSF', 'Python'] 2073554059392  
>>> print(l2,id(l2))-----[10, 'Rossum', 'PSF', 'Python'] 2073554059392
```

=====

Inner List OR Nested List

=====

=>The Process of defining one list in another list is called Inner or Nested List

=>Syntax:

listobj=[Val1, Val2.....[Val11,Val12..] , [Val21,Val22.....], Val-n]

=>here [Val11,Val12..] is one Inner List

=>Here [Val21,Val22.....] is another Inner list

=>[Val1, Val2....., Val-n] is called Outer List

Examples:

```
-----  
>>> sinfo=[10,"Rossum",[19,17,20], [78,77,79],"OUCET" ]  
>>> sinfo[0]-----10  
>>> sinfo[-1]----- 'OUCET'  
>>> sinfo[1]----- 'Rossum'  
>>> sinfo[2]-----[19, 17, 20]  
>>> print(sinfo[2],type(sinfo[2]))-----[19, 17, 20] <class 'list'>  
>>> print(sinfo[2],type(sinfo[2]), type(sinfo))-----[19, 17, 20] <class 'list'> <class 'list'>  
>>> print(sinfo[-2],type(sinfo[-2]), type(sinfo))-----[78, 77, 79] <class 'list'> <class 'list'>  
>>> print(sinfo[0],type(sinfo[0]), type(sinfo))-----10 <class 'int'> <class 'list'>  
-----  
>>> sinfo=[10,"Rossum",[19,17,20], [78,77,79],"OUCET" ]  
>>> print(sinfo)-----[10, 'Rossum', [19, 17, 20], [78, 77, 79], 'OUCET']  
>>> sinfo[2][-1]-----20  
>>> sinfo[2][::]-----[19, 17, 20]  
>>> sinfo[2][::-1]-----[20, 17, 19]  
>>> sinfo[2][2]=18  
>>> print(sinfo)-----[10, 'Rossum', [19, 17, 18], [78, 77, 79], 'OUCET']  
>>> sinfo[2].sort()  
>>> print(sinfo)-----[10, 'Rossum', [17, 18, 19], [78, 77, 79], 'OUCET']  
>>> sinfo[-2].sort(reverse=True)  
>>> print(sinfo)-----[10, 'Rossum', [17, 18, 19], [79, 78, 77], 'OUCET']  
>>> sinfo[2][0:2]-----[17, 18]  
>>> sinfo[2][:2]-----[17, 19]  
>>> sinfo[-3].remove(18)  
>>> print(sinfo)-----[10, 'Rossum', [17, 19], [79, 78, 77], 'OUCET']  
>>> del sinfo[-2][1:]  
>>> print(sinfo)-----[10, 'Rossum', [17, 19], [79], 'OUCET']  
>>> sinfo[2].clear()  
>>> print(sinfo)-----[10, 'Rossum', [], [79], 'OUCET']  
>>> del sinfo[2]  
>>> print(sinfo)-----[10, 'Rossum', [79], 'OUCET']  
>>> del sinfo[-2]  
>>> print(sinfo)-----[10, 'Rossum', 'OUCET']  
>>> im=[16,17,14]  
>>> sinfo.insert(2,im)  
>>> print(sinfo)-----[10, 'Rossum', [16, 17, 14], 'OUCET']  
>>> sinfo.insert(3,[67,74,66])  
>>> print(sinfo)-----[10, 'Rossum', [16, 17, 14], [67, 74, 66], 'OUCET']  
-----  
>>> print(sinfo)-----[10, 'Rossum', [16, 17, 14], [67, 74, 66], 'OUCET']  
>>> k=["PYTHON","R"]  
>>> sinfo[2].insert(1,k)  
>>> print(sinfo)-----[10, 'Rossum', [16, ['PYTHON', 'R'], 17, 14], [67, 74, 66], 'OUCET']
```

consider the following

sinfo=[10, 'Rossum', [19, 17, 20], [78, 77, 79], 'OUCET']

		-5	-4	-3	-2	-1	
sinfo-->	10	Rossum		-3 -2 -1	-3 -2 -1	OUCET	
	0	1	2	3	4		
		19 17 20		78 77 79			
		0 1 2		0 1 2			

2) tuple

- =>'tuple' is one of the pre-defined class and treated as list data type.
- =>The purpose of tuple data type is that "To store Collection of Values or multiple values either of Same type or different type or both the types with unique and duplicate."
- =>The elements of tuple must be stored within braces () and the elements must be separated by comma.

=>An object of tuple maintains insertion Order.

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

=>An object of tuple belongs to immutable bcoz tuple' object does not support item assignment

=>To convert any other object into tuple type object, we use tuple()
Syntax:- tupleobject=tuple(another object)

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

- a) empty tuple
- b) non-empty tuple

a) empty tuple:

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

=>Syntax:- tupleobj=()

or

tupleobj=tuple()

Examples:

```
>>> t=()
```

```
>>> print(t,type(t),id(l))----- () <class 'tuple'>
2722448959680
>>> len(t)----- 0
>>> l1=tuple()
>>> print(l1,type(l1),id(l1))----- () <class 'tuple'>
2722452472064
>>> len(l1)----- 0
```

b) non-empty tuple:

=>A non-empty tuple is one, which contains elements and length is >0
Syntax:- tplobj=(val1,val2...val-n)

(OR)
tplobj=val1,val2...val-n

Note: The Functionality of tuple is exactly similar to list but an object of list belongs to mutable and an object of tuple belongs to immutable.

Examples:

```
>>> t1=(10,20,30,40,10,10)
>>> print(t1,type(t1))-----(10, 20, 30, 40, 10, 10) <class 'tuple'>
>>> t2=(10,"Ram",34.56,True,2+4.5j)
>>> print(t2,type(t2),id(t2))-----(10, 'Ram', 34.56, True, (2+4.5j)) <class 'tuple'>
>>> t2[0]-----10
>>> t2[1]-----'Ram'
>>> t2[-1]-----(2+4.5j)
>>> t2[1:4]-----('Ram', 34.56, True)
>>> t2[2]=56.78-----TypeError: 'tuple' object does not support item assignment
```

```
>>> t1=()
>>> print(t1,len(t1))-----() 0
(OR)
>>> t2=tuple()
>>> print(t2,len(t2))-----() 0
```

```
>>> l1=[10,"Rossum"]
>>> print(l1,type(l1))-----[10, 'Rossum'] <class 'list'>
>>> t1=tuple(l1)
>>> print(t1,len(t1))-----(10, 'Rossum') 2
```

```
>>> a=10,"KVR ","Python",True # without braces ( )
>>> print(a,type(a))-----(10, 'KVR', 'Python', True) <class 'tuple'>
>>> a=10,
```

```

>>> print(a,type(a))-----(10,) <class 'tuple'>
>>> a=10
>>> print(a,type(a))-----10 <class 'int'>
>>> t=tuple(a)-----TypeError: 'int' object is not iterable
>>> t=tuple(a,)-----TypeError: 'int' object is not iterable
>>> t=tuple((a))-----TypeError: 'int' object is not iterable
>>> t=(a,) # correct conversion
>>> print(t,type(t))-----(10,) <class 'tuple'>
>>> print(a,type(a))-----10 <class 'int'>
-----X-----
=====
        pre-defined functions in tuple
=====
```

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

1. count()
2. index()

Examples:

```

-----  

>>> t1=(10,10,20,30,10,10,30)
>>> t1.count(10)-----4
>>> t1.count(30)-----2
>>> t1.count(300)-----0
>>> t1.count("KVR")-----0  

-----  

>>> t1=(10,10,20,30,10,10,30)
>>> t1.index(10)-----0
>>> t1.index(20)-----2
>>> t1.index(230)-----ValueError: tuple.index(x): x not in tuple  

-----
```

```

>>> t1=(10,10,20,30,10,10,30)
>>> for i,v in enumerate(t1):
...     print(i,v)
```

Output

```

0 10
1 10
2 20
3 30
4 10
5 10
6 30
```

NOTE: tuple object does not contain the following pre-defined Functions bcoz tuple object belongs to immutable.

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

=====

Inner tuple OR Tuple List

=====

=>The Process of defining one tuple in another tuple is called Inner or Nested tuple

=>Syntax:

tupleobj=(Val1, Val2.....(Val11,Val12..) , (Val21,Val22.....), Val-n)

=>here (Val11,Val12..) is one Inner tuple

=>Here (Val21,Val22.....) is another Inner tuple

=>(Val1, Val2....., Val-n) is called Outer tuple

NOTE:

=>We can define One Tuple Inside of Another Tuple

=>We can define One List Inside of Another List

=>We can define One Tuple Inside of Another List

=>We can define One List Inside of Another Tuple

Examples

```
>>> t1=(10,"Rossum",(15,18,17),(66,67,56),"OUCET")
>>> print(t1,type(t1))-----(10, 'Rossum', (15, 18, 17), (66, 67, 56), 'OUCET') <class
'tuple'>
>>> t1[2]-----(15, 18, 17)
>>> print(t1[2],type(t2))-----(15, 18, 17) <class 'tuple'>
>>> print(t1[-2],type(t2))-----(66, 67, 56) <class 'tuple'>

>>> t1=(10,"Rossum",[15,18,17],(66,67,56),"OUCET")
>>> print(t1,type(t1))-----(10, 'Rossum', [15, 18, 17], (66, 67, 56), 'OUCET') <class 'tuple'>
>>> print(t1[2],type(t1[2]))-----[15, 18, 17] <class 'list'>
>>> print(t1[3],type(t1[3]))-----(66, 67, 56) <class 'tuple'>
>>> t1[2].insert(1,16)
>>> print(t1,type(t1))-----(10, 'Rossum', [15, 16, 18, 17], (66, 67, 56), 'OUCET') <class 'tuple'>
```

```
>>> t1[2].sort(reverse=True)
>>> print(t1,type(t1))-----(10, 'Rossum', [18, 17, 16, 15], (66, 67, 56), 'OUCET') <class
'tuple'>
-----
>>> l1=[10,"Rossum",[15,18,17],(66,67,56),"OUCET"]
>>> print(l1,type(l1))-----[10, 'Rossum', [15, 18, 17], (66, 67, 56), 'OUCET'] <class 'list'>
>>> l1[2].remove(18)
>>> print(l1,type(l1))-----[10, 'Rossum', [15, 17], (66, 67, 56), 'OUCET'] <class 'list'>
=====X=====
=====
```

Special Case:

sorted():

=>It is one of the general pre-defined function and is used for Sorting the elements of tuple (in this case) and gives the sorted elements in th form of list(But Sorted Elements will not place in tuple bcoz tuple is immutable).

Syntax: sorted(tuple object)

(OR)
listobj=sorted(tupleobj)

Examples:

```
>>> t1=(12,45,-3,3,0,14)
>>> print(t1,type(t1))-----(12, 45, -3, 3, 0, 14) <class 'tuple'>
>>> t1.sort()-----AttributeError: 'tuple' object has no attribute 'sort'
>>> sorted(t1)----- [-3, 0, 3, 12, 14, 45]
>>> print(t1,type(t1))-----(12, 45, -3, 3, 0, 14) <class 'tuple'>
>>> x=sorted(t1)
>>> print(x,type(x))-----[-3, 0, 3, 12, 14, 45] <class 'list'>
(OR)
>>> t1=(12,45,-3,3,0,14)
>>> print(t1,type(t1))-----(12, 45, -3, 3, 0, 14) <class 'tuple'>
>>> l1=list(t1)
>>> print(l1,type(l1))-----[12, 45, -3, 3, 0, 14] <class 'list'>
>>> l1.sort()
>>> print(l1,type(l1))-----[-3, 0, 3, 12, 14, 45] <class 'list'>
>>> t1=tuple(l1)
>>> print(t1,type(t1))-----(-3, 0, 3, 12, 14, 45) <class 'tuple'>
```

=====

Set Category Data Types(Collections Data Types or Data Structures)

=====

=>The purpose of Set Category Data Types is that " To store Collection or multiple values either of same type or different type or both the types with Unique Values (No duplicates are allowed)".

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

1. set (mutable and immutable)
 2. frozenset (immutable)
-
-
-

1. set

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

=>The purpose of set data type is that " To store Collection or multiple values either of same type or different type or both the types with Unique Values (No duplicates are allowed)".

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

=>An object of set does not maintain insertion order bcoz PVM displays any order of multiple possibilities.

=>On the object of set, we can't perform Indexing and slicing Operations bcoz set object does not maintain Insertion order.

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

=>By using set class, we can two types of set objects. They are

- a) empty set
- b) non-empty set

a) empty set:

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

=>Syntax:- setobj=set()

b) non-empty set:

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

=>Syntax:- setobj={ val1,val2...val-n }

=>To convert one type of object into set type object, we use set()

Syntax: setobj=set(obj)

Examples:

```
>>> s1={ 10,20,30,40,50,10,10,20,75 }
>>> print(s1,type(s1))-----{ 50, 20, 40, 10, 75, 30 } <class 'set'>
>>> s1={ 10,20,25,35,10,20 }
>>> print(s1,type(s1))-----{ 25, 10, 35, 20 } <class 'set'>
>>> s1[0]-----TypeError: 'set' object is not subscriptable
>>> s1[0:3]-----TypeError: 'set' object is not subscriptable
```

```
>>> s1={10,20,30,40,50}
>>> print(s1,id(s1))-----{50, 20, 40, 10, 30} 1473821509440
>>> s1[0]=100-----TypeError: 'set' object does not support item assignment
>>> s1.add("KVR")
>>> print(s1,id(s1))-----{50, 20, 40, 10, 'KVR', 30} 1473821509440
-----
>>> s1=set()
>>> print(s1,type(s1))-----{} <class 'set'>
>>> len(s1)-----0
>>> s2={10,20,30,10,20}
>>> print(s2,type(s2))-----{10, 20, 30} <class 'set'>
>>> len(s2)-----3
-----
>>> l1=[10,20,10,20,"Python",23.45]
>>> s1=set(l1)
>>> print(s1)-----{10, 20, 23.45, 'Python'}
>>> t1=tuple(s1)
>>> print(t1,type(t1))-----(10, 20, 23.45, 'Python') <class 'tuple'>
>>> t1=list(s1)
>>> print(t1,type(t1))-----[10, 20, 23.45, 'Python'] <class 'list'>
=====X=====
```

===== pre-defined functions in set =====

=>on the object of set, we can perform different type of Operations by using pre-defined functions in set object.

1) add()

=>This Function is used for adding the elements to set object.

=>Syntax: setobj.add(Value)

Examples:

```
>>> s1={10,20,30}
>>> print(s1,type(s1),id(s1))-----{10, 20, 30} <class 'set'> 1691649314592
>>> s1.add(12.34)
>>> print(s1,type(s1),id(s1))-----{10, 20, 12.34, 30} <class 'set'> 1691649314592
>>> s1.add("python")
>>> print(s1,type(s1),id(s1))-----{10, 12.34, 'python', 20, 30} <class 'set'> 1691649314592
>>> s2=set()
>>> print(s2,type(s2),id(s2))-----set() <class 'set'> 1691645340672
>>> s2.add(100)
```

```
>>> s2.add("Rajesh")
>>> s2.add("Kasif")
>>> print(s2,type(s2),id(s2))----{ 100, 'Kasif', 'Rajesh'} <class 'set'> 1691645340672
>>> s2.add(23.45)
>>> print(s2,type(s2),id(s2))----{ 100, 23.45, 'Kasif', 'Rajesh'} <class 'set'> 1691645340672
```

2) remove()

=>Syntax:- setobj.remove(Value)
=>This Function is used for removing the element from set object.
=>The element / value does not exist in setobject then we get KeyError(bcoz all the elements of set are Unique and they are called Keys)

Examples:

```
>>> s1={ 10,"Rajesh",34.56,400,True,2+3j}
>>> print(s1,type(s1))----{ 400, True, 34.56, 'Rajesh', 10, (2+3j)} <class 'set'>
>>> print(s1,type(s1),id(s1))---{ 400, True, 34.56, 'Rajesh', 10, (2+3j)} <class 'set'>
>>> s1.remove(34.56)
>>> print(s1,type(s1),id(s1))----{ 400, True, 'Rajesh', 10, (2+3j)} <class 'set'> 1691649315936
>>> s1.remove(True)
>>> print(s1,type(s1),id(s1))---{ 400, 'Rajesh', 10, (2+3j)} <class 'set'> 1691649315936
>>> s1.remove("Rajesh")
>>> print(s1,type(s1),id(s1))----{ 400, 10, (2+3j)} <class 'set'> 1691649315936
>>> s1.remove("KVR")-----KeyError: 'KVR'
>>> set().remove(10)-----KeyError: 10
```

3) discard()

=>Syntax: setobj.discard(value)
=>This Function is used for removing theelement from set object.
=>The element / value does not exist in setobject then we never get KeyError
Examples:

```
>>> s1={ 10,20,30,40,50,60,70,10}
>>> print(s1,type(s1))-----{ 50, 20, 70, 40, 10, 60, 30} <class 'set'>
>>> s1.discard(50)
>>> print(s1,type(s1))-----{ 20, 70, 40, 10, 60, 30} <class 'set'>
>>> s1.discard(10)
>>> print(s1,type(s1))-----{ 20, 70, 40, 60, 30} <class 'set'>
>>> s1.discard(100) # we never get KeyError
>>> print(s1,type(s1))-----{ 20, 70, 40, 60, 30} <class 'set'>
```

```
>>> s1.discard("Python") # we never get KeyError  
>>> s1.remove("Python")-----KeyError: 'Python'
```

4) clear()

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

```
>>> s1={ 10,20,30,40,50,60,70,10}  
>>> print(s1,type(s1))-----{ 50, 20, 70, 40, 10, 60, 30} <class 'set'>  
>>> len(s1)-----7  
>>> s1.clear()  
>>> print(s1,type(s1))-----set() <class 'set'>  
>>> len(s1)-----0  
>>> print( set().clear() )-----None
```

5) copy() -----Shallow Copy

Syntax: setobj2=setobj1.copy()
=>This Function is used for copying the content of one set object into another set object

=>Examples:

```
>>> s1={ 10,20,30,40,50,60,70,10}  
>>> print(s1,type(s1),id(s1))-----{ 50, 20, 70, 40, 10, 60, 30} <class 'set'> 2424304921600  
>>> s2=s1.copy()  
>>> print(s2,type(s2),id(s2))-----{ 50, 20, 70, 40, 10, 60, 30} <class 'set'> 2424308895072  
>>> s1.add(12.34)  
>>> s2.add("Python")  
>>> print(s1,type(s1),id(s1))-----{ 50, 20, 70, 40, 10, 60, 12.34, 30} <class 'set'> 2424304921600  
>>> print(s2,type(s2),id(s2))---{ 50, 20, 'Python', 70, 40, 10, 60, 30} <class 'set'> 2424308895072
```

6) isdisjoint()

=>Syntax: setobj1.isdisjoin(s2)

=>This Function returns True Provided there is no common eleement between setobj1 and setobj2.
=>This Function returns False Provided there is atleast common eleement between setobj1

and setobj2.

Examples:

```
>>> s1={10,20,30,40}  
>>> s2={"Apple","Mango","kiwi"}  
>>> s3={10,50,60}  
>>> s1.isdisjoint(s2)-----True  
>>> s1.isdisjoint(s3)-----False
```

7) issuperset()

Syntax: setobj1.issuperset(setobj2)

=>This Function return True provided all elements setobj2 must present setobj1

OR

setobj1 must contains all elements of setobj2

Examples:

```
>>> s1={10,20,30,40}  
>>> s2={10,20}  
>>> s3={10,20, "Apple","Mango","kiwi"}  
>>> s1.issuperset(s2)-----True  
>>> s1.issuperset(s3)-----False
```

8) issubset()

Syntax: setobj1.issubset(setobj2)

=>This Function return True provided all elements setobj1 must present setobj2

OR

setobj2 must contains all elements of setobj1

Examples:

```
>>> s1={10,20,30,40}  
>>> s2={10,20}  
>>> s3={10,20, "Apple","Mango","kiwi"}  
>>> s2.issubset(s1)-----True  
>>> s3.issubset(s1)-----False  
>>> s3.issubset(s2)-----False  
>>> s2.issubset(s3)-----True
```

9) union()

Syntax:- setobj1.union(setobj2)
(OR)
setobj3=setobj1.union(setobj2)

=>This is used for obtaining all Unique Elements of setobj1 and setobj2 and result unique values placed in setobj3.

Examples:

```
>>> s1={ 10,20,30,40 }
>>> s2={ 15,10,25 }
>>> s3=s1.union(s2)
>>> print(s1)-----{ 40, 10, 20, 30 }
>>> print(s2)-----{ 25, 10, 15 }
>>> print(s3)-----{ 20, 40, 25, 10, 30, 15 }
>>>
>>> print(s1.union(s2))-----{ 20, 40, 25, 10, 30, 15 }
```

10) intersection()

Syntax: setobj1.intersection(setobj2)
(OR)
setobj3= setobj1.intersection(setobj2)

=>This function is used for obtaining common elements from setobj1 and setobj2.

Examples:

```
>>> s1={ 10,20,30,40 }
>>> s2={ 15,10,25 }
>>> s3=s1.intersection(s2)
>>> print(s3)-----{ 10 }
>>> s3=s2.intersection(s1)
>>> print(s3)-----{ 10 }
>>> print(s1.intersection(s2))-----{ 10 }

>>> s1={ 10,20,30,40 }
>>> s2={ "Apple","Mango","kiwi" }
>>> print(s1.intersection(s2))-----set()
```

11)difference()

=>Syntax: setobj1.difference(setobj2)
=>This obtains removes common elements from setobj1 and setobj2 and Takes remaining elements from setobj1 and place them setobj3.

Examples:

```
-----  
>>> s1={ 10,20,30,40 }  
>>> s2={ 10,15,25 }  
>>> s3=s1.difference(s2)  
>>> print(s1)-----{ 40, 10, 20, 30 }  
>>> print(s2)-----{ 25, 10, 15 }  
>>> print(s3)-----{ 40, 20, 30 }  
>>> s4=s2.difference(s1)  
>>> print(s4)-----{ 25, 15 }
```

```
-----  
>>> a = { 1, 3 ,5 }  
>>> b = { 2, 4, 6 }  
>>> c = { 1, 2 }  
>>> print(a)-----{ 1, 3, 5 }  
>>> print(b)-----{ 2, 4, 6 }  
>>> print(c)-----{ 1, 2 }  
>>> d=a.difference(b).difference(c)  
>>> print(d)-----{ 3, 5 }  
>>> d=a.difference(b,c)  
>>> print(d)-----{ 3, 5 }
```

12) symmetric_difference()

=>Syntax: setobj1.symmetric_difference(setobj2)
=>This function removes common elements from both setobj1 and setobj2 and Takes remaining elements from both setobj1 and setobj2 and place them setobj3.

Examples:

```
-----  
>>> s1={ 10,20,30,40 }  
>>> s2={ 10,15,25 }  
>>> s3=s1.symmetric_difference(s2)  
>>> print(s1)-----{ 40, 10, 20, 30 }  
>>> print(s2)-----{ 25, 10, 15 }  
>>> print(s3)-----{ 40, 15, 20, 25, 30 }  
>>> s3=s2.symmetric_difference(s1)  
>>> print(s3)-----{ 40, 15, 20, 25, 30 }
```

Use-Case:

```
-----
>>> cp={"sachin","kohli","rohit"}
>>> tp={"rossum","saroj","rohit"}
-----
>>> allcptp=cp.union(tp)
>>> print(allcptp)-----{'kohli', 'sachin', 'rohit', 'rossum', 'saroj'}
>>> bothcptp=cp.intersection(tp)
>>> print(bothcptp)-----{'rohit'}
>>> onlycp=cp.difference(tp)
>>> print(onlycp)-----{'kohli', 'sachin'}
>>> onlytp=tp.difference(cp)
>>> print(onlytp)-----{'rossum', 'saroj'}
>>> exclcptp=cp.symmetric_difference(tp)
>>> print(exlcptp)-----{'sachin', 'rossum', 'kohli', 'saroj'}
```

MOST IMP Case:

```
>>> allcptp=cp|tp # Bitwise OR Operator ( | )
>>> print(allcptp)-----{'kohli', 'sachin', 'rohit', 'rossum', 'saroj'}
>>> bothptp=cp&tp # Bitwise AND Operator ( & )
>>> print(bothcptp)-----{'rohit'}
>>> onlycp=cp-tp # Subtract Operator
>>> print(onlycp)-----{'kohli', 'sachin'}
>>> onlytp=tp-cp # Subtract Operator
>>> print(onlytp)-----{'rossum', 'saroj'}
>>> exclcptp=cp^tp # Biwise XOR Operator ( ^ )
>>> print(exlcptp)-----{'sachin', 'rossum', 'kohli', 'saroj'}
```

```
>>> print({10,20,30}) & {10,25,67,34})-----{10}
```

13) update()

```
--> Syntax: setobj1.update(setobj2)
=> This Function is used for updating the values of setobj2 with setobj1.
```

Examples:

```
>>> s1={10,20,30}
>>> s2={"Python","Java"}
>>> print(s1,id(s1))-----{10, 20, 30} 2424308898432
>>> print(s2,id(s2))-----{'Java', 'Python'} 2424308895072
>>> s1.update(s2)
>>> print(s1,id(s1))-----{20, 'Java', 10, 'Python', 30} 2424308898432
```

```
>>> s1={10,20,30}
```

```
>>> s2={ 10,20,"Python" }
>>> print(s1,id(s1))-----{ 10, 20, 30} 2424308896416
>>> print(s2,id(s2))-----{ 10, 20, 'Python'} 2424308898432
>>> s1.update(s2)
>>> print(s1,id(s1))-----{ 20, 10, 'Python', 30} 2424308896416
```

14) pop()

=>Syntax: setobj.pop()
=>This Function is used for removing any Arbitrary Element from setobject.
=>when we call pop() on empty set() then we get KeyError
Examples:

```
>>> s1={ 10,"Abinash","Python",45.67,True,2+3j}
>>> s1.pop()-----True
>>> s1.pop()-----10
>>> s1.pop()-----'Abinash'
>>> s1.pop()-----'Python'
>>> s1.pop()-----(2+3j)
>>> s1.pop()-----45.67
>>> s1.pop()-----KeyError: 'pop from an empty set'
>>> set().pop()-----KeyError: 'pop from an empty set'
```

Nested or Inner Formulas

Imp Points:

=> Set in Set Not Possible
=>Tuple in set Possible (No use bcoz we can't locate by using Indexing)
=>List in set Not Possible (bcoz list is mutable and allows changes)
=>set in Tuple Possible (boz tuple permits to locate set object by using indexing)
=>Set in list Possible (boz tuple permits to locate set object by using indexing)

Examples:

```
>>> l1=[10,"Akash",{ 10,20,30},[23,45,23],"OUCET" ]
>>> print(l1,type(l1))
[10, 'Akash', { 10, 20, 30}, [23, 45, 23], 'OUCET'] <class 'list'>
>>> print(l1[0],type(l1[0]))-----10 <class 'int'>
>>> print(l1[1],type(l1[2]))-----Akash <class 'set'>
>>> print(l1[2],type(l1[2]))-----{ 10, 20, 30} <class 'set'>
>>> l1[2][0]-----TypeError: 'set' object is not subscriptable
>>> l1[:3]-----[10, 'Akash', { 10, 20, 30}]
>>> l1[2].add(23)
>>> print(l1)-----[10, 'Akash', { 10, 20, 30, 23}, [23, 45, 23], 'OUCET']
```

```
>>> l1[-2][0]-----23
>>> l1[-3][0]-----TypeError: 'set' object is not subscriptable
-----
>>> t1=(10,"Akash",{ 10,20,30},[23,45,23],"OUCET")
>>> print(t1,type(t1))----(10, 'Akash', { 10, 20, 30 }, [23, 45, 23], 'OUCET') <class 'tuple'>
>>> print(t1[2],type(t1[2]))----{ 10, 20, 30 } <class 'set'>
>>> print(t1[2],type(t1[3]))----{ 10, 20, 30 } <class 'list'>
-----
>>> s1={ 10,"Akash",10,20,30,(23,45,23),"OUCET" }
>>> print(s1,type(s1))----{'OUCET', 'Akash', (23, 45, 23), 10, (10, 20, 30)} <class 'set'>
>>> print(s1[2],type(s1[2]))---TypeError: 'set' object is not subscriptable
>>> s1={ 10,"Akash",10,20,30,(23,45,23),"OUCET"}---TypeError: unhashable type: 'list'
```

2. frozenset

=>'frozenset' is one of the pre-defined class and treated as set data type.
=>The purpose of frozenset data type is that To store multiple values of either of same type or different type or both types with Unique Values in single object."
=>The elements set must organized with curly braces {} and values must separated by comma and those values can converted into frozenset by using frozenset()

Syntax:- frozensetobj1=frozenset(setobj)
 frozensetobj1=frozenset(listobj)
 frozensetobj1=frozenset(tupleobj)

=>An object of frozenset does not maintain insertion Order bcoz PVM displays any possibility of elements of frozenset
=>Since frozenset object does not maintain insertion order, we can't perform Indexing and Slicing Operations (frozenset' object is not subscriptable)
=>An object of frozenset belongs to immutable (in the case frozenset' object does not support item assignment and adding elements also not possible)

Note:-The Functionality of frozenset is similar to set but an object of set belongs to both immutable (in case of item assigment) and mutable (in the case of add()) where as an object frozenset belongs to immutable.

Examples:

```
l1=[10,20,30,40,10]
fs=frozenset(l1)
print(fs,type(fs))----frozenset({40, 10, 20, 30}) <class 'frozenset'>
fs.add(100)----AttributeError: 'frozenset' object has no attribute 'add'
fs[0]=345----TypeError: 'frozenset' object does not support item assignment
-----
>>> t1=(10,20,30,10,40,23,45,56)
```

```
>>> print(t1,type(t1))-----(10, 20, 30, 10, 40, 23.45, 56) <class 'tuple'>
>>> fs1=frozenset(t1)
>>> print(fs1,type(fs1))-----frozenset({40, 10, 20, 23.45, 56, 30}) <class 'frozenset'>
>>> s1={ 10,"KVR",34.56,"Python","Java"}
>>> print(s1,type(s1))-----{34.56, 10, 'KVR', 'Java', 'Python'} <class 'set'>
>>> fs2=frozenset(s1)
>>> print(fs2,type(fs2))-----frozenset({34.56, 10, 'KVR', 'Java', 'Python'}) <class 'frozenset'>
>>> fs2[0]-----TypeError: 'frozenset' object is not subscriptable
>>> fs2[0:3]-----TypeError: 'frozenset' object is not subscriptable
>>> fs2[0]=123-----TypeError: 'frozenset' object does not support item assignment
>>> fs2.add(100)-----AttributeError: 'frozenset' object has no attribute 'add'
```

Pre-defined functions in frozenset

- 1) copy()
 - 2) union()
 - 3) intersection()
 - 4) difference()
 - 5) symmetric_difference()
-

Examples:

```
>>> s1={ 10,20,30,40}
>>> s2={ 15,25,30,40}
>>> fs1=frozenset(s1)
>>> fs2=frozenset(s2)
>>> print(fs1)
frozenset({40, 10, 20, 30})
>>> print(fs2)
frozenset({40, 25, 30, 15})
>>> fs3=fs1.union(fs2)
>>> print(fs3)
frozenset({40, 10, 15, 20, 25, 30})
>>> fs4=fs1.intersection(fs2)
>>> print(fs4)
frozenset({40, 30})
>>> fs5=fs1.difference(fs2)
>>> print(fs5)
frozenset({10, 20})
>>> fs6=fs2.difference(fs1)
>>> print(fs6)
frozenset({25, 15})
>>> fs7=fs2.symmetric_difference(fs1)
>>> print(fs7)
frozenset({10, 15, 20, 25})
>>> fs7=fs1.symmetric_difference(fs2)
```

```
>>> print(fs7)
frozenset({10, 15, 20, 25})
-----
>>> s1={10,20,30,40}
>>> fs1=frozenset(s1)
>>> fs2=fs1.copy()
>>> print(fs1,id(fs1))----- frozenset({40, 10, 20, 30}) 2299638113984
>>> print(fs2,id(fs2))----- frozenset({40, 10, 20, 30}) 2299638113984
===== X =====
```

The following Pre-defined functions not found in frozenset

- 1) add()
 - 2) remove()
 - 3) discard()
 - 4) update()
 - 5) pop()
 - 6) clear()

dict Catagery Data Type (Collection Data Types or Data Structures)

=>The purpose of dict Catagory Data Type is the "To organize the data in the form of (Key,Value)"

=>To organize the data in the form (key,value), we use a pre-defined class called "dict".

=>"dict" is one of the pre-defined class and treated as dict Catagory Data Type

=>The elements of dict must be organized in the form curly braces { } and (key,value) must separated by comma.

=>An object of dict maintains insertion Order.

=>On the object of dict, we can't perform Indexing and Slicing Operations.

=>An object of dict belongs to mutable. In otherwords , The Values of Key are belongs to immutable and Values of Value are belongs to mutable

=>By using dict class , we can create two types of dict objects. They are

- a) empty dict
 - b) non-empty dict

a) An empty dict is one, which does not contain any elements and whose length is 0.

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

```
dictobj[Key1]=Value1  
dictobj[Key2]=Value2
```

dictobj[Key-n]=Value-n

b) A non-empty dict is one, which contains elements and whose length is >0.

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

Here Key1,Key2...Key-n are called Keys and They are Unique and they can be either Strs or Numerics

Here Val1,Val2...val-n are called Values and They may be Unique or duplicate and they can be either Strs or Numerics

Examples:

```
>>> d1={}
>>> print(d1,type(d1))-----{} <class 'dict'>
>>> len(d1)-----0
>>> d2={10:1.2,20:2.5,30:2.5,40:4.5}
>>> print(d2,type(d2))-----{10: 1.2, 20: 2.5, 30: 2.5, 40: 4.5} <class 'dict'>
>>> len(d2)-----4
>>>
>>> d3=dict()
>>> print(d3,type(d3), len(d3))----- {} <class 'dict'> 0
>>> d1={"Rossum":"Python","Ritche":"C", "Gosling":"Java"}
>>> print(d1,type(d1))---{'Rossum': 'Python', 'Ritche': 'C', 'Gosling': 'Java'} <class 'dict'>

>>> d2={10:"Apple",20:"Mango",30:"Kiwi",40:"Sberry"}
>>> print(d2)---{10: 'Apple', 20: 'Mango', 30: 'Kiwi', 40: 'Sberry'}
>>>
>>> print(d2[0])-----KeyError: 0
>>> print(d2[10])-----Apple
>>> print(d2[40])-----Sberry
>>> print(d2[400])-----KeyError: 400
>>>
>>> d1={}
>>> print(d1,type(d1),len(d1), id(d1))---{} <class 'dict'> 0 2299637840384
>>> d1[100]="Rossum"
>>> d1[101]="Ritche"
>>> d1[102]="Travis"
>>> d1[103]="MCKinney"
>>> print(d1,type(d1),len(d1), id(d1))---{100: 'Rossum', 101: 'Ritche', 102: 'Travis', 103:
>>> d1[100]="Guido"
>>> print(d1,type(d1),len(d1), id(d1))---{100: 'Guido', 101: 'Ritche', 102: 'Travis', 103:
=====X=====
```

pre-defined functions in dict data type

=>dict object contains the following pre-defined function to perform Various Operations.

1) clear()

=>Syntax:- dictobj.clear()

=>This function removes all the (key,Value) from dict object

=>When we call clear() upon empty dict object then we get None

Examples:

```
>>> d1={ 10:1.2,20:2.3,40:5.6,50:1.2}
>>> print(d1,type(d1), id(d1))-----{ 10: 1.2, 20: 2.3, 40: 5.6, 50: 1.2} <class 'dict'>
1228171857856
>>> d1.clear()
>>> print(d1,type(d1), id(d1))-----{ } <class 'dict'> 1228171857856
>>> print(d1.clear())-----None
```

2) copy()

=>Syntax: dictobj2=dictobj1.copy()

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

Examples:

```
>>> d1={ 10:1.2,20:2.3,40:5.6,50:1.2}
>>> print(d1,type(d1), id(d1))----{ 10: 1.2, 20: 2.3, 40: 5.6, 50: 1.2} <class 'dict'>
1228176102528
>>> d2=d1.copy()
>>> print(d2,type(d2), id(d2))----{ 10: 1.2, 20: 2.3, 40: 5.6, 50: 1.2} <class 'dict'>
1228171857856
```

3) pop()

=>Syntax: dictobj.pop(Key)

=>This Function is used removing (Key,Value) from non-dict object

=>if we call this function on empty dict object we get KeyError

Examples:

```
>>> d1={ 10:1.2,20:2.3,40:5.6,50:1.2}
>>> print(d1,type(d1), id(d1))----{ 10: 1.2, 20: 2.3, 40: 5.6, 50: 1.2} <class 'dict'>
1228176103168
>>> d1.pop(20)-----2.3
>>> print(d1,type(d1), id(d1))----{ 10: 1.2, 40: 5.6, 50: 1.2} <class 'dict'> 1228176103168
```

```
>>> d1.pop(40)----5.6
>>> print(d1,type(d1), id(d1))--{ 10: 1.2, 50: 1.2} <class 'dict'> 1228176103168
>>> d1.pop(10)-----1.2
>>> print(d1,type(d1), id(d1))--{ 50: 1.2} <class 'dict'> 1228176103168
>>> d1.pop(50)-----1.2
>>> d1.pop(150)-----KeyError: 150
```

4) popitem()

=>Syntax: dictobj.popitem()

=>This Function is used removing last entry of (Key,Value) from non-dict object

=>if we call this function on empty dict object we get KeyError

Examples:

```
>>> d1={ 10:1.2,20:2.3,40:5.6,50:1.2}
>>> print(d1,type(d1), id(d1))--{ 10: 1.2, 20: 2.3, 40: 5.6, 50: 1.2} <class 'dict'> 1228171857920
>>> d1.popitem()--(50, 1.2)
>>> print(d1,type(d1), id(d1))--{ 10: 1.2, 20: 2.3, 40: 5.6} <class 'dict'> 1228171857920
>>> d1.popitem()--(40, 5.6)
>>> print(d1,type(d1), id(d1))--{ 10: 1.2, 20: 2.3} <class 'dict'> 1228171857920
>>> d1.popitem()--(20, 2.3)
>>> print(d1,type(d1), id(d1))--{ 10: 1.2 } <class 'dict'> 1228171857920
>>> d1.popitem()--(10, 1.2)
>>> print(d1,type(d1), id(d1))--{ } <class 'dict'> 1228171857920
>>> d1.popitem()---KeyError: 'popitem(): dictionary is empty'
>>> { }.popitem()----KeyError: 'popitem(): dictionary is empty'
>>> dict().popitem()---KeyError: 'popitem(): dictionary is empty'
```

5) keys()

=>Syntax: Varname=dictobj.keys()

(OR)

dictobj.keys()

=>This Function is used for obtaining values of Key.

Examples:

```
>>> d1={ 10:"Python ",20:"Data Sci",30:"Django",40:"Java" }
>>> print(d1,type(d1))-----{ 10: 'Python', 20: 'Data Sci', 30: 'Django', 40: 'Java'} <class 'dict'>
>>> d1.keys()-----dict_keys([10, 20, 30, 40])
>>> kvs=d1.keys()
>>> print(kvs)-----dict_keys([10, 20, 30, 40])
>>> for k in kvs:
...     print(k)
...
...
```

```

10
20
30
40
>>> for k in d1.keys():
...     print(k)
...
10
20
30
40

```

NOTE:

```

>>> d1={ 10:"Python",20:"Data Sci",30:"Django",40:"Java"}
>>> print(d1,type(d1))-----{ 10: 'Python', 20: 'Data Sci', 30: 'Django', 40: 'Java'} <class
'dict'>
>>> k=d1.keys()
>>> print(k)-----dict_keys([10, 20, 30, 40])
>>> print(k,type(k))-----dict_keys([10, 20, 30, 40]) <class 'dict_keys'>
>>> l=list(k)
>>> print(l,type(l))-----[10, 20, 30, 40] <class 'list'>
>>> print(l[0])-----10
-----OR-----
>>> d1={ 10:"Python",20:"Data Sci",30:"Django",40:"Java"}
>>> print(d1,type(d1))-----{ 10: 'Python', 20: 'Data Sci', 30: 'Django', 40: 'Java'} <class 'dict'>
>>> list(d1.keys())[0]-----10
-----
```

6) values()

Syntax: Varname= dictobj.values()
(OR)
dictobj.values()

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

Examples:

```

>>> d1={ 10:"Python",20:"Data Sci",30:"Django",40:"Java"}
>>> print(d1,type(d1))-----{ 10: 'Python', 20: 'Data Sci', 30: 'Django', 40: 'Java'} <class
'dict'>
>>> d1.values()-----dict_values(['Python', 'Data Sci', 'Django', 'Java'])
>>> vs=d1.values()
>>> print(vs)-----dict_values(['Python', 'Data Sci', 'Django', 'Java'])
>>> for v in vs:
...     print(v)
...
Python
Data Sci

```

```
Django
Java
>>> for v in d1.values():
...     print(v)
        Python
        Data Sci
        Django
        Java
-----
7) items()
-----
Syntax:- varname=dictobj.items()
          (OR)
          dictobj.items()
=>This Function is used for obtaing (Key,Value) from dict object in the form of list of tuples.
-----
Examples
-----
>>> d1={ 10:"Python",20:"Data Sci",30:"Django",40:"Java" }
>>> print(d1,type(d1))-----{ 10: 'Python', 20: 'Data Sci', 30: 'Django', 40: 'Java' } <class 'dict'>
>>> d1.items()---dict_items([(10, 'Python'), (20, 'Data Sci'), (30, 'Django'), (40, 'Java')])
>>> kv=d1.items()
>>> print(kv)--dict_items([(10, 'Python'), (20, 'Data Sci'), (30, 'Django'), (40, 'Java')])
-----
>>> for x in kv:
...     print(x)
...
        (10, 'Python')
        (20, 'Data Sci')
        (30, 'Django')
        (40, 'Java')
>>> for k,v in kv:
...     print(k,v)
...
        ...
        10 Python
        20 Data Sci
        30 Django
        40 Java
>>> for k,v in kv:
...     print(k,"-->",v)
...
        ...
        10 --> Python
        20 --> Data Sci
        30 --> Django
        40 --> Java
>>> for k,v in d1.items():
```

```
...     print(k,"-->",v)
...
    10 --> Python
    20 --> Data Sci
    30 --> Django
    40 --> Java
```

8) update()

Examples:

```
>>> d1={ 10:1.2,20:3.4 }
>>> d2={ 30:1.5,40:5.6 }
>>> print(d1,type(d1))-----{ 10: 1.2, 20: 3.4 } <class 'dict'>
>>> print(d2,type(d2))-----{ 30: 1.5, 40: 5.6 } <class 'dict'>
>>> d1.update(d2)
>>> print(d1,type(d1))-----{ 10: 1.2, 20: 3.4, 30: 1.5, 40: 5.6 } <class 'dict'>
>>> print(d2,type(d2))-----{ 30: 1.5, 40: 5.6 } <class 'dict'>
```

```
>>> d1={ 10:1.2,20:3.4 }
>>> d2={ 10:6.5,20:7.6 }
>>> print(d1,type(d1))-----{ 10: 1.2, 20: 3.4 } <class 'dict'>
>>> print(d2,type(d2))-----{ 10: 6.5, 20: 7.6 } <class 'dict'>
>>> d1.update(d2)
>>> print(d1,type(d1))-----{ 10: 6.5, 20: 7.6 } <class 'dict'>
>>> print(d2,type(d2))-----{ 10: 6.5, 20: 7.6 } <class 'dict'>
```

```
>>> d1={ 10:1.2,20:3.4 }
>>> d2={ 30:1.5,10:15.6 }
>>> print(d1,type(d1))-----{ 10: 1.2, 20: 3.4 } <class 'dict'>
>>> print(d2,type(d2))-----{ 30: 1.5, 10: 15.6 } <class 'dict'>
>>> d1.update(d2)
>>> print(d1,type(d1))-----{ 10: 15.6, 20: 3.4, 30: 1.5 } <class 'dict'>
>>> print(d2,type(d2))-----{ 30: 1.5, 10: 15.6 } <class 'dict'>
```

9) get()

=>Syntax: Varname= dictobj.get(Key)
(OR)
dictobj.get(Key)

=>This Function is used for finding value of Value by passing Value of Key
=>If Value of Key does not exist then we get None.

Examples:

```
>>> d1={ "TS":"HYD","AP":"AMVT","KAR ":"BANG","TAMIL ":"CHE"}
>>> print(d1)-----{ 'TS': 'HYD', 'AP': 'AMVT', 'KAR': 'BANG', 'TAMIL': 'CHE'}
>>> d1["TS"]-----'HYD'
>>> d1["TAMIL"]-----'CHE'
>>> d1["AMPT"]-----KeyError: 'AMPT'
>>> d1.get("AP")-----'AMVT'
>>> d1.get("KAR")-----'BANG'
>>> d1.get("SRN")-----
>>> print(d1.get("SRN"))-----None
```

NOTE:

We can get Value of Value by passing Key by using the syntax also.

Syntax: varname=dictobj[Key]

=>Here if Value of Key does not Exist then we get KeyError

MISC Examples:

```
>>> d1={ 10:"Python",20:"Data Sci",30:"Django",40:"Java"}
>>> print(d1,type(d1))-----{ 10: 'Python', 20: 'Data Sci', 30: 'Django', 40: 'Java'} <class 'dict'>
>>> kvs=d1.items()
>>> print(kvs,type(kvs))-----dict_items([(10, 'Python'), (20, 'Data Sci'), (30, 'Django'), (40, 'Java')]) <class 'dict_items'>
>>> l1=list(kvs)
>>> print(l1,type(l1))---[(10, 'Python'), (20, 'Data Sci'), (30, 'Django'), (40, 'Java')] <class 'list'>
>>> l1[0]-----(10, 'Python')
>>> l1[1]-----(20, 'Data Sci')
>>> l1[2]-----(30, 'Django')
>>> l1[-1]-----(40, 'Java')

>>> d1={ 10:"Python",20:"Data Sci",30:"Django",40:"Java"}
>>> print(d1,type(d1))---{ 10: 'Python', 20: 'Data Sci', 30: 'Django', 40: 'Java'} <class 'dict'>
>>> lst=list(d1.items())
>>> print(lst)----[(10, 'Python'), (20, 'Data Sci'), (30, 'Django'), (40, 'Java')]
```

Most Useful case:

```
>>> l1=[(10,"Rizwan"),(20,"Rossum"),(30,"Rajesh"),(40,"Ranjit")]
>>> print(l1,type(l1))----[(10, 'Rizwan'), (20, 'Rossum'), (30, 'Rajesh'), (40, 'Ranjit')] <class 'list'>
>>> d1=dict(l1)
>>> print(d1,type(d1))---{ 10: 'Rizwan', 20: 'Rossum', 30: 'Rajesh', 40: 'Ranjit'} <class 'dict'>
```

Most Useful case

```
> l1=(10,20,30,40)
>>> l2=(1.2,2.3,4.5,1.5)
>>> d1=dict(zip(l1,l2))
>>> print(d1)-----{ 10: 1.2, 20: 2.3, 30: 4.5, 40: 1.5}

-----  
>>> l1=(10,20,30,40)
>>> l2=(1.2,2.3)
>>> d=dict(zip(l1,l2))
>>> print(d,type(d))-----{ 10: 1.2, 20: 2.3} <class 'dict'>

-----  
>>> print(d1,type(d1))
{ 10: ['C', 'CPP'], 20: {'PYTHON': {'Core': 'GUI', 'ADV': 'OOPS'}, 30: 'OUCET'} } <class 'dict'>
>>>
>>> for k,v in d1.items():
...     print(k,"--->",v)
...
    10 ---> ['C', 'CPP']
    20 ---> {'PYTHON': {'Core': 'GUI', 'ADV': 'OOPS'}, 30: 'OUCET'}
>>> len(d1)----2
>>> d1[20]
{'PYTHON': {'Core': 'GUI', 'ADV': 'OOPS'}, 30: 'OUCET'}
>>> type(d1[20])-----<class 'dict'>
>>> len(d1[20])-----2
```

=====

NoneType data type

=====

=>'NoneType' is one the pre-defined class and treated as None type Data type
=> "None" is keyword acts as value for <class,'NoneType'>
=>The value of 'None' is not False, Space , empty ,0
=>An object of NoneType class can't be created explicitly.

Examples:

```
>>> a=None
>>> print(a,type(a))-----None <class 'NoneType'>
>>> a=NoneType()-----NameError: name 'NoneType' is not defined
>>> l1=[]
>>> print(l1.clear())-----None
>>> s1=set()
>>> print(s1.clear())-----None
>>> d1=dict()
>>> print(d1.clear())-----None
>>> d1={ 10:1.2,20:3.4 }
```

```
>>> print(d1.get(100))-----None
```

Number of approaches for development of Programs in Python

=>Definition of Program:

=>Set of Optimized Instructions is called Program.

=>Programs are always developed by Language Programmers for solving Real Time Applications.

=>To solve any Real Time Application, we must write Set of Optimized Instructions and save those Instructions on Some File Name with an extension .py
(FileName.py----->Considered as Python Program)

=>In Python Programming, we can develop any Program with Approaches. They are

1. Interactive Mode Approach
 2. Batch Mode Approach
-

1. Interactive Mode Approach

=>In This approach, the programmer can issue one Instruction at a time and gets One Output at a time

=>This Approach is more useful to test one Instruction at a time.

=>This Approach is not useful for Developing Code for Big Problems and more over we are unable to save the instructions.

Examples:

```
>>> a=10
>>> b=20
>>> c=a+b
>>> print(a)-----10
>>> print(b)-----20
>>> print(c)-----30
```

Softwares---- Python Command Comand (Will come on the Installation Python)
Python IDLE Shell (Will come on the
Installation Python)

2. Batch Mode Approach

=>The Process of defining Group of Instructions under one editor and save those instructions on some file name with an extension .py (FileName.py--Source Code) is called Batch Mode Approach

=>This Approach is more useful for solving Big Problems

Software:

- Python IDLE Shell (Will come on the Installation Python)
- Edit Plus (Install Explicitly)
- Pycharm
- Jupiter NoteBook
- Spider
- Visual Studio (VS Code)
- Google Clab
- Atom
- sub lime Text....etc

```
#Program for mul of two numbers
#MulEx.py---File Name
a=float(input("Enter Value a:"))
b=float(input("Enter Value b:"))
c=a*b
print("-----")
print("Val of a=",a)
print("Val of b=",b)
print("Mul=",c)
print("-----")
```

=>To run the Python Program from Windows Command Prompt , we use "python" or "py"
=>Syntax:

E:\KVR-PYTHON-11am\Batch-Mode>python MulEx.py
(OR)
E:\KVR-PYTHON-11am\Batch-Mode>py MulEx.py

```
=====
#Program for mul of two numbers
a=float(input("Enter Value a:"))
b=float(input("Enter Value b:"))
c=a*b
print("-----")
print("Val of a=",a)
print("Val of b=",b)
print("Mul=",c)
print("-----")
```

```
#Program for computing sum of two numbers
a=10
b=20
```

```
c=a+b
print("value of a=",a)
print("Value of b=",b)
print("sum=",c)


---


#program for computing sum of two numbers
a=float(input("Enter First value:"))
b=float(input("Enter Second value:"))
c=a+b
print("=====Result=====")
print("val of a=",a)
print("val of b=",b)
print("Sum=",c)
print("=====")
```

=====

Display the Result of Python Program on the console

=====

=>To display the result of Python Program on the console, we use a pre-defined Function called print().

=>print() is one of the pre-defined Function used for displaying the result of Python Program on the console

=>print() contains the following Syntaxes

Syntax-1:

=>Syntax: print(value)

(OR)

print(Variable Name)

(OR)

print(Val1,Val2....val-n)

(OR)

print(var1,var2.....var-n)

=>This Syntax used for Displaying only Values or Values of variables.

Examples:

```
>>> sno=10
>>> sname="Rossum"
>>> sub="Python"
>>> print(sno)-----10
>>> print(sname)-----Rossum
>>> print(sub)-----Python
>>> print(sno,sname,sub)-----10 Rossum Python
>>> print(100,200,300)-----100 200 300
```

Syntax-2

Syntax: print(Message)
(OR)
print(Message1,Message2,....,Message-n)

=>This Syntax display only Messges.

Examples:

```
>>> print("hello Python world")-----hello Python world  
>>> print('hello Python world')-----hello Python world
```

Syntax-3

Stntax: print(Values cum Messages)
(OR)
print(Messages cum Values)

=>This Syntax displayes the values cum messages or Messages cum Values

Examples:

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

Syntax-4

Stntax: print(Values cum Messages with format())
(OR)
print(Messages cum Values with format())

Examples:

```
-----  
>>> a=10  
>>> b=20  
>>> c=a+b  
>>> print("Sum={ }".format(c))-----Sum=30  
>>> print(" { } is the sum".format(c))----30 is the sum  
>>> print("sum of ",a," and ",b,"=",c)----sum of 10 and 20 = 30  
>>> print("sum of { } and { }={ }".format(a,b,c))---sum of 10 and 20=30  
>>> sno=10  
>>> sname="Rossum"  
>>> print(" '{ }' is a student and roll number is { }".format(sname,sno))-- 'Rossum' is a student
```

Syntax-5

Stntax: print(Values cum Messages with format specifiers)
(OR)
print(Messages cum Values with format specifiers)

Examples:

```
-----  
>>> a=10  
>>> b=20  
>>> c=a+b  
>>> print("Sum=%d" %c)-----Sum=30  
>>> print("%d is the sum" %c)----30 is the sum  
>>> print("Sum of %d and %d = %d" %(a,b,c))----Sum of 10 and 20 = 30  
>>> sno=10  
>>> sname="Elite Elderson"  
>>> marks=33.33  
>>> print("My Number is %d and name is '%s' and Marks=%f" %(sno,sname,marks))  
        My Number is 10 and name is 'Elite Elderson' and Marks=33.330000  
>>> print("My Number is %d and name is '%s' and Marks=%0.2f" %(sno,sname,marks))  
My Number is 10 and name is 'Elite Elderson' and Marks=33.33  
>>> print("My Number is %d and name is '%s' and Marks=%0.1f" %(sno,sname,marks))  
        My Number is 10 and name is 'Elite Elderson' and Marks=33.3  
-----  
>>> a=1.2
```

```
>>> b=20
>>> c=a+b
>>> print("sum of %f and %f=%f".format(a,b,c))---sum of %f and %f=%f
>>> print("sum of %f and %f=%f" %(a,b,c) )---sum of 1.200000 and 20.000000=21.200000
>>> print("sum of %0.2f and %0.2f=%0.3f" %(a,b,c) )--sum of 1.20 and 20.00=21.200
>>> t=(10,"Mr.Crazy",33.33,"Sathish")
>>> print(t)-----(10, 'Mr.Crazy', 33.33, 'Sathish')
>>> print("content of t=%s" %t)-----content of t= (10, 'Mr.Crazy', 33.33, 'Sathish')
>>> print("content of t=%s" %format(t))---content of t=(10, 'Mr.Crazy', 33.33, 'Sathish')
>>> print("content of t=%s" %str(t))---content of t=(10, 'Mr.Crazy', 33.33, 'Sathish')
```

Syntax-6:

Syntax: print(Value cum Message, end=" ")
=>This syntax displays the data in same Line

Examples:

```
>>> lst=[10,20,30,40,50,60]
>>> for val in lst:
...     print(val)
...
10
20
30
40
50
60
>>> for val in lst:
...     print(val,end=" ")----- 10 20 30 40 50 60
>>> for val in lst:
...     print(val,end="-->")---- 10-->20-->30-->40-->50-->60-->
>>> lst=[10,20,30,40,50,60]
>>> for val in lst:
...     print("{ }".format(val), end="\t")    10    20    30    40    50    60    >>>
>>> lst=[10,20,30,40,50,60]
>>> for val in lst:
...     print("%d" %val, end=" ")----10 20 30 40 50 60
```

Reading the data or input from Key Board

=>To read the data from Keyboard, we use Two pre-defined Functions. They are

1. `input()`
 2. `input(Message)`
-

1) `input()`

=>This Function is used for Reading any type of data from Key board in the form of str type only.

=>Syntax:- `varname=input()`

=>Here `input()` reads the value in the form str and place that value in varname.

=>The value of str can type casted to any other types by using Type Casting functions.

Examples

```
#Program for accepting two integer values and multiply them
#MulExample3.py
print("Enter two Values:")
a=float( input() )
b=float( input() )
#Multiply them
c=a*b
print("Mul({},{})={ }".format(a,b,c))
```

2) `input(Message)`

=>This Function is used for Reading any type of data from Key board in the form of str type only and with Function additionally we can provide User-Prompting Message.

=>Syntax: `varname=input(Message)`

=>here Message Represents User-Prompting Message.

=>Here `input(Message)` reads the value in the form str and place that value in varname by giving User-Prompting Message.

=>The value of str can type casted to any other types by using Type Casting functions.

Examples:

```
#MulExample6.py
a=float(input("Enter First value:"))
b=float(input("Enter Second value:"))
c=a*b
print("Mul({},{})={ }".format(a,b,c))
```

```
#Program for accepting two integer values and multiply them
#MulExample1.py
print("Enter First Value:")
s1=input()
print("Enter Second Value:")
s2=input()
#convert s1 and s2 into float
a=float(s1)
b=float(s2)
#Multiply them
c=a*b
print("Mul({},{})={}".format(a,b,c))


---


#Program for accepting two integer values and multiply them
#MulExample2.py
print("Enter two Values:")
s1=input()
s2=input()
#convert s1 and s2 into float
a=float(s1)
b=float(s2)
#Multiply them
c=a*b
print("Mul({},{})={}".format(a,b,c))


---


#Program for accepting two integer values and multiply them
#MulExample3.py
print("Enter two Values:")
a=float( input() )
b=float( input() )
#Multiply them
c=a*b
print("Mul({},{})={}".format(a,b,c))


---


#Program for accepting two integer values and multiply them
#MulExample4.py
print("Enter two Values:")
c=float( input() )*float( input() )
print("Mul={}{}".format(c))


---


#Program for accepting two integer values and multiply them
#MulExample5.py
print("Enter two Values:")
a=float( input() )
b=float( input() )
print("Mul({},{})={}{},{}{}".format(a,b, a*b))


---


#Program for accepting two integer values and multiply them
#MulExample6.py
```

```
s1=input("Enter First value:")
s2=input("Enter Second value:")
a=float(s1)
b=float(s2)
c=a*b
print("Mul({},{})={}".format(a,b,c))


---


#Program for accepting two integer values and multiply them
#MulExample6.py
a=float(input("Enter First value:"))
b=float(input("Enter Second value:"))
c=a*b
print("Mul({},{})={}".format(a,b,c))


---


#Program for accepting two integer values and multiply them
#MulExample8.py
print("Mul={ }".format(float(input("Enter First Value:")) * float(input("Enter Second Value:"))))


---


#Program cal area of rectangle
l=float(input("Enter Length:"))
b=float(input("Enter breadth:"))
ar=l*b
print("====")
print("Length={ }".format(l))
print("Breadth={ }".format(b))
print("Area of Rect={ }".format(ar))
print("====")


---


#program cal area of Circle
r=float(input("Enter Radious:"))
ac=3.14*r*r
print("====")
print("Radious={ }".format(r))
print("Area of Circl={ }".format(ac))
print("-----OR-----")
print("Area of Circle=%0.2f" % ac)
print("-----OR-----")
print("Area of Circle=",round(ac,2))
print("-----OR-----")
print("Area of Circle={ }".format(round(ac,2)))
print("====")
```

Operators and Expressions in python

=>An Operator is a symbol which is used to perform certain operations.

=>If any operator connected with two or more Objects / Variables then it is called Expression.

=>An Expression is a collection of objects or variables connected with Operators.

=>In python Programming, we have 7 types of Operators. They are

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

1. Arithmetic Operators

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

=>If Two or More Objects or Variables connected with Arithmetic Operators then it is called Arithmetic Expressions.

=>In Python programming, we have 7 types of Arithmetic Operators. They are given in the following Table.

SLNO	SYMBOL	MEANING	EXAMPLES a=10 b=3
1.	+	Addition	print(a+b)----13
2.	-	Subtract	print(a-b)-----7
3.	*	Multiplication	print(a*b)---30
4.	/	Division (Float Quotient)	print(a/b)--3.33
5.	//	Floor Division (Integer Quotient)	print(a//b)--3
6.	%	Modulo Division (Reminder after Division)	print(a%b)--1
7.	**	Exponentiation	print(a**b)--100

```
=====
#Program for demonstarting Arithmetic Operators
#Aop.py
a=int(input("Enter Value of a:"))
b=int(input("Enter Value of b:"))
print("-"*50)
print("Arithemtic Operators")
print("*"*50)
print("\tsum({},{}]={}.format(a,b,a+b))")
print("\n\tsub({},{}]={}.format(a,b,a-b))")
print("\n\tmul({},{}]={}.format(a,b,a*b))")
print("\n\tDiv({},{}]={}.format(a,b,a/b))")
print("\n\tFloor Div({},{}]={}.format(a,b,a//b))")
print("\n\tMod({},{}]={}.format(a,b,a%b))")
print("\n\tExp({},{}]={}.format(a,b,a**b))")
print("#"*50)
```

```
#Program for cal square root of a given number
#sqrrex.py
n=float(input("Enter a number for can square root:"))
res=n**(1/2)
print("square root({})={}".format(n,res))
```

2. Assigment Operator

=>The purpose of assignment operator is that " To assign or transfer Right Hand Side (RHS)

Value / Expression Value to the Left Hand Side (LHS) Variable "

=>The Symbol for Assigment Operator is single equal to (=).

=>In Python Programming,we can use Assigment Operator in two ways.

 1. Single Line Assigment

 2. Multi Line Assigment

1. Single Line Assigment:

=>Syntax: LHS Varname= RHS Value
 LHS Varname= RHS Expression

=>With Single Line Assigment at a time we can assign one RHS Value / Expression to the single LHS Variable Name.

Examples:

```
>>> a=10
>>> b=20
>>> c=a+b
>>> print(a,b,c)-----10 20 30
```

2. Multi Line Assigment:

=>Syntax: Var1,Var2.....Var-n= Val1,Val2....Val-n

Var1,Var2.....Var-n= Expr1,Expr2...Expr-n

Here The values of Val1, Val2... Val-n are assigned to Var1,Var2...Var-n Respectively.

Here The values of Expr1, Expr2...Expr-n are assigned to Var1,Var2...Var-n Respectively.

Examples:

```
-----  
>>> a,b=10,20  
>>> print(a,b)-----10 20  
>>> c,d,e=a+b,a-b,a*b  
>>> print(c,d,e)-----30 -10 200  
=====X=====
```

Relational Operators

=>The purpose of Relational Operators is that "To Compare Two or More Values "

=>If two or more Variables or Objects connected with Relational Operator then it is Relational Expression.

=>The Result of Relational Expression is either True or False.

=>The Relational Expression is called Test Condition

=>In Python Program, The Relational Operators are classified into 6 types. They are given in the following table

SLNO	SYMBOL	MEANING	EXAMPLE
1.	>	Greater than	print(10>20)---False
2.	<	Less Than	print(10<20)---True
3.	==	Equality (Double Equal to)	print(10==10)---True print(10==5)---False
4.	!=	Not Equal to	print(10!=20)---True
5.	>=	Greater Than or Equal to	print(10>=5)---True print(10>=11)---False

6.	<=	Less Than or Equal to	print(-30<=-34)False print(-15>=-16)-True
----	----	--------------------------	----------------------------------------------

```
=====
=====
#Program for performing all type of arithmetic Operations by using Multi line assigment
#MultiLineaop.py
a,b=int(input("Enter Value of a:")), int(input("Enter Value of b:")) # Multi line assigment-
reading
sum,sum,mul,div,fdiv,mod,exp=a+b,a-b,a*b,a/b,a//b,a%b,a**b # Multi line assigment--cal
print("=*50)
print("Sum=",sum)
print("Sub=",sub)
print("Mul=",mul)
print("Div=",div)
print("Floor Div=",fdiv)
print("Mod=",mod)
print("Exp({},{})={}".format(a,b,exp))
print("=*50)
```

```
#program for demonstaring Relational Operators
#relOprEx.py
a,b=float(input("Enter Value of a:")),float(input("Enter Value of b:"))
print("=*40)
print("\tResults of Relational Operators")
print("=*40)
print("\t{} > {}={}" .format(a,b,a>b))
print("\n\t{} < {}={}" .format(a,b,a<b))
print("\n\t{} == {}={}" .format(a,b,a==b))
print("\n\t{} != {}={}" .format(a,b,a!=b))
print("\n\t{} >= {}={}" .format(a,b,a>=b))
print("\n\t{} <= {}={}" .format(a,b,a<=b))
print("=*40)
```

```
#Program for accepting any two values and swap them
#SwapValues.py
a,b=input("Enter Value of a:"),input("Enter Value of b:")
print("-*50)
print("\nOriginal value of a={}" .format(a))
print("Original value of b={}" .format(b))
print("-*50)
#swapping logic
a,b=b,a # Multi Line assigment
print("\nSwapped value of a={}" .format(a))
print("Swapped value of b={}" .format(b))
print("-*50)
```

Bitwise Operators (Most Imp)

=>Bitwise Operators are Performing the Operation on Integer Data in the form Binary Bits.

=>Bitwise Operators are applicable on Integer Data but not floating point values bcoz floating

values does not have certainty.

=>In Bitwise Operators , First Given Integer Data Converted into Binary data and they starts performing operation Bit by Bit and hence they named Bitwise Operators.

=>In Python Programming, we have 6 types of Bitwise Operators. They are

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

1. Bitwise Left Shift Operator (<<):

Syntax:- varname = Given Number << No. of Bits

=>This Operator Shifts or Popping-off No. of Bits of Given Number from Left Side and add Number of Zeros (depends on No. of Bits) at Right Side.

Examples:

```
>>> print(10<<3)-----80
>>> print(4<<4)-----64
>>> print(8<<3)-----64
>>> print(2<<3)-----16
>>> print(5<<2)-----20
```

2. Bitwise Right Shift Operator (>>):

Syntax:- varname = Given Number >> No. of Bits

=>This Operator Shifts or Popping-off No. of Bits of Given Number from Right Side and add Number of Zeros (depends on No. of Bits) at Left Side.

Examples:

```
>>> print(10>>3)-----1
>>> print(10>>2)-----2
>>> print(12>>2)-----3
>>> print(100>>4)-----6
```

3. Bitwise AND Operator (&)

=>Syntax:- Varname = Var1 & Var2

=>The Functionality of Bitwise AND Operator (&) is expressed in the following Truth table.

Var1	Var2	Var1 & Var2
0	1	0
1	0	0
0	0	0
1	1	1

Examples:

```
>>>a=10----->0000 0000 0000 1010  
>>>b=4----->0000 0000 0000 0100
```

```
>>>c=a&b----->0000 0000 0000 0000  
>>>print(c)----0  
>>> print(7&4)----4  
>>> print(6&10)----2  
>>> 10 & 20-----0  
>>> 10 and 20----20
```

4. Bitwise OR Operator (|)

=>Syntax:- Varname = Var1 | Var2

=>The Functionality of Bitwise OR Operator (|) is expressed in the following Truth table.

Var1	Var2	Var1 Var2
0	1	1
1	0	1
0	0	0
1	1	1

Examples:

```
>>>a=4-----0100  
>>>b=3-----0011
```

```
>>>c=a|b-----0111  
>>>print(c)----7
```

```
>>> print(10|15)-----15  
>>> print(7|3)-----7  
>>> print(2|5)-----7
```

5. Bitwise Complement Operator (~)

=>Bitwise Complement Operator (~) is used obtaining complement of a Given Number.
=>complement of a Given Number= - (Given Number+1)
=>Internally, Bitwise Complement Operator invert the bits (Nothing But 1 becomes 0 and 0 becomes 1--- called 1 's complement)

Examples:

```
>>> a=17  
>>> ~a-----18  
>>> a=-98  
>>> ~a----- 97  
>>> n=200  
>>> ~n----- -201  
>>> n=304  
>>> ~n----- -305
```

Working Examples:

4-----> 0100

~ 4 -----> -(0100+1)

$$\begin{array}{r} 0100 \\ 0001 \\ \hline -0101 \end{array}$$

10----->1010

~ 10 -----> -(1010+1)

$$\begin{array}{r} 1010 \\ 0001 \\ \hline -1011 \end{array}$$

OR

10-----> 1010

~ 10 ----- 0101 (Inverting the bits)

11-----> 1011

1's complement----- 0100 (Inverting the bits)

2's complement---1's complement of 11 + 1

0100

0001

0101

=====

4-----0100
~4-----1011

What is -5 (2's complement 5=1 's complement of 5+1)

5----- 0101

1's complement-----1010

2'2complement= 1's complement +1

1010 +1

1010

0001

1011

6. Bitwise XOR Operator (^)

=>Syntax:- Varname = Var1 ^ Var2

=>The Functionality of Bitwise XOR Operator (^) is expressed in the following Truth table.

Var1	Var2	Var1 ^ Var2
0	1	1
1	0	1
0	0	0
1	1	0

Examples:

>>> a=3

>>> b=4

```

>>> c=a^b
>>> print(c)-----7
>>> print(10^15)----5
>>> print(4^6)-----2
-----
Special Case:
-----
>>>s1={10,20,30,40}
>>>s2={10,15,25}
>>>s3=s1.union(s2)
>>> print(s3)-----{20, 40, 25, 10, 30, 15}
>>> s4=s1|s2 # Bitwise OR
>>> print(s4,type(s4))-----{20, 40, 25, 10, 30, 15} <class 'set'>
-----
>>>s1={10,20,30,40}
>>>s2={10,15,25}
>>> s3=s1.intersection(s2)
>>> print(s3)-----{10}
>>> s4=s1&s2 # Bitwise AND
>>> print(s4,type(s4))-----{10} <class 'set'>
-----
>>> s1={10,20,30,40}
>>> s2={10,15,25}
>>> s3=s1.difference(s2)
>>> print(s3)-----{40, 20, 30}
>>> s4=s1-s2
>>> print(s4)-----{40, 20, 30}
>>> s5=s2-s1
>>> print(s5)-----{25, 15}
-----
>>> s1={10,20,30,40}
>>> s2={10,15,25}
>>> s3=s1.symmetric_difference(s2)
>>> print(s3)-----{40, 15, 20, 25, 30}
>>> s4=s1^s2 # Bitwise XOR
>>> print(s4,type(s4))-----{40, 15, 20, 25, 30} <class 'set'>

```

Special Examples

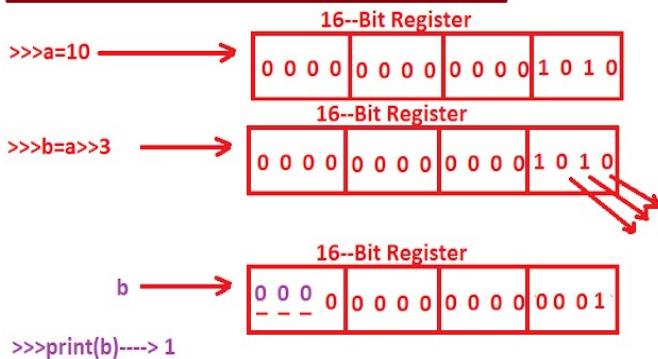
```

>>>a=3
>>>b=4
>>>print(a,b)--- 3  4
>>>a=a^b
>>>b=a^b
>>>a=a^b
>>>print(a,b)---- 4  3

```

2) Bitwise Right Shift Operator (>>)

Syntax: varname= Given Data >> No. of Bits



Formula for Bitwise Right Shift Operator (>>)

Syntax: Varname= Given data >> No. of Bits

Given data
Varname= -----
No. of Bits
2

>>>print(10>>3)----> $\frac{10}{2^3}$

$$\text{Ans: } 1 = \frac{10}{8} \text{ use //}$$

>>>print(12>>2)---->3

>>> print(10>>3)-----1

>>> print(10>>2)-----2

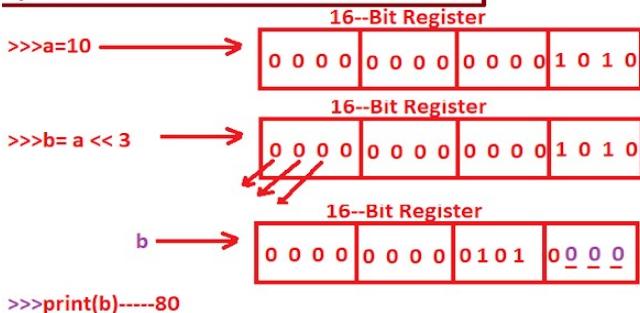
>>> print(12>>2)-----3

>>> print(100>>4)-----6

5

1. Bitwise Left Shift Operator (<<):

Syntax: varname=Given Data<<No.of Bits



Formula for Bitwise Left Shift :

Varname= Given data <<No. of Bits
No. of Bits
varname= Given data x 2

print(10<<3)----> 10×2^3

-- 80

print(4<<4) -----> 64

Membership Operators

- =>The purpose of Membership Operators is that "To Check the existence of specific value in Iterable object".
- =>An Iterable Object is one which contains Two or More Number of values
- =>Sequece Types (str,bytes,bytearray,range) , List (list, tuple) Types , set (set , frozenset) Types , and dict type(dict) are comes under Iterable object.
- =>In Python Programming, we have two type of Membership Operators. They are

- 1) in
 - 2) not in
-

1) in

Syntax: Value in Iterable Object

- =>"in" operator returns True provided "Value" present in Iterable Object
 - =>"in" operator returns False provided "Value" present not in Iterable Object
-

2) not in

Syntax: Value not in Iterable Object

- =>"not in" operator returns True provided "Value" not present in Iterable Object
 - =>"not in" operator returns False provided "Value" present in Iterable Object
-

Examples:

```
>>> s="PYTHON"
>>> print(s)
PYTHON
>>>
>>>
>>> s="PYTHON"
>>> "P" in s
True
>>> "O" in s
True
>>> "O" not in s
False
>>> "k" not in s
True
>>> "k" in s
False
>>> "p" in s
False
>>> "p" not in s
True
>>> not ("p" not in s)
False
```

```
>>>
>>>
>>>
>>>
>>> s="PYTHON"
>>> print(s)
PYTHON
>>> "PYT" in s
True
>>> "PYTK" in s
False
>>> "PYTK" not in s
True
>>>
>>>
>>> s="PYTHON"
>>> print(s)
PYTHON
>>> "PON" in s
False
>>> "PYN" in s
False
>>> "PYN" not in s
True
>>>
>>>
>>> s="PYTHON"
>>> print(s)
PYTHON
>>> "NOH" in s
False
>>> "HON" not in s
False
>>> "NOH" in s
False
>>> "OTP" in s
False
>>> "OTP" not in s
True
>>>
>>>
>>> s="PYTHON"
>>> print(s)
PYTHON
>>> "NOH" in s[::-1]
True
>>> "OTP" not in s[::-2]
True
>>> s[::-2]
'NHY'
```

```

>>> s="PYTHON"
>>> print(s)
PYTHON
>>> s in s
True
>>> s in s[::-1]
False
>>> s="MADAM"
>>> s in s[::-1]
True
-----
>>> s="MADAM"
>>> s not in s[::-1][::]
False
-----
>>> lst=[10,"Rossum",True,45,2+3j]
>>> print(lst)
[10, 'Rossum', True, 45, (2+3j)]
>>> 10 in lst
True
>>> True in lst
True
>>> False not in lst
True
>>> False in lst
False
-----
>>> lst=[10,"Rossum",True,45,2+3j]
>>> print(lst)
[10, 'Rossum', True, 45, (2+3j)]
>>> "sum" in lst
False
>>> "sum" in lst[1]
True
>>> lst[1] in lst[-4][::]
True
>>> lst[1][::-2] not in lst[-4][::-2]-----False
-----
>>> lst=[10,"Rossum",True,45,2+3j]
>>> lst[-1].real in lst[-1]-----TypeError: argument of type 'complex' is not iterable
-----
```

4. Logical Operators

- =>The purpose of Logical Operators is that "To Connect two or more Relational Expressions".
- =>If two or more Relational Expressions connected with Logical Operators then it is called Logical Expression or Compound Conditions(Multiple condition).
- =>The result of Logical Expression or Compound Conditions is either True or False.

=>In Python Programming, we have three types of Logical Operators. They are given in the following Table.

SLNO	SYMBOL	MEANING
1 ANDing	and	Physical
2.	or	Physical ORing
3.	not	-----

1) and :

=>The Functionality of "and" operator is described in the following Truth Table.

Rel Expr2	Rel Expr1	RelExpr2	RelExpr1 and
	False	False	False
	True	False	False
	False	True	False
	True	True	True

Examples:

```
>>> print(100>20 and 20>4)-----True
>>> print(100>200 and 20>4)-----False----Short Circuit Evaluation
>>> print(-100>200 and 20>4 and 10>2)--False---Short Circuit Evaluation
```

=>Short Circuit Evaluation (or) Lazy Evaluation in the case of "and"

In the case of "and" operator, if First Relational Expression result is False Then PvM will not evaluate rest of the Relational Expression and total Logical Expression result will be considered as False. This process is called Short Circuit Evaluation (or) Lazy Evaluation of "and" operator.

2) or :

=>The Functionality of "or" operator is described in the following Truth Table.

Expr2	Rel Expr1	RelExpr2	RelExpr1 or Rel
	False	False	False
	True	False	True
	False	True	True
	True	True	True

Examples:

```
>>> print(10>2 or 10>4)-----True----Short Circuit Evaluation
>>> print(10<2 or 20>3 or 5>50)-----True----Short Circuit Evaluation
>>> print(10>2 or 20==3 or 5!=50)-----True---Short Circuit Evaluation
>>> print(10==2 or 20==3 or 5>=50)-----False
```

=>Short Circuit Evaluation (or) Lazy Evaluation in the case of "or"

In the case of "or" operator, if First Relational Expression result is True Then PvM will not evaluate rest of the Relational Expressions and total Logical Expression result will be considered as True. This process is called Short Circuit Evaluation (or) Lazy Evaluation of "or" operator.

3) not operator:

=>The Functionality of "not" operator is described in the following Truth Table.

Rel Expr1	not RelExpr1
False	True
True	False

Examples:

```
>>> a=10
>>> b=20
>>> a==b-----False
>>> not(a==b)-----True
>>> print( not (10==2 or 20==3 or 5>=50))-----True
```

```
>>> a=True
>>> not a-----False
>>> not False-----True
```

Special Examples:

```
-----  
>>> 100>20 and 100>40-----True  
>>> 100>20 or 100>40-----True  
=====
```

```
>>> 100 and -100  
-100  
>>> 100 and 0  
0  
>>> -100 and -225  
-225  
>>> 0 and 100  
0  
>>>  
>>> 100 and -1234567  
-1234567  
>>> 100 and 0  
0  
>>> 0 and 345  
0  
>>> 0 and 0  
0  
>>>  
>>> 100 or 200  
100  
>>> -100 or -223  
-100  
>>> 0 or -223  
-223  
>>> 0 or 0  
0  
>>> not (0 or 0)  
True
```

```
>>> 100 and -100  
-100  
>>> 0 and 10  
0  
>>> 10 and 20  
20  
>>> 0 and -100  
0  
>>> 100 and 0  
0  
>>> 100 and 200 and -100  
-100  
>>> 100 and 200 and 0  
0  
>>> 100 and -200 and 0 and 234  
0
```

```

>>>
>>>
>>> "KVR" and "PYTHON"
'PYTHON'
>>> "KVR" and 0
0
>>> "KVR" and "Python" and True
True
>>>
>>>
>>>
>>> 100 or 200
100
>>> 100 or 0
100
>>> 0 or 200
200
>>> 0 or 300 or 300 or 100
300
>>>
>>>
>>> "KVR" or "PYTHON"
'KVR'
>>>
>>>
>>>
>>> 10 or 20 and 30 or 30 and 450
10
>>> 10 and 20 or 300 and 450 and 0 or 23
20

>>> 100 and "or"
'or'
>>> "and" and "or"
'or'
>>> "and" or "or"
'and'
>>>
=====X=====
=====
```

Identity Operators (Python Command Prompt)

=>The purpose of Identity Operators is that "To Check the Memory Address of Two Objects".
=>In Python Programming, we have two types of Identity Operators. They are

1. is
2. is not

1) is

Syntax:- object1 is object2
=>"is" operator Returns True provided Memory address of Object1 and Object2 are SAME
=>"is" operator Returns False provided Memory address of Object1 and Object2 are
DIFFERENT

2) is not

Syntax:- object1 is not object2
=>"is not" operator Returns True provided Memory address of Object1 and Object2 are
DIFFERENT
=>"is" operator Returns False provided Memory address of Object1 and Object2 are SAME

Examples:

```
>>> a=None
>>> b=None
>>> print(a,id(a))-----None 140709648996344
>>> print(b,id(b))-----None 140709648996344
>>> a is b-----True
>>> a is not b-----False

-----  
>>> d1={10:"Apple",20:"Mango",30:"CApple"}
>>> d2={10:"Apple",20:"Mango",30:"CApple"}
>>> print(d1,id(d1))-----{10: 'Apple', 20: 'Mango', 30: 'CApple'} 1938668998592
>>> print(d2,id(d2))-----{10: 'Apple', 20: 'Mango', 30: 'CApple'} 1938668998656
>>> d1 is d2----False
>>> d1 is not d2----True

-----  
>>> s1={10,20,30,40}
>>> s2={10,20,30,40}
>>> print(s1,id(s1))-----{40, 10, 20, 30} 1938669202432
>>> print(s2,id(s2))-----{40, 10, 20, 30} 1938673175904
>>> s1 is s2-----False
>>> s1 is not s2----True
>>> fs1=frozenset(s1)
>>> fs2=frozenset(s1)
>>> print(fs1,id(fs1))-----frozenset({40, 10, 20, 30}) 1938673176352
>>> print(fs2,id(fs2))-----frozenset({40, 10, 20, 30}) 1938673177696
>>> fs1 is fs2-----False
>>> fs1 is not fs2----True
```

```
>>> t1=(10,20,30)
>>> t2=(10,20,30)
>>> print(t1,id(t1))-----(10, 20, 30) 1938669461184
>>> print(t2,id(t2))-----(10, 20, 30) 1938673242496
>>> t1 is t2-----False
>>> t1 is not t2-----True
>>> l1=[10,"Python","R"]
>>> l2=[10,"Python","R"]
>>> print(l1,id(l1))-----[10, 'Python', 'R'] 1938673238208
>>> print(l2,id(l2))-----[10, 'Python', 'R'] 1938669045952
>>> l1 is l2-----False
>>> l1 is not l2-----True
-----
>>> r1=range(10)
>>> r2=range(10)
>>> print(r1,id(r1))-----range(0, 10) 1938669658224
>>> print(r2,id(r2))-----range(0, 10) 1938669663312
>>> r1 is r2-----False
>>> r1 is not r2-----True
>>> b1=bytes([10,20,30])
>>> b2=bytes([10,20,30])
>>> print(b1,id(b1))-----b'\n\x14\x1e' 1938669663408
>>> print(b2,id(b2))-----b'\n\x14\x1e' 1938669663456
>>> b1 is b2-----False
>>> b1 is not b2-----True
>>> ba1=bytearray((10,20,123))
>>> ba2=bytearray((10,20,123))
>>> print(ba1,id(ba1))-----bytearray(b'\n\x14{') 1938673243440
>>> print(ba2,id(ba2))-----bytearray(b'\n\x14{') 1938673243632
>>> ba1 is ba2-----False
>>> ba1 is not ba2-----True
MOST IMP
>>> s1="PYTHON"
>>> s2="PYTHON"
>>> print(s1,id(s1))-----PYTHON 1938673243696
>>> print(s2,id(s2))-----PYTHON 1938673243696
>>> s1 is s2-----True
>>> s1 is not s2-----False
>>> s1="INDIA"
>>> s2="INDIA"
>>> s1 is s2-----True
>>> s1 is not s2-----False
>>> s1="INDIA"
>>> s2="INDia"
>>> s1 is s2-----False
>>> s1 is not s2-----True
-----
>>> a=2+3j
>>> b=2+3j
```

```
>>> print(a,id(a))-----(2+3j) 1938668707664
>>> print(b,id(b))-----(2+3j) 1938668707696
>>> a is b-----False
>>> a is not b-----True
>>> a=True
>>> b=True
>>> print(a,id(a))-----True 140709648943976
>>> print(b,id(b))-----True 140709648943976
>>> a is b-----True
>>> a is not b-----False
>>> a=1.2
>>> b=1.2
>>> print(a,id(a))-----1.2 1938668708560
>>> print(b,id(b))-----1.2 1938668708144
>>> a is b-----False
>>> a is not b-----True
-----
>>> a=10
>>> b=10
>>> print(a,id(a))
10 1938667667984
>>> print(b,id(b))
10 1938667667984
>>> a is b
True
>>> a is not b
False
>>> a=256
>>> b=256
>>> print(a,id(a))
256 1938667675856
>>> print(b,id(b))
256 1938667675856
>>> a is b
True
>>> a is not b
False
>>> a=300
>>> b=300
>>> print(a,id(a))
300 1938668707664
>>> print(b,id(b))
300 1938668706064
>>> a is b
False
>>> a is not b
True
>>> a=257
>>> b=257
>>> print(a,id(a))
```

```
257 1938668711440
>>> print(b,id(b))
257 1938668707664
>>> a is b
False
>>> a is not b
True
>>> a=0
>>> b=0
>>> print(a,id(a))
0 1938667667664
>>> print(b,id(b))
0 1938667667664
>>> a is b
True
>>> a is not b
False
>>> a=-4
>>> b=-4
>>> print(a,id(a))
-4 1938667667536
>>> print(b,id(b))
-4 1938667667536
>>> a is b
True
>>> a is not b
False
>>> a=-1
>>> b=-1
>>> print(a,id(a))
-1 1938667667632
>>> print(b,id(b))
-1 1938667667632
>>> a is b
True
>>> a is not b
False
>>> a=-5
>>> b=-5
>>> print(a,id(a))
-5 1938667667504
>>> print(b,id(b))
-5 1938667667504
>>> a is b
True
>>> a is not b
False
>>> a=-6
>>> b=-6
>>> print(a,id(a))
```

```
-6 1938668707664
>>> print(b,id(b))
-6 1938668711440
>>> a is b
False
>>> a is not b
True
>>>
>>>
>>>
>>> a,b=300,300
>>> print(a,id(a))
300 1938668707696
>>> print(b,id(b))
300 1938668707696
>>> a is b
True
>>> a is not b
False
>>> a,b=-256,-256
>>> print(a,id(a))
-256 1938668706064
>>> print(b,id(b))
-256 1938668706064
>>> a is b
True
>>> a is not b
False
>>> l1,l2=[10,"KVR"],[10,"KVR"]
>>> print(l1,id(l1))
[10, 'KVR'] 1938669059648
>>> print(l2,id(l2))
[10, 'KVR'] 1938673238272
>>> l1 is l2
False
>>> l1 is not l2
True
```

===== Python Ternary Operator =====

=>The name of Python Ternary Operator is " if else Operator "

Syntax:- varname= Expr1 if Test Cond else Expr2

Explanation:

=>Here "if" and " else " are called Keywords

=>The Execution Process of if..else operator (Python Ternary Operator) is that" if the Test Cond result is True then PVM executes Expr1 and whose Result assigned to Varname. If the Resul of Test Cond is False PVM executes Expr2 and whose Result assigned to Varname".

#Program for finding big and small and equality of two numbers by using Ternary Operator

#bigsmallex1.py

```
a=float(input("Enter Value of a:")) # a=1
b=float(input("Enter Value of b:")) # b=20
big=a if a>b else b
small=a if a<b else b
print("big({},{})={}".format(a,b,big))
print("small({},{})={}".format(a,b,small))
```

#Program for finding big and small and equality of two numbers by using Ternary Operator

#bigsmallex2.py

```
a,b=float(input("Enter Value of a:")),float(input("Enter Value of b:"))
print("big({},{})={}".format(a,b,a if a>b else b))
print("small({},{})={}".format(a,b,a if a<b else b))
```

#Program for finding big and small and equality of two numbers by using Ternary Operator

#bigsmallex3.py

```
a,b=float(input("Enter Value of a:")),float(input("Enter Value of b:"))
big="BOTH VALUE ARE EQUAL" if a==b else a if a>b else b
small=a if (a<b) else b if b<a else "BOTH VALUE ARE EQUAL"
print("big({},{})={}".format(a,b,big))
print("small({},{})={}".format(a,b,small))
```

#Program for finding big and small and equality of Three numbers by using Ternary Operator

#bigsmallex4.py

```
a,b,c=float(input("Enter Value of a:")),float(input("Enter Value of b:")),float(input("Enter Value of c:"))
```

big="ALL VALUES ARE EQUAL" if (a==b) and (b==c) else a if (a>b) and (a>c) else b if (b>a) and (b>c) else c
sm=a if (a<b) and (a<c) else b if (b<a) and (b<c) else c if (c<a) and (c<b) else "ALL ARE EQUAL"

```
print("big({},{},{})={}".format(a,b,c,big))
print("small({},{},{})={}".format(a,b,c,sm))
```

#Program for finding big and small and equality of Three numbers by using Ternary Operator

#bigsmallex5.py

```
a,b,c=float(input("Enter Value of a:")),float(input("Enter Value of b:")),float(input("Enter Value of c:"))
```

big="ALL VALUES ARE EQUAL" if (a==b==c) else a if (b<a>c) else b if (ac) else c
sm=a if (b>a<c) else b if (a>b<c) else c if (a>c<b) else "ALL ARE EQUAL"

```
print("big({},{},{})={}".format(a,b,c,big))
print("small({},{},{})={}".format(a,b,c,sm))
```

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

Index

=>Purpose of Flow Control Statements in Python

=>Types of Control Statements in Python

- I) Conditional or Selection or Branching Statements
 - 1. Simple if statement
 - 2. if..else statement
 - 3. if..elif..else statement
 - 4. match case statement (Python 3.10 Version Onwards)

=>Programming Examples

- II) Looping or Iterating or Repeatative Statements
 - 1. while loop or while..else loop
 - 2. for loop or for..else loop

=>Programming Examples

- III) Transfer Flow Control Statements
 - 1. break
 - 2. continue
 - 3. pass

=>Programming Examples

=>Inner or Nested Loops

- a) while loop in while Loop
- b) for loop in for loop
- c) while loop in for loop
- d) for loop in while loop

=>Programming Examples

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

=>The Purpose of Flow Control Statements in Python is that "To Perform certain Operation

Either ONCE (True--X-Operation and False--Y-Operation) OR Repeatedly for finite number of Times Until Condition Becomes False".

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

- 1. Conditional or Selection or Branching Statements
- 2. Looping or Iterating or Repeatative Statements
- 3. Transfer Flow Control Statements

1. Conditional or Selection or Branching Statements

=>The purpose Conditional or Selection or Branching Statements is that " To perform Certain Operation Only Once depends on Condition Evaluation".

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

=>In Python programming, Conditional or Selection or Branching Statements are classified into 4 types. They are

1. Simple if statement
 2. if..else statement
 3. if..elif..else statement
 4. match case statement (Python 3.10 Version Onwards)
-

1. Simple if statement

Syntax:

```
if ( Test Cond ):  
    -->Statement-1  
    -->Statement-2  
    -->  
    -->Statement-n
```

Explanation:

=>Here 'if' is called Key Word
=>Here Test Condition is either True or False
=>If the Test Cond is True PVM Executes Indentation Block of Statements and Later Executes Other Statements in Program.
=>If the Test Cond is False PVM Executes Other Statements in Program without executing Indentation Block of Statements.

Flow Chart

```
graph TD; Entry((Entry)) --> TestCond{Test Cond ?}; TestCond -- True --> Indentation[Execute Indentation Block of Statements]; Indentation --> OtherStatements[Execute Other Statements in Program]; TestCond -- False --> OtherStatements;
```

```
#Moviee.py
tkt=input("D u have ticket(yes/no):")
if(tkt.lower()=="yes"):
    print("Enter into theater")
    print("Watch Moviee")
    print("Enjoy")
print("\nGoto Home and Read")
```

#WAP program which will accept any numerical number and decide the biggest among them two numbers

```
#bigex1.py
a=float(input("Enter value of a:")) # a=100
b=float(input("Enter value of b:")) # b=200
if( a>b ):
    print("max({},{})={}".format(a,b,a))
if(b>a):
    print("max({},{})={}".format(a,b,b))
print("\nProgram execution Completed!")
```

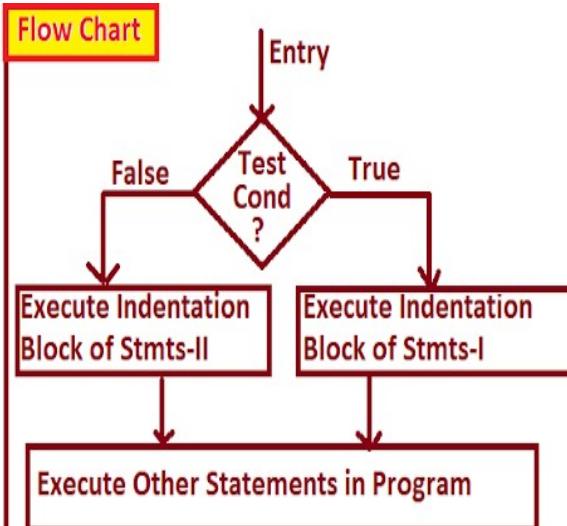
#WAP program which will accept any numerical number and decide the biggest among them two numbers

```
#bigex2.py
a=float(input("Enter value of a:")) # a=100
b=float(input("Enter value of b:")) # b=100
if( a>b ):
    print("max({},{})={}".format(a,b,a))
if(b>a):
    print("max({},{})={}".format(a,b,b))
if(a==b):
    print("Both values are Equal:")
print("\nProgram execution Completed!")
```

2. if..else statement

Syntax:

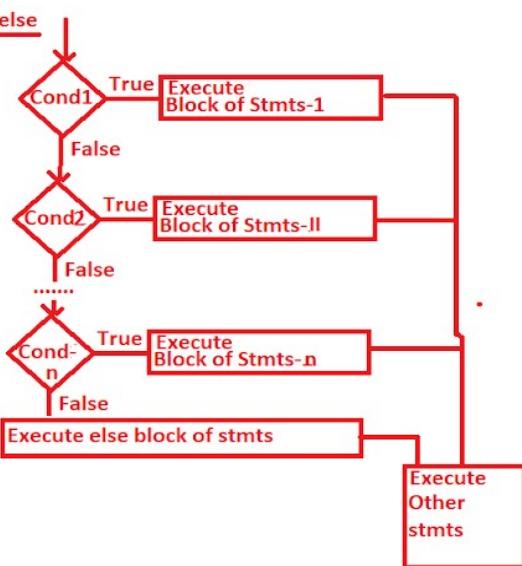
```
if (Test Cond) :  
    Statement-1 Indentation  
    Statement-2 Block-I  
    Statement-n  
else:  
    Statement-11 Indentation  
    Statement-12 Block-II  
    Statement-1n  
Other Statements  
in Program
```



Explanation:

- =>Here if and else are the keywords
- =>If the Test Cond is True then PVM Executes Indentation Block of Statements-I and later executes Other Statements in program.
- =>If the Test Cond is False then PVM Executes Indentation Block of Statements-II and later executes Other Statements in program.

flow chart for if..elif..else



if..elif..else statement:

Syntax:-

```

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

```

Explanation:

=>if the Test Cond-1 is True then PVM executes Block of stmts-1 and other stmts.
=>if the Test Cond-1 is False and if Test cond-2 is True then PVM executes Block of stmts-II and other stmts.
=>This Process will be reapeated until all test conditions evaluated and all the test conditions are false PVM executes else block of stmts and other stmts in Program.
=>Writing else block is Optional.

```

#Program for accepting 3 values and find biggest among them by using if..else statement
#BigThreeEx1.py
a=float(input("Enter Value of a:")) # a=10
b=float(input("Enter Value of b:"))# b=20
c=float(input("Enter Value of c:")) # c=20
if(a>b) and (a>c):
    print("big({},{},{}]={}.".format(a,b,c,a))
else:
    if(b>a) and (b>=c):
        print("big({},{},{}]={}.".format(a,b,c,b))
    else:
        if(c>a) and (c>b):
            print("big({},{},{}]={}.".format(a,b,c,c))
        else:
            print("ALL VALUES ARE EQUAL")

```

```

#Program for accepting 3 values and find biggest among them by using if..else statement
#BigThreeEx3.py
a=float(input("Enter Value of a:")) # a=20
b=float(input("Enter Value of b:"))# b=20
c=float(input("Enter Value of c:")) # c=20
if(a>b) and (a>c):
    print("big({},{},{}]={}.".format(a,b,c,a))
elif(b>a) and (b>=c):
    print("big({},{},{}]={}.".format(a,b,c,b))
elif(c>a) and (c>b):
    print("big({},{},{}]={}.".format(a,b,c,c))
elif(a==b==c):

```

```

        print("ALL VALUES ARE EQUAL")
print("\nProgram is completed")


---


#Program accepting a digit from key board and display its name
#digitex1.py
d=int(input("Enter a digit:")) # d= 0 1 2 3 4 5 6 7 8 9 99
if(d==0):
    print("{} is ZERO".format(d))
else:
    if(d==1):
        print("{} is ONE".format(d))
    else:
        if(d==2):
            print("{} is TWO".format(d))
        else:
            if(d==3):
                print("{} is THREE".format(d))
            else:
                if(d==4):
                    print("{} is FOUR".format(d))
                else:
                    if(d==6):
                        print("{} is SIX".format(d))
                    else:
                        if(d==5):
                            print("{} is FIVE".format(d))
                        else:
                            if(d==8):
                                print("{} is")
                                print("{} is EIGHT".format(d))
                            else:
                                if(d==7):
                                    print("{} is")
                                    print("{} is SEVEN".format(d))
                                else:
                                    if(d==9):
                                        print("{} is")
                                        print("{} is NINE".format(d))
                                    else:
                                        print("{} is a number:".format(d))


---



```

```

#Program accepting a digit from key board and display its name
#digitex2.py
d=int(input("Enter a digit:")) # d= 0 1 2 3 4 5 6 7 8 9 99
if(d==0):
    print("{} is ZERO".format(d))
elif(d==1):

```

```
        print("{} is ONE".format(d))
elif(d==2):
    print("{} is TWO".format(d))
elif(d==3):
    print("{} is THREE".format(d))
elif(d==4):
    print("{} is FOUR".format(d))
elif(d==5):
    print("{} is FIVE".format(d))
elif(d==7):
    print("{} is SEVEN".format(d))
elif(d==6):
    print("{} is SIX".format(d))
elif(d==8):
    print("{} is EIGHT".format(d))
elif(d==9):
    print("{} is NINE".format(d))
else:
    print("{} is NUMBER".format(d))
```

```
#Program accepting a digit from key board and display its name
#digitex3.py
d=int(input("Enter a digit:")) # d= 0  1  2  3  4  5  6  7  8  9  99
if(d==0):
    print("{} is ZERO".format(d))
if(d==1):
    print("{} is ONE".format(d))
if(d==2):
    print("{} is TWO".format(d))
if(d==3):
    print("{} is THREE".format(d))
if(d==4):
    print("{} is FOUR".format(d))
if(d==5):
    print("{} is FIVE".format(d))
if(d==7):
    print("{} is SEVEN".format(d))
if(d==6):
    print("{} is SIX".format(d))
if(d==8):
    print("{} is EIGHT".format(d))
if(d==9):
    print("{} is NINE".format(d))
if(d not in [0,1,2,3,4,5,6,7,8,9] ):
    print("{} is NUMBER".format(d))
```

```
#Program accepting a digit from key board and display its name
#digitex4.py
```

```
d={0:"ZERO",1:"ONE",2:"TWO",3:"THREE",4:"FOUR",5:"FIVE",6:"SIX",7:"SEVEN",8:"EIGHT",9:"NINE"}
dig=int(input("Enter a digit:"))
res=d.get(dig)
if(res!=None):
    print("{} is {}".format(dig,res))
else:
    print("{} is Number:".format(dig))
```

```
#Program accepting a digit from key board and display its name
#digitex4.py
d={0:"ZERO",1:"ONE",2:"TWO",3:"THREE",4:"FOUR",5:"FIVE",6:"SIX",7:"SEVEN",8:"EIGHT",9:"NINE"}
dig=int(input("Enter a digit:"))
res= d.get(dig)  if (d.get(dig)!=None) else "Not a Number"
print("{} is {}".format(dig,res))
print("=====OR=====")
print("{} is {}".format(dig,d.get(dig))  if (d.get(dig)!=None) else "Not a Number"))

#Program for acl area and perimeter of circle
#Circle.py
r=float(input("Enter Radius:"))
if(r>=0):
    ar=3.14*r**2
    pr=2*3.14*r
    print("Area of circle={}".format(ar))
    print("Perimeter of circle={}".format(pr))
else:
    print("{} is Invalid input:".format(r))
```

===== match .. case statement (Python 3.10) =====

=>It is one of new Feature in Python 3.10
=>This features is also known as Multi way decision making statement.
=>It is always recommended to handling Pre-designed Conditions.

=>Syntax:

```
match (Choice Expression):
    case label1:
        block of statements-I
    case label2:
        block of statements-II
```

```
case label-n:  
    block of statements-n  
case _ : # default case block  
    default block of statements
```

```
Other statements in Program
```

```
Explanation:
```

- 1) here 'match' and 'case' are the keywords
 - 2) here 'Choice expression' can be any data type value except float.
 - 3) here the value of choice expression is comparing with case label1 . If it is True then execute Block of statements-I and also execute other statements in Program. Choice expression is not matching with case label1 then it compares with case label2 and if it matches then execute Block of statements-II and also execute other statements in Program and so on.
 - 4) In general if the value of choice expression is matching with any case label then PVM executes corresponding block of statements and also executes Other statements in the program.
 - 5) if value of choice expression is not matching with any case labels then PVM executes block of statements written under default case block and also executes Other statements in the program.
 - 6) Writing default case label is optional.
 - 7) If we write default case label then we must write it at last otherwise we get SyntaxError.
-

```
#WAPP which will implement the following mean driven application
```

```
import sys  
#matchcaseex1.py  
print("=*50)  
print("tArithematic Operations")  
print("=*50)  
print("\t1.Addition:")  
print("\t2.Substration:")  
print("\t3.Multiplication:")  
print("\t4.Division:")  
print("\t5.Modulo Div:")  
print("\t6.Exponentiation:")  
print("\t7.Exit:")  
print("=*50)  
ch=int(input("Enter Ur Choice:"))  
match(ch):  
    case 1:  
        a=float(input("Enter First Value for Addition:"))  
        b=float(input("Enter Second Value for Addition:"))  
        print("\tsum({},{})={}".format(a,b,a+b))
```

```

case 2:
    a=float(input("Enter First Value for Subtraction:"))
    b=float(input("Enter Second Value for Subtraction:"))
    print("\tsub({},{})={}".format(a,b,a-b))
case 3:
    a=float(input("Enter First Value for Multiplication:"))
    b=float(input("Enter Second Value for Multiplication:"))
    print("\tMul({},{})={}".format(a,b,a*b))
case 4:
    a=float(input("Enter First Value for Division:"))
    b=float(input("Enter Second Value for Division:"))
    print("\tDivision({},{})={}".format(a,b,a/b))
    print("\tFloor Division({},{})={}".format(a,b,a//b))
case 5:
    a=float(input("Enter First Value for Modulo Div:"))
    b=float(input("Enter Second Value for Modulo Div:"))
    print("\tModulo Div({},{})={}".format(a,b,a%b))
case 6:
    a=float(input("Enter Base:"))
    b=float(input("Enter Power:"))
    print("\tpow({},{})={}".format(a,b,a**b))
case 7:
    print("Thx for using Program")
    sys.exit()
case _: # Default Case Block
    print("{} is invalid Choice, try again:".format(ch))

```

```

#matchcaseex2.py
wkd=input("Enter a week name:")
match(wkd.lower()):
    case "monday":
        print("{} is working day".format(wkd))
    case "tuesday":
        print("{} is working day".format(wkd))
    case "wednessday":
        print("{} is working day".format(wkd))
    case "thursday":
        print("{} is working day".format(wkd))
    case "friday":
        print("{} is working day".format(wkd))
    case "saturday":
        print("{} is week end--underground action plans".format(wkd))
    case "sunday":
        print("{} is holiday and implementing UG Plans".format(wkd))

```

```

#matchcaseex3.py
wkd=input("Enter a week name:")
match(wkd.lower()):

```

```

case "monday"|"tuesday" | "wednesday"|"thursday"|"friday":
    print("{} is working day".format(wkd))
case "saturday":
    print("{} is week end--underground action plans".format(wkd))
case "sunday":
    print("{} is holiday and implementing UG Plans".format(wkd))
case _:
    print("{} is not a week day".format(wkd))

```

```

#matchcaseex4.py
wkd=input("Enter a week name:")
match(wkd.lower())[0:3]:
    case "mon"|"tue" | "wed"|"thu"|"fri":
        print("{} is working day".format(wkd))
    case "sat":
        print("{} is week end--underground action plans".format(wkd))
    case "sun":
        print("{} is holiday and implementing UG Plans".format(wkd))
    case _:
        print("{} is not a week day".format(wkd))

```

```

import sys
#nonmatchcaseex1.py
print("=*50)
print("\tArithematic Operations")
print("=*50)
print("\t1.Addition:")
print("\t2.Substration:")
print("\t3.Multiplication:")
print("\t4.Division:")
print("\t5.Modulo Div:")
print("\t6.Exponentiation:")
print("\t7.Exit:")
print("=*50)
ch=int(input("Enter Ur Choice:"))
if(ch==1):
    a=float(input("Enter First Value for Addition:"))
    b=float(input("Enter Second Value for Addition:"))
    print("\tsum({},{})={}".format(a,b,a+b))
elif(ch==2):
    a=float(input("Enter First Value for Substration:"))
    b=float(input("Enter Second Value for Substration:"))
    print("\tsub({},{})={}".format(a,b,a-b))
elif(ch==3):
    a=float(input("Enter First Value for Multiplication:"))
    b=float(input("Enter Second Value for Multiplication:"))
    print("\tMul({},{})={}".format(a,b,a*b))
elif(ch==4):

```

```

a=float(input("Enter First Value for Division:"))
b=float(input("Enter Second Value for Division:"))
print("\tDivision({},{})={}".format(a,b,a/b))
print("\tFloor Division({},{})={}".format(a,b,a//b))
elif(ch==5):
    a=float(input("Enter First Value for Modulo Div:"))
    b=float(input("Enter Second Value for Modulo Div:"))
    print("\tModulo Div({},{})={}".format(a,b,a%b))
elif(ch==6):
    a=float(input("Enter Base:"))
    b=float(input("Enter Power:"))
    print("\tpow({},{})={}".format(a,b,a**b))
elif(ch==7):
    print("Thx for using Program")
    exit()
else: # Default Case Block
    print("{} is invalid Choice, try again:".format(ch))

```

===== Looping or Iterating or Repeataative Statements =====

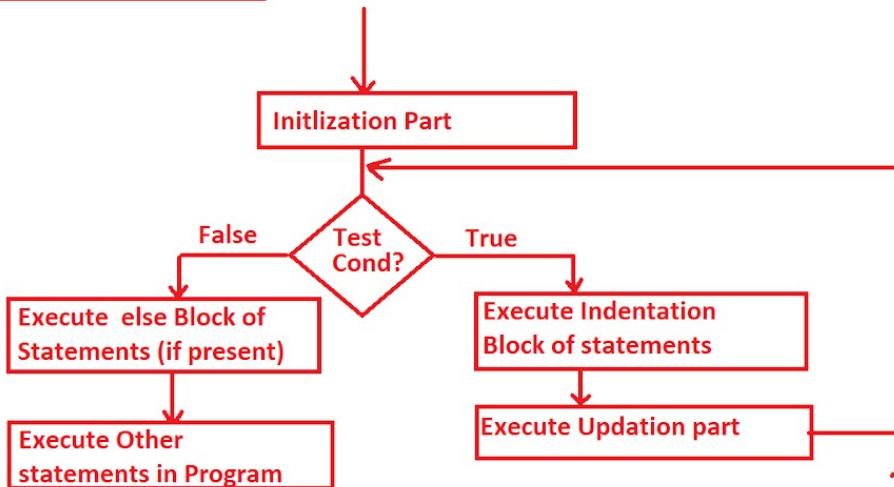
=>The purpose of Looping statements is that "To perform Certain Operation Repeatedly for finite number of times until Test Cond Becomes False."
=>In Python Programming, we have 2 types of Looping statements. They are

1. while loop (OR) while ... else loop
2. for loop (OR) for.... else loop

=>At the time of dealing with looping statements, Programmer must ensure there must 3 parts. They are

1. Initlization Part (From Where to Start)
 2. Conditiional Part (Upto How Many times to repeat)
 3. Updation Part (Incrmentation or decrementation)
-

Flow Chart for while loop



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

Syntax:

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

OR

while (Test Cond) :
--- Block of statements
--- else:

--- Other statements in Program

Explanation:-

=>Test condition result may be True or False

=>In the while loop, if the test condition is true then PVM executes Indentation block of statements and once again PVM control goes to Test Cond. If the Test Cond is once again True then PVM executes Indentation block of statements once again. This Process will be continued until Test Cond becomes False.

=>Once The test cond becomes False then PVM execute else block of statememnts, which are written in else block and later also executes other statements in program.

=====

2. for loop or for ...else loop

=====

Syntax1:-

for varname in Iterable_object:

Indentation block of stmts

Other statements in Program

Syntax2:

for varname in Iterable_object:

Indentation block of stmts

else:

else block of statements

Other statements in Program

Explanation:

=>Here 'for' and 'else' are keywords

=>Here Iterable_object can be Sequence(bytes,bytearray,range,str),
list(list,tuple),set(set,frozenset) and dict.

=>The execution process of for loop is that " Each of Element of Iterable_object selected , placed
in varname and executes Indentation block of statements".This Process will be repeated until all
elements of Iterable_object completed.

=>After execution of Indentation block of statements, PVM executes else block of statements
which are written under else block and later PVM executes Other statements in Program.

=>Writing else block is optional.

#WAP program which will generate 1 to n numbers where n must be positive value

#NumGenEx1.py

n=int(input("Enter How Many Values u want to generate:"))

if(n<=0):

 print("{} is invalid input".format(n))

else:

 print("*50)

```
print("Numbers within :{}".format(n))
print("*50)
i=1 # Initialization Part
while(i<=n): # Cond Part
    print("\t{}".format(i))
    i=i+1 # Updation Part
```

```
#WAP program which will generate 1 to n numbers where n must be positive value
#NumGenEx2.py
n=int(input("Enter How Many Values u want to generate:"))
if(n<=0):
    print("{} is invalid input".format(n))
else:
    print("*50)
    print("Numbers within :{}".format(n))
    print("*50)
    i=1 # Initialization Part
    while(i<=n): # Cond Part
        print("\t{}".format(i))
        i=i+1 # Updation Part
    else:
        print("*"*50)
```

```
#WAP program which will generate even number within n where n must be the positive int
value
#EvenNumbers.py
n=int(input("Enter How Many Range of Even Numbers u want to genertate:"))
if(n<=0):
    print("{} is invalid input:".format(n))
else:
    print("*50)
    print("List of Even Numbers within: {}".format(n))
    print("*50)
    i=2 # Initialization Part
    while(i<=n): # Cond Part
        print("\t{}".format(i))
        i=i+2 #Updation Part
    else:
        print("*50)
```

```
#WAP program which will generate n to 1 where n is the positive int value
#MulTableEx.py
n=int(input("Enter a number for generating mul table:"))
if(n<=0):
    print("{} is invalid input".format(n))
else:
    print("*50)
    print("Mul Table for {}".format(n))
```

```
print("*50)
i=1
while(i<=10):
    print("\t{} x {} = {}".format(n,i,n*i))
    i=i+1
else:
    print("*50)
```

```
#WAP program which will generate n to 1 where n is the positive int value
#
n=int(input("Enter a number for generating mul table:"))
if(n<=0):
    print("{} is invalid input".format(n))
else:
    print("*50)
    print("Mul Table for {}".format(n))
    print("*50)
    for i in range(1,11):
        print("\t{} x {} = {}".format(n,i,n*i))
    else:
        print("#*50)
```

#whileforloop.py

```
s="PYTHON" # Iterable-object
print("====By using While Loop====")
i=len(s)-1
while(i>=0):
    print("\t{}".format(s[i]))
    i=i-1
print("-----")
print("\n====By using For Loop====")
for ch in s[:-1]:
    print("\t{}".format(ch))
```

#Wap program which will accept the line of text and find and display number of vowels

```
#VowelsCount.py
line=input("Enter Line of Text:") # Python is an oop lang
cnt=0
for ch in line:
    if ch in ['a','e','i','o','u','A','E','I','O','U']:
        print("\t{}".format(ch))
        cnt=cnt+1
else:
    print("Number of Vowels={}".format(cnt))
```

#Wap program which will accept the line of text and find and display number of vowels

```
#VowelsCount1.py
line=input("Enter Line of Text:") # Python is an oop lang
```

```

cnt=0
for ch in line:
    if ch.lower() in ['a','e','i','o','u']:
        print("\t{}{}".format(ch))
        cnt=cnt+1
    else:
        print("Number of Vowels={}".format(cnt))


---


#Wap program which will Test Wether the given number is perfect or not
#perfect.py
n=int(input("Enter a Number:"))
if(n<=0):
    print("{} is invalid input".format(n))
else:
    print("-"*40)
    print("Factors of {}".format(n))
    print("-"*40)
    fs=0
    for i in range(1,(n//2)+1):
        if(n%i==0):
            print("\t{}".format(i))
            fs=fs+i
    else:
        print("-"*50)
        print("Factors Sum=",fs)
        if(n==fs):
            print("{} is Perfect:".format(n))
        else:
            print("{} is Not Perfect:".format(n))


---


#WAP program which will find some of first n natural numbers.some of square naturals number
,some of cubes of natural numbers..
#NatSumsSumEx1.py
n=int(input("Enter How Many Natural Numbers sum u want to Find:"))
if(n<=0):
    print("{} is invalid input".format(n))
else:
    print("*50")
    print("\tNatSums\tSquare Numbs\tCubes Numbs")
    s,ss,cs=0,0,0
    print("*50")
    for i in range(1,n+1):
        print("\t{}\t{}\t{}".format(i,i**2,i**3))
        s=s+i
        ss=ss+i**2
        cs=cs+i**3
    else:
        print("*50")

```

```
    print("\t{}\t{}\t{}".format(s,ss,cs))
    print("=*50)
```

```
#Program for cal factorial of a Number--- 1 x 2 x 3 x 4.....n
```

```
#FactorialEx1.py
```

```
n=int(input("Enter a number:"))
```

```
if(n<0):
```

```
    print("{} is invalid input".format(n))
```

```
else:
```

```
    f=1
```

```
    for i in range(1,n+1):
```

```
        f=f*i
```

```
    else:
```

```
        print("Factorial({})={}".format(n,f))
```

```
#Program for cal factorial of a Number--- n x n-1 x n-2----- 0!
```

```
#FactorialEx2.py
```

```
n=int(input("Enter a number:"))
```

```
if(n<0):
```

```
    print("{} is invalid input".format(n))
```

```
else:
```

```
    f=1
```

```
    for i in range(n,0,-1):
```

```
        f=f*i
```

```
    else:
```

```
        print("{}!={}".format(n,f))
```

```
#WAP program which will accept any numerical int value and find its factors..
```

```
# HINT: n=6----Factors----- 1 2 3
```

```
#FactorsEx1.py
```

```
n=int(input("Enter For Which Number u want to Find factors:"))
```

```
print("=*50)
```

```
print("Factors of {}".format(n))
```

```
for i in range(1,(n//2)+1):
```

```
    if(n%i==0):
```

```
        print("\t{}".format(i))
```

```
print("=*50)
```

----- break statement -----

=>break is a key word

=>The purpose of break statement is that "To terminate the execution of loop logically when certain condition is satisfied and PVM control comes out of corresponding loop and executes other statements in the program".

=>when break statement takes place inside for loop or while loop then PVM will not execute corresponding else block(bcoz loop is not becoming False) but it executes other statements in the program

=>Syntax1:

```
for varname in Iterable_object:
```

```
-----  
if (test cond):  
    break  
-----
```

=>Syntax2:

```
while(Test Cond-1):
```

```
-----  
if (test cond-2):  
    break  
-----
```

=====X=====

continue statement

=>continue is a keyword

=>continue statement is used for making the PVM to go to the top of the loop without executing the following statements which are written after continue statement for that current Iteration only.

=>continue statement to be used always inside of loops.

=>when we use continue statement inside of loop then else part of corresponding loop also executes provided loop condition becomes false.

=>Syntax:-

```
for varname in Iterable-object:
```

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

=>Syntax:-

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

=====X=====

```
#breakex1.py  
s="PYTHON"  
for ch in s:  
    print(ch)  
print("-----")  
for ch in s:  
    if(ch=="H"):  
        break  
    else:  
        print(ch)  
else:  
    print("else part of for loop")  
print("\tOther part of the program")
```

```
#breakex2.py  
s="PYTHON"  
for ch in s:  
    print(ch)  
print("-----")  
i=0  
while(i<len(s)):  
    if(s[i]=="H"):  
        break  
    else:  
        print("\t{} ".format(s[i]))  
        i=i+1  
else:  
    print("else part of while loop")  
print("\tOther part of the program")
```

```
#breakex3.py
lst=[10,"Rossum",34.56,True,False,2+3j]
for val in lst:
    if(val==True):
        break
    else:
        print("\t{}{}".format(val))
else:
    print("Else part of for loop")
print("Other part of program")
```

```
#Program for deciding whetehr the given number is prime or Not
#Prime Number = Dividing by 1 and itself.---- 2  3  5  7  11 13 17....
#PrimeEx1.py
n=int(input("Enter a Number:")) #n=9
if(n<2):
```

```
    print("{} is invalid input".format(n))
else:
    dec="PRIME"
    for i in range(2,n):
        if(n%i==0):
            dec="NOTPRIME"
            break

    if(dec=="PRIME"):
        print("{} is Prime Number".format(n))
    else:
        print("{} is Not Prime Number".format(n))
```

```
#Program for deciding whetehr the given number is prime or Not
#Prime Number = Dividing by 1 and itself.---- 2  3  5  7  11 13 17....
#PrimeEx2.py
n=int(input("Enter a Number:")) #n=9
if(n<2):
```

```
    print("{} is invalid input".format(n))
else:
    prmno=True
    for i in range(2,n):
        if(n%i==0):
            prmno=False
            break

    if(prmno==True):
        print("{} is Prime Number".format(n))
    else:
        print("{} is Not Prime Number".format(n))
```

```
#continueex1.py
s="PYTHON"
for ch in s:
```

```
    print("\t{}".format(ch))
print("-----")
for ch in s:
    if(ch=="H"):
        continue
    else:
        print("\t{}".format(ch))
else:
    print("else part of for loop")
print("Other part of program")
```

```
#continueex2.py
s="PYTHON"
for ch in s:
    if(ch=="Y" or (ch=="O")):
        continue
    else:
        print("\t{}".format(ch))
else:
    print("else part of for loop")
print("Other part of program")
```

```
#continueex3.py
lst=[10,20,-34,56,-12,0,45,-56,5]
print("Positive Elements:")
for val in lst:
    if(val<=0):
        continue
    print("\t{}".format(val))
```

```
#continueex4.py
lst=[10,20,-34,56,-12,0,45,-56,5]
print("Negative Elements:")
for val in lst:
    if(val>=0):
        continue
    print("\t{}".format(val))
```

```
#continueex5.py
lst=[10,20,-34,56,-12,0,45,-56,5]
ps=0
print("List of Positive Values:")
for val in lst:
    if(val<=0):
        continue
    print("\t{}".format(val))
    ps=ps+val
else:
    print("Sum of Positive Elements={}".format(ps))
    print("-----")
```

```

ns=0
for val in lst:
    if(val>=0):
        continue
    print("\t{}".format(val))
    ns=ns+val
else:
    print("Sum of Negative Elements={}".format(ns))
    print("-----")

```

```

#continueex6.py
lst=[11,-24,12,56,-23,57,0,27,-45,71]
print("List of Even Numbers")
for val in lst:
    if(val%2!=0) or (val<0):
        continue
    print("\t{}".format(val))
else:
    print("List of Odd Numbers")
    for val in lst:
        if(val%2==0) or (val<0):
            continue
        print("\t{}".format(val))

```

=====

Inner or Nested Loops

=====

=>The Process of Defining One Loop in another Loop is called Inner or Nested Loop
=>The Execution Process of Inner or Nested Loop is that "For Every Value of Outer Loop , inner Loop
process repeats Multiple Finite number of times until Test Cond becomes False".
=>We can define Inner or Nested Loops in Four Ways. They are

Syntax-1: for loop in for loop

```

for varname1 in Iterable_object:
    -----
    -----
    for varname2 in Iterable_object:
        -----
        -----
        else:
            -----
else:
    -----

```

Syntax-2: while loop in while loop

```
while(Test Cond1):
```

```
    while (Test Cond2):
```

```
        else:
```

```
    else:
```

Syntax-3: while loop in for loop

```
for varname in Iterable_object:
```

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

```
        else:
```

```
    else:
```

Syntax-4: for loop in while loop

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

```
    for varname in Iterable_object:
```

```
        else:
```

```
    else:
```

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

#Program bt using Inner for loops ---for in for loop
#InnerLoopEx1.py
for i in range(1,5):
    print("Val of i-outer for loop:{}".format(i))
    print("-"*50)
    for j in range(1,4):
        print("\tVal of j-inner for loop:{}".format(j))
    else:
        print("I am out of inner-for loop")
        print("-"*50)
else:
    print("I am out of outer-for loop")

=====
```

```
E:\KVR-PYTHON-11AM\LOOPS>py InnerLoopEx1.py
Val of i-outer for loop:1
```

```
-----  
    Val of j-inner for loop:1  
    Val of j-inner for loop:2  
    Val of j-inner for loop:3
```

```
I am out of inner-for loop
```

```
-----  
Val of i-outer for loop:2
```

```
-----  
    Val of j-inner for loop:1  
    Val of j-inner for loop:2  
    Val of j-inner for loop:3
```

```
I am out of inner-for loop
```

```
-----  
Val of i-outer for loop:3
```

```
-----  
    Val of j-inner for loop:1  
    Val of j-inner for loop:2  
    Val of j-inner for loop:3
```

```
I am out of inner-for loop
```

```
-----  
Val of i-outer for loop:4
```

```
-----  
    Val of j-inner for loop:1
```

```
Val of j-inner for loop:2
Val of j-inner for loop:3
I am out of inner-for loop
```

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

```
#Program by using Inner for loops ---while in while loop
#InnerLoopEx2.py
i=1
while(i<=4):
    print("Val of i-outer while loop:{}".format(i))
    print("-"*50)
    j=1
    while(j<=3):
        print("\tVal of j-inner for loop:{}".format(j))
        j=j+1
    else:
        i=i+1
        print("I am out of inner-while loop")
        print("-"*50)
else:
    print("I am out of outer-while loop")
```

```
"""""
E:\KVR-PYTHON-11AM\LOOPS>py InnerLoopEx2.py
Val of i-outer while loop:1
```

```
Val of j-inner for loop:1
Val of j-inner for loop:2
Val of j-inner for loop:3
I am out of inner-while loop
```

```
Val of i-outer while loop:2
```

```
Val of j-inner for loop:1
Val of j-inner for loop:2
Val of j-inner for loop:3
I am out of inner-while loop
```

```
Val of i-outer while loop:3
```

```
Val of j-inner for loop:1
Val of j-inner for loop:2
Val of j-inner for loop:3
I am out of inner-while loop
```

Val of i-outer while loop:4

Val of j-inner for loop:1

Val of j-inner for loop:2

Val of j-inner for loop:3

I am out of inner-while loop

I am out of outer-while loop

#Program by using Inner for loops ---while in for loop

#InnerLoopEx3.py

for i in range(4,0,-1):

 print("Val of i--Outer For loop:{} ".format(i))

 print("-"*50)

 j=3

 while(j>=1):

 print("\tValue of j={} ".format(j))

 j=j-1

 else:

 print("I am out of inner while loop")

 print("-"*50)

else:

 print("I am out of outer for loop")

E:\KVR-PYTHON-11AM\LOOPS>py InnerLoopEx3.py

Val of i--Outer For loop:4

Value of j=3

Value of j=2

Value of j=1

I am out of inner while loop

Val of i--Outer For loop:3

Value of j=3

Value of j=2

Value of j=1

I am out of inner while loop

Val of i--Outer For loop:2

Value of j=3

Value of j=2

Value of j=1

I am out of inner while loop

Val of i--Outer For loop:1

Value of j=3

Value of j=2

Value of j=1

I am out of inner while loop

I am out of outer for loop """

#Program by using Inner for loops ---while in for loop

#InnerLoopEx4.py

i=1

while(i<=4):

 print("Val of i--Outer while loop:{}".format(i))

 print("-"*50)

 for j in range(3,0,-1):

 print("\tVal of j-inner for loop:{}".format(j))

 else:

 print("I am out of inner-for loop")

 print("-"*50)

 i=i+1

else:

 print("I am out of outer WHILE loop")

"""

E:\KVR-PYTHON-11AM\LOOPS>py InnerLoopEx4.py

Val of i--Outer while loop:1

 Val of j-inner for loop:3

 Val of j-inner for loop:2

 Val of j-inner for loop:1

I am out of inner-for loop

Val of i--Outer while loop:2

 Val of j-inner for loop:3

 Val of j-inner for loop:2

 Val of j-inner for loop:1

I am out of inner-for loop

Val of i--Outer while loop:3

 Val of j-inner for loop:3

 Val of j-inner for loop:2

 Val of j-inner for loop:1

I am out of inner-for loop

Val of i--Outer while loop:4

Val of j-inner for loop:3

Val of j-inner for loop:2

Val of j-inner for loop:1

I am out of inner-for loop

I am out of outer-while loop

"""

#MulTablesUsingInnerLoops.py

n=int(input("Enter How Many Mul Tables u Want:"))

if(n<=0):

 print("{} is invalid input".format(n))

else:

 for i in range(1,n+1): # Outer loop supply the Number

 print("-"*50)

 print("Mul Table for :{}".format(i))

 print("-"*50)

 for j in range(1,11): # Inner Loop--generates Mul table

 print("\t {} x {} = {}".format(i,j,i*j))

 else:

 print("-"*50)

#RandomMulTablesUsingInnerLoops.py

n=int(input("Enter How Many Mul Tables u Want:"))

if(n<=0):

 print("{} is invalid input".format(n))

else: # 5 19 3 -3 6

 lst=[] # create an empty list for adding random numbers

 for i in range(1,n+1):

 val=int(input("Enter {} Value:".format(i)))

 lst.append(val)

 else:

 print("-"*50)

 print("Given Values:{}".format(lst)) # [5, 9, -4, 19, 2]

 print("-"*50)

 for num in lst:

 if(num<=0):pass

 else:

 print("Mul Table for :{}".format(num))

 print("-"*50)

 for j in range(1,11): # Inner Loop--generates Mul table

 print("\t {} x {} = {}".format(num,j,num*j))

 else:

```
    print("-"*50)


---


#WAPP Which will generate list of prime number within the given range
#RangePrimes.py
n=int(input("Enter How Many Primes u want within the range:"))
if(n<=1):
    print("{} is invalid input".format(n))
else:
    for num in range(2,n+1):
        res="PRIME"
        for i in range(2,num):
            if(num%i==0):
                res="NOTPRIME"
                break

        if(res=="PRIME"):
            print("\t{}".format(num))


---


```

```
#RandomPrimes.py
n=int(input("Enter How Many Numbers u have to decide whether they are prime or not:"))
if(n<=1):
    print("{} is invalid input".format(n))
else:
    lst=list() # create an empty list for adding random numbers
    for i in range(1,n+1):
        val=int(input("Enter {} Value:".format(i)))
        lst.append(val)
    else:
        print("-"*50)
        print("Given Values: {}".format(lst)) # [12, 11, -5, 7, 1, 3]
        print("-"*50)
        #get primes
        for num in lst:
            if(num<=1):pass
            else:
                res="PRIME"
                for i in range(2,num):
                    if(num%i==0):
                        res="NOTPRIME"
                        break
                if(res=="PRIME"):
                    print("\t{}".format(num))


---


```

String Handling Part-2

=>On String Data, we can perform Indexing, Slicing Operations and with these operations, we can also perform different type of operations by using pre-defined functions present in str object.

Pre-defined Functions in str object

1) capitalize()

=>This Function is used for capitalizing the first letter of First word of a given Sentence only.

=>Syntax: strobj.capitalize()

(OR)

strobj=strobj.capitalize()

Examples:

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

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

2) title():

=>This is used for obtaining Title Case of a Given Sentence (OR) Making all words First Letters are capital.

Syntax: s.title()

(OR)

s=s.title()

Examples:

```
>>> s="python"
>>> print(s,type(s))-----python <class 'str'>
>>> s.capitalize()-----'Python'
>>> s.title()-----'Python'
```

```
-----  
>>> s="python is an oop lang"  
>>> print(s,type(s))-----python is an oop lang <class 'str'>  
>>> s.capitalize()-----'Python is an oop lang'  
>>> s.title()-----'Python Is An Oop Lang'  
>>> print(s)-----python is an oop lang  
>>> s=s.title()  
>>> print(s)-----Python Is An Oop Lang  
-----
```

3) index()

```
-----  
=>This Function obtains Index of the specified Value  
=>If the specified value does not exist then we get ValueError  
=>Syntax: strobj.index(Value)  
=>Syntax: indexvalue=strobj.index(value)
```

Examples:

```
-----  
>>> s="python"  
>>> s.index("p")-----0  
>>> s.index("y")-----1  
>>> s.index("o")-----4  
>>> s.index("n")-----5  
>>> s.index("K")-----ValueError: substring not found
```

=>enumerate() is one the general function, which is used for finding Index and Value of an Iterable object.

NOTE:

```
-----  
>>> for i,v in enumerate(s):  
...     print("Index:{} and Value:{} ".format(i,v))
```

OUTPUT

```
-----  
Index:0 and Value:p  
Index:1 and Value:y  
Index:2 and Value:t  
Index:3 and Value:h  
Index:4 and Value:o  
Index:5 and Value:n
```

```
-----  
>>> lst=[10,"Rossum",23.45,True]  
>>> for i,v in enumerate(lst):  
...     print("Index:{} and Value:{} ".format(i,v))
```

OUTPUT

Index:0 and Value:10
Index:1 and Value:Rossum
Index:2 and Value:23.45
Index:3 and Value:True

4) upper()

=>It is used for converting any type of Str Data into Upper Case.

=>Syntax:- strobj.upper()

OR
strobj=strobj.upper()

Examples:

```
>>> s="python"
>>> print(s)-----python
>>> s.upper()-----'PYTHON'
>>> s="python is an oop lang"
>>> print(s)-----python is an oop lang
>>> s.upper()-----'PYTHON IS AN OOP LANG'
>>> s="Python IS an OOP lang"
>>> print(s)-----Python IS an OOP lang
>>> s.upper()-----'PYTHON IS AN OOP LANG'
>>> s="AbCdEf"
>>> print(s)-----AbCdEf
>>> s.upper()-----'ABCDEF'
>>> s="PYTHON"
>>> print(s)-----PYTHON
>>> s.upper()-----'PYTHON'
>>> s="123"
>>> print(s)-----123
>>> s.upper()-----'123'
```

5) lower()

=>It is used for converting any type of Str Data into lower Case.

=>Syntax:- strobj.lower()

OR
strobj=strobj.lower()

Examples:

```
>>> s="Data Science"
>>> print(s)-----Data Science
>>> s.lower()-----'data science'
>>> s="python"
>>> print(s)-----python
>>> s.lower()-----'python'
>>> s="PYTHON"
>>> print(s)-----PYTHON
>>> s.lower()-----'python'
>>> s="PYThon"
>>> print(s)-----PYThon
>>> s.lower()-----'python'
```

6) isupper()

=>This Function returns True provided the given str object data is purely Upper Case otherwise it returns False.

Syntax: strobj.isupper()

Examples:

```
>>> s="PYTHON"
>>> s.isupper()-----True
>>> s="python"
>>> s.isupper()-----False
>>> s="Python"
>>> s.isupper()-----False
>>> s="PYThon"
>>> s.isupper()-----False
>>> s="123"
>>> s.isupper()-----False
>>> s="%$#^&@"
>>> s.isupper()-----False
```

7)islower()

=>This Function returns True provided the given str object data is purely lower Case otherwise it returns False.

Syntax: strobj.islower()

Examples:

```
>>> s="pythopn"
>>> s.islower()-----True
>>> s="pythOn"
>>> s.islower()-----False
>>> s="PYTHON"
>>> s.islower()-----False
>>> s="123"
>>> s.islower()-----False
```

8) isalpha()

=>This Function returns True provided str object contains Purely Alphabets otherwise returns False.

Syntax: strobj.isalpha()

Examples:

```
>>> s="Ambition"
>>> s.isalpha()-----True
>>> s="Ambition123"
>>> s.isalpha()-----False
>>> s="1234"
>>> s.isalpha()-----False
>>> s=" "
>>> s.isalpha()-----False
>>> s="#$%^@"
>>> s.isalpha()-----False
>>> s="AaBbZz"
>>> s.isalpha()-----True
```

9) isdigit()

=>This Function returns True provided given str object contains purely digits otherwise returns False

Examples:

```
>>> s="python"
>>> s.isdigit()-----False
```

```
>>> s="python123"
>>> s.isdigit()-----False
>>> s="123"
>>> s.isdigit()-----True
>>> s="123 456"
>>> s.isdigit()-----False
>>> s="1_2_3"
>>> s.isdigit()-----False
>>> s="123KV"
>>> s.isdigit()-----False
```

10) isalnum()

=>This Function returns True provided str object contains either Alphabets OR Numerics or Alpha-Numerics only otherwise It returns False.

=>Syntax: strobj.isalphanum()

=>Examples:

```
>>> s="python310"
>>> s.isalnum()-----True
>>> s="python"
>>> s.isalnum()-----True
>>> s="310"
>>> s.isalnum()-----True
>>> s="$python310"
>>> s.isalnum()-----False
>>> s="python 310"
>>> s.isalnum()-----False
>>> s="$python3.10"
>>> s.isalnum()-----False
>>> s="python3.10"
>>> s.isalnum()-----False
```

11) isspace()

=>This Function returns True provided str obj contains purely space otherwise it returns False.

=>Syntax: strobj.isspace()

Examples:

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

```
>>> s.isspace()-----True
>>> s=""
>>> s.isspace()-----False
>>> s="python Prog"
>>> s.isspace()-----False
>>> s="Prasana Laxmi"
>>> s.isspace()-----False
>>> s.isalpha()-----False
>>> s.isalpha() or s.isspace()-----False
```

12) split()

=>This Function is used for splitting the given str object data into different words base specified delimiter (- _ # % ^ ^ , ;etc)

=>The default deleimiter is space

=>The Function returns Splitting data in the form of list object

=>Syntax: strobj.split("Delimter")

(OR)

strobj.split()

(OR)

listobj= strobj.split("Delimter")

(OR)

listobj=strobj.split()

Examples:

```
>>> s="Python is an oop lang"
>>> print(s)-----Python is an oop lang
>>> s.split()-----['Python', 'is', 'an', 'oop', 'lang']
>>> len(s.split())-----5
>>> x=s.split()
>>> print(x,type(x))-----['Python', 'is', 'an', 'oop', 'lang'] <class 'list'>
>>> len(x)-----5
>>> s="12-09-2022"
>>> print(s)-----12-09-2022
>>> s.split("-")-----['12', '09', '2022']
>>> s="12-09-2022"
>>> dob=s.split("-")
>>> print(dob,type(dob))-----['12', '09', '2022'] <class 'list'>
>>> print("Day",dob[0])-----Day 12
>>> print("Month ",dob[1])-----Month 09
>>> print("Year ",dob[2])-----Year 2022
```

```
>>> s="Apple#Banana#kiwi/Guava"
```

```
>>> words=s.split("#")
>>> print(words)-----['Apple', 'Banana', 'kiwi/Guava']
>>> words=s.split("/")
>>> print(words)-----['Apple#Banana#kiwi', 'Guava']
```

13) join():

=>This Function is used for combining or joining list of values from any Iterable object

=>Syntax: strobj.join(Iterableobject)

Examples:

```
>>> lst=["HYD","BANG","AP","DELHI"]
>>> print(lst,type(lst))-----['HYD', 'BANG', 'AP', 'DELHI'] <class 'list'>
>>> s=""
>>> s.join(lst)-----'HYDBANGAPDELHI'
>>> s=" "
>>> s.join(lst)-----'HYD BANG AP DELHI'
```

```
>>> t=("Rossum","is", "Father" "of" , "Python")
>>> print(t,type(t))
('Rossum', 'is', 'Fatherof', 'Python') <class 'tuple'>
>>> k=" "
>>> k.join(t)
'Rossum is Fatherof Python'
>>> t=("Rossum","is", "Father", "of" , "Python")
>>> k=" "
>>> k.join(t)
'Rossum is Father of Python'
>>>
```

Functions in Python

Index

=>Purpose of Functions
=>Advantages of Functions
=>Definition of Functions
=>Parts of Functions
=>Phases in Functions
=>Syntax for Defining Functions
=>Number of Approaches for Defining Functions
=>Programming Examples

=>Arguments and Parameters
=>Types of Arguments and Parameters

- a) Positional Arguments and Parameters
- b) Default Arguments and Parameters
- c) Keyword Arguments and Parameters
- d) Variable Length Arguments and Parameters
- e) Keyword Variable Length Arguments and Parameters

=>Programming Examples

=>Global Variables and Local Variables
=>global keyword and globals()
=>Programming Examples
=>Anonymous Functions OR Lambda Functions
=>Programming Examples

=>Special Function in Python

- a) filter() with Normal and Anonymous Functions
- b) map() with Normal and Anonymous Functions
- c) reduce() with Normal and Anonymous Functions

=>Programming Examples

Types of Languages in the context of Functions

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

1. Un-Structured Programming Languages
 2. Structured Programming Languages
-

1. Un-Structured Programming Languages

=>Un-Structured Programming Languages does not contain the concept of Functions and hence whose applications having the following Limitations.

1. Application Development time is More

2. Application Memory Space is More
3. Application Execution Time is More
4. Application Performance is degraded
5. Redundancy of the code is More

Examples: GW-BASIC

2. Structured Programming Languages

=>Structured Programming Languages contains the concept of Functions and hence whose applications having 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. Redundancy of the code is Minimized

Examples: C,C++,Java, PYTHON.....etc

Functions in Python

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

=>Definition of Function:

=>A part of main program is called Function
(OR)

=>Sub Program of Main Program is called Function.

Parts of Functions

=>When we are dealing with Functions, we must ensure that , There must exist Two parts. They are

1. Function Definition
2. Function Call(s)

=>Here Particular Function Definition Exist Only Once

=>For Every Function Call There must Exist Function Definition otherwise we get NameError.

=> Function Definition can't Execute by itself but they are executed through a function call.
In

Otherwords Function Definition will be executed when we call the Function by using Function Call.

Phases In Functions

=>At thy time Defining the functions, we must ensure that there must exist 3 Phases.

1. Every Function must takes INPUT
 2. Every Function must PROCESS the input
 3. Every Function must give RESULT / OUTPUT
-

Syntax for Defining a Function in Python

```
def functionname(list of formal Params if any) : <---Function Heading
    """doc string"""
    statement-1
    statement-2
    -----
    statement-n
        <--Function Body
    Note: Function def=
          Function Heading +Function Body
```

Explanation:-

1. here 'def' is a keyword , which is used for defining Programmer-defined Functions.
 2. "functionname" represents a valid variable name and treated as function name and every function name is an object of type <class, 'function'>
 3. "list of formal params" represents list of variable names used in function heading and they are used for storing or holding the input(s) coming from function call(s).
 4. "'''doc string''' " represents documentation string and it used for giving or writing the description about functionality of function.
 5. The statement-1,statement-2....statement-n indentation block of statements and it is process the input or logic for problem solving and it is known Business Logic.
 6. In the Function Body, we use some special variables and they are called Local Variables and whose purpose is to store the temporary results.
 7. The values of Formal Params and Local Variables can be accessed only inside of corresponding Function Definition but not possible to access in other part of the program and in Other Function Definitions(Scope of Formal params and Local Var)
-
- =====X=====

Number of Approaches to Define a Function

=>If any Problem Statement is given then it be solved by using Function in 4 Approaches.
They are

#Approach1:

=>INPUT Taking From Function Call
=>PROCESSING done in Function Body
=>OUTPUT / RESULT giving to Function Call

Examples:

#Program for defining a function for addition of two numbers
#ApproachEx1.py
def addop(a,b): # Here 'a' and 'b' are called Formal Parameters

```
c=a+b # Here 'c' is called Local variable  
return c # here return statement is used for giving the result back
```

```
#main program  
x=float(input("Enter First Value:"))  
y=float(input("Enter Second Value:"))  
res=addop(x,y) # Function Call  
print("sum({},{})={}".format(x,y,res))
```

```
-----  
#Approach2:  
-----
```

```
=>INPUT Taking in Function Body  
=>PROCESSING done in Function Body  
=>OUTPUT / RESULT displaying in Function Body
```

Examples:

```
-----  
#Program for defining a function for addition of two numbers
```

```
#ApproachEx2.py
```

```
def addop():  
    a=float(input("Enter First Value:"))  
    b=float(input("Enter Second Value:"))  
    c=a+b  
    print("sum({},{})={}".format(a,b,c))
```

```
-----  
#main program
```

```
addop() # Function Call
```

```
-----  
#Approach3:  
-----
```

```
=>INPUT Taking in Function Body  
=>PROCESSING done in Function Body  
=>OUTPUT / RESULT giving to Function Call
```

```
-----  
#Program for defining a function for addition of two numbers
```

```
#ApproachEx3.py
```

```
def addop():  
    a=float(input("Enter First Value:"))  
    b=float(input("Enter Second Value:"))  
    c=a+b  
    return a,b,c # In Python, return stmt can return one or more values
```

```
-----  
#main program
```

```
a,b,c=addop() # Function Call with multi line assigment
```

```
print("sum({},{})={}".format(a,b,c))
```

```
print("-----")
```

```
kvr=addop() # Here kvr is an object of type tuple.  
print("sum({},{})={}".format(kvr[0],kvr[1],kvr[2]))
```

```
#Approach4:
```

```
=>INPUT Taking From Function Call  
=>PROCESSING done in Function Body  
=>OUTPUT / RESULT displaying in Function Body
```

```
#Program for defining a function for addition of two numbers
```

```
#ApproachEx4.py
```

```
def addop(a,b):  
    c=a+b  
    print("sum({},{})={}".format(a,b,c))
```

```
#main program
```

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

```
#Program for defining a function for addition of two numbers
```

```
#Approach1.py
```

```
def addop(a,b): # Here 'a' and 'b' are called Formal Parameters
```

```
    print("Line-5: I am from addop())"  
    c=a+b # Here 'c' is called Local variable  
    return c # here return statement is used for giving the result back
```

```
#main program
```

```
print("Line-10: I am from Main Program:")  
res=addop(10,20) # Function Call  
print("Line-12--sum=",res)  
print("\nLine-13: I am from Main Program:")  
res=addop(100,200) # Function Call  
print("Line-15--sum=",res)  
print("\nLine-16: I am from Main Program:")  
res=addop(-10,-20) # Function Call  
print("Line-18--sum=",res)
```

```
#Program for defining a function for addition of two numbers
```

```
#ApproachEx1.py
```

```
def addop(a,b): # Here 'a' and 'b' are called Formal Parameters  
    c=a+b # Here 'c' is called Local variable  
    return c # here return statement is used for giving the result back
```

```
#main program
```

```
x=float(input("Enter First Value:"))  
y=float(input("Enter Second Value:"))  
res=addop(x,y) # Function Call  
print("sum({},{})={}".format(x,y,res))
```

```
#Program for defining a function for addition of two numbers
#ApproachEx2.py
def addop():
    a=float(input("Enter First Value:"))
    b=float(input("Enter Second Value:"))
    c=a+b
```

```
    print("sum({},{})={}".format(a,b,c))
```

```
#main program
addop() # Function Call
```

```
#Program for defining a function for addition of two numbers
```

```
#ApproachEx3.py
```

```
def addop():
    a=float(input("Enter First Value:"))
    b=float(input("Enter Second Value:"))
    c=a+b
```

```
    return a,b,c # In Python, return stmt can return one or more values
```

```
#main program
```

```
a,b,c=addop() # Function Call with multi line assigment
```

```
print("sum({},{})={}".format(a,b,c))
```

```
print("-----")
```

```
kvr=addop() # Here kvr is an object of type tuple.
```

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

```
#Program for defining a function for addition of two numbers
```

```
#ApproachEx4.py
```

```
def addop(a,b):
    c=a+b
```

```
    print("sum({},{})={}".format(a,b,c))
```

```
#main program
```

```
a=float(input("Enter First Value:"))
```

```
b=float(input("Enter Second Value:"))
```

```
addop(a,b) # Function Call
```

```
#Program generating Mul Table for a give number by using Functions.
```

```
#MulTable.py
```

```
def table(n):
```

```
    if(n<=0):
```

```
        print("{} is invalid input".format(n))
```

```
    else:
```

```
        print("-"*50)
```

```
        print("Mu Table for {}".format(n))
```

```
        print("-"*50)
```

```
        for i in range(1,11):
```

```
            print("\t{} x {} = {}".format(n,i,n*i))
```

```
        else:
```

```
            print("-"*50)
```

```
#main program
```

```
n=int(input("Enter a value:"))
```

```
table(n) # Function call


---


#rectareaperi.py
def area():
    l=float(input("Enter Length for cal area of Rect:"))
    b=float(input("Enter Breadth for cal area of Rect:"))
    area=l*b
    print("Area of Rect=",area)

def peri():
    l=float(input("Enter Length for cal peri of Rect:"))
    b=float(input("Enter Breadth for cal peri of Rect:"))
    peri=2*(l+b)
    print("Perimter of Rect=",peri)

#main program
peri()
area()


---


#write a python program which will accept list of values and find sum and average.
# HINT 11=[10,20,30,40]
# OUTPUT : sum=100 and AVG=25
#SumAvg.py

def readvalues():
    lst=list() # create an empyp list
    n=int(input("Enter How Many values u want fint sum and Avg:"))
    if(n<=0):
        return lst # returning empty list
    else:
        for i in range(1,n+1):
            val=float(input("Enter {} Value:".format(i)))
            lst.append(val)
    return lst # returning non-empty list

def sumavg(lst): # By using Logic
    print("-"*50)
    print("List of Values:")
    print("-"*50)
    for val in lst:
        print("\t{}".format(val))
    print("-"*50)
    #Find sum and avg
    s=0
    for val in lst:
        s=s+val
    else:
        print("-"*50)
        print("Sum=",s)
        print("Avg=",s/len(lst))
```

```

        print("-"*50)

#main program
lstobj=readvalues() # Function Call
if(len(lstobj)==0):
    print("List is Empty-- and can't find sum and Avg")
else:
    sumavg(lstobj) # Function Call


---


#write a python program which will accept list of values and find sum and average.
# HINT  l1=[10,20,30,40]
# OUTPUT : sum=100  and AVG=25
#SumAvg1.py

def  readvalues():
    lst=list() # create an empty list
    n=int(input("Enter How Many values u want fint sum and Avg:"))
    if(n<=0):
        return lst # returning empty list
    else:
        for i in range(1,n+1):
            val=float(input("Enter {} Value:".format(i)))
            lst.append(val)
    return lst # returning non-empty list

def  sumavg(lst): # By using pre-defined sum()
    print("Sum({})={}".format(lst, sum(lst)) )
    print("Avg={}".format(sum(lst)/len(lst)) )

#main program
lstobj=readvalues() # Function Call
if(len(lstobj)==0):
    print("List is Empty-- and can't find sum and Avg")
else:
    sumavg(lstobj) # Function Call


---


#write a python program which will accept list of numerical values and sort them in
ascending and descending order.
#SortData.py
def readvalues():
    lst=list() # create an empty list
    n=int(input("Enter How Many values u want Sort:"))
    if(n<=0):
        return lst # returning empty list
    else:
        for i in range(1,n+1):
            val=float(input("Enter {} Value:".format(i)))
            lst.append(val)
    return lst # returning non-empty list

def  sortdata(lst):

```

```

print("Orginal Data={}".format(lst))
lst.sort()
print("ASC Order :{}".format(lst))
lst.sort(reverse=True)
print("DESC Order :{}".format(lst))

#main program
listobj= readvalues()
if(len(listobj)==0):
    print("List is Empty-- and can't sort")
else:
    sortdata(listobj)


---


# Program for displaying the values of Iterable Objects by using Functions
#IterableObjects.py
def disp(kvr):
    print("-"*50)
    print("Type of kvr=",type(kvr))
    for val in kvr:
        print("\t{}".format(val))
    print("-"*50)

def dispdictvalues(kvr):
    print("-"*50)
    print("Type of kvr=",type(kvr))
    for key,val in kvr.items():
        print("\t{}-->{}".format(key,val))
    print("-"*50)

#main program
lst=[10,"Rossum",34.56,True,2-3.4j]
disp(lst) # Function Call
tpl=(10,20,30,40,23.45)
disp(tpl) # Function Call
s={10,"Rajesh",45.67}
disp(s) # Function Call
d1={10:"Rossum",20:"Travis",30:"Kinney"}
dispdictvalues(d1)# Function Call


---


#write a Python program which will accept list of values and find highest element and lowest
element
#MaxMinEx1.py
def readvalues():
    lst=list() # create an empty list
    n=int(input("Enter How Many values u want finf max and min:"))
    if(n<=0):
        return lst # returning empty list
    else:
        for i in range(1,n+1):
            val=float(input("Enter {} Value:".format(i)))
            lst.append(val)

```

```

        return lst # returning non-empty list

def kvrmax(lst):
    #Logic for max
    maxv=lst[0]
    for i in range(1,len(lst)):
        if (lst[i] > maxv):
            maxv=lst[i]
    return maxv

def kvrmin(lst):
    print("-"*50)
    #Logic for min
    minv=lst[0]
    for i in range(1,len(lst)):
        if (lst[i] < minv):
            minv=lst[i]
    return minv

#main program
listobj=readvalues()
if(len(listobj)==0):
    print("List is Empty-- and can't find max and min")
else:
    if(len(set(listobj))==1):
        print("\nMax({})={}".format(listobj,"ALL VALUES ARE EQUAL"))
        print("\nMin({})={}".format(listobj,"ALL VALUES ARE EQUAL"))
    else:
        mv=kvrmax(listobj) # Function Call
        print("\nMax({})={}".format(listobj,mv))
        minv=kvrmin(listobj) # Function Call
        print("\nMin({})={}".format(listobj,minv))

```

Arguments and Parameters

=>Parameters represents list of Variables used in Function Heading and they are used for Storing the inputs coming Functions Calls and these Parameters are called Formal Parameters.

=>Arguments are the Variables used inside of Function Call(s) and they are also called Actual Arguments.

Syntax for Function Definition

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

Syntax for Function Call

functionname(args1,args2.....args-n)

=>Hence relationship between arguments and Parameters is that Every Value of arguments are passing Parameters.

Types of Arguments and Parameters

=>Based on Passing the values of Arguments from Function Calls to Parameters of Function Definition, Arguments and Parameters are classified into 5 types. They are

1. Positional Arguments OR Parameters
 2. Default Arguments OR Parameters
 3. Keyword Arguments OR Parameters
 4. Variable Length Arguments OR Parameters
 5. Keyword Variable Length Arguments OR Parameters
-

1) Positional Arguments (or) Parameters

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

=>This Parameter mechanism also recommends Order and Meaning of Parameters for Higher accuracy.

=>To pass the Specific Data from Function Call to Function Definition then we must take Positional Argument Mechanism.

=>The default Argument Passing Mechanism is Positional Arguments (or) Parameters.

Syntax for Function Definition :

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

Syntax for Function Call:

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

=>Here the values of arg1,arg2...arg-n are passing to param-1,param-2..param-n respectively.
=>PVM gives First Priority for Positional Arguments (or) Parameters

```
#Program for demonstarting Possitional Arguments OR Parameters  
#PossArgsEx1.py  
def dispstudinfo(sno,sname,marks):  
    print("\t{}\t{}\t{}".format(sno,sname,marks))
```

```
#main program
print("-"*50)
print("\tSno\tName\tMarks")
print("-"*50)
dispstudinfo(10,"RS",34.56)
dispstudinfo(20,"TR",54.56)
dispstudinfo(30,"RJ",14.56)
print("-"*50)


---


#Program for demonstarting Possitional Arguments OR Parameters
#PossArgsEx2.py
def dispstudinfo(sno,sname,marks,crs):
    print("\t{}\t{}\t{}\t{}".format(sno,sname,marks,crs))
```

```
#main program
print("-"*50)
print("\tSno\tName\tMarks\tCourse")
print("-"*50)
dispstudinfo(10,"RS",34.56,"PYTHON")
dispstudinfo(20,"TR",54.56,"PYTHON")
dispstudinfo(30,"RJ",14.56,"PYTHON")
dispstudinfo(40,"NN",24.56,"PYTHON")
dispstudinfo(50,"MM",34.56,"PYTHON")
print("-"*50)


---


```

2) Default Parameters (or) arguments

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

Syntax: for Function Definition with Default Parameters

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

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

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

```
#Program for demonstartng Default Arguments OR Parameters
#DefArgsEx1.py
def dispstudinfo(sno,sname,marks,crs="PYTHON"):
    print("\t{}\t{}\t{}\t{}".format(sno,sname,marks,crs))
```

```
#main program
```

```

print("-"*50)
print("\tSno\tName\tMarks\tCourse")
print("-"*50)
dispstudinfo(10,"RS",34.56)
dispstudinfo(20,"TR",54.56)
dispstudinfo(30,"RJ",14.56)
dispstudinfo(40,"NN",24.56)
dispstudinfo(50,"MM",34.56)
dispstudinfo(60,"KV",14.56,"JAVA")
dispstudinfo(70,"AV",54.56)
print("-"*50)


---


#Program for demonstarting Default Arguments OR Parameters
#DefArgsEx2.py
def dispstudinfo(sno,sname,marks,crs="PYTHON",cnt="INDIA"):
    print("\t{}\t{}\t{}\t{}\t{}".format(sno,sname,marks,crs,cnt))



---


#main program
print("-"*50)
print("\tSno\tName\tMarks\tCourse\tCountry")
print("-"*50)
dispstudinfo(10,"RS",34.56)
dispstudinfo(20,"TR",54.56)
dispstudinfo(30,"RJ",14.56)
dispstudinfo(40,"NN",24.56)
dispstudinfo(50,"MM",34.56)
dispstudinfo(60,"KV",14.56,"JAVA")
dispstudinfo(70,"AV",54.56)
dispstudinfo(80,"DT",4.56,"PHP","USA")
dispstudinfo(90,"JB",64.56,cnt="USA")
dispstudinfo(85,"AF",34.56,"PYTHON,JAVA","UK")
print("-"*50)


---


#Program for demonstarting Default Arguments OR Parameters
#DefArgsEx3.py
def dispstudinfo1(sno,sname,marks,cnt="INDIA",crs="PYTHON"):
    print("\t{}\t{}\t{}\t{}\t{}".format(sno,sname,marks,crs,cnt))

def dispstudinfo2(sno,sname,marks,crs="JAVA",cnt="HYD"):
    print("\t{}\t{}\t{}\t{}\t{}".format(sno,sname,marks,crs,cnt))



---


#main program
print("-"*50)
print("\t\t List of Python Students")
print("\tSno\tName\tMarks\tCourse\tCountry")
print("-"*50)
dispstudinfo1(10,"RS",34.56)
dispstudinfo1(30,"RJ",14.56)
dispstudinfo1(50,"MM",34.56)
dispstudinfo1(80,"DT",4.56,"PHP","USA")
dispstudinfo1(90,"JB",64.56,cnt="USA")

```

```

dispstudinfo2(85,"AF",34.56,"PYTHON,JAVA","UK")
print("-"*50)
print("\t\t List of Java Students")
print("\tSno\tName\tMarks\tCourse\tCountry")
print(" "*50)
dispstudinfo2(20,"TR",54.56)
dispstudinfo2(40,"NN",24.56)
dispstudinfo2(60,"KV",14.56,"JAVA")
dispstudinfo2(70,"AV",54.56)
dispstudinfo2(60,"KV",14.56,"JAVA")
dispstudinfo2(70,"AV",54.56)
dispstudinfo2(85,"AF",34.56,"PYTHON,JAVA","UK")
print("-"*50)


---


#Program for cal area of circle using Functions
#DefArgsEx4.py
def areacircle(r,PI=3.14):
    area=PI*r**2
    print("Area of Circle=",area)

```

```

#Main program
areacircle(1.2)
areacircle(2.2)
areacircle(2)
areacircle(4.2)
areacircle(float(input("Enter Radious:")))


---


=====
```

3) Keyword Parameters (or) arguments

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

Syntax for function definition:-

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

Syntax for function call:-

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

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

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

#Program for demonstarting Keyword args
#KwdArgsEx1.py

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

#main program
print("*50)
print("\tA\tB\tC")
print("-*50)
disp(10,20,30) # Function Call--Positional args
disp(c=30,b=20,a=10) # Function Call--keyword args
disp(b=20,c=30,a=10) # Function Call--keyword args
disp(10,c=30,b=20)# Function Call--Positional keyword args
disp(10,20,c=30)# Function Call--Positional keyword args
#disp(b=20,10,c=30) # SyntaxError: positional argument follows keyword argument
print("-*50)

#Program for demonstarting Keyword args
#KwdArgsEx2.py
def disp(a,b,c,E=2.71):
    print("\t{}\t{}\t{}\t{}".format(a,b,c,E))
```

```
#main program
print("-*50)
print("\tA\tB\tC\tE")
print("-*50)
disp(10,20,30) # Function Call--Positional args
disp(c=30,b=20,a=10) # Function Call--keyword args
disp(b=20,c=30,a=10) # Function Call--keyword args
disp(10,c=30,b=20)# Function Call--Positional keyword args
disp(10,20,c=30)# Function Call--Positional keyword args
#disp(b=20,10,c=30) # SyntaxError: positional argument follows keyword argument
disp(c=30,E=3.41,a=10,b=20)# Function Call--Positional ,keyword args, default
print("-*50)
```

4) Variables Length Parameters (or) arguments

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

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

Syntax for function definition with Variables Length Parameters:

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

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

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

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

```
#Program for demonstrating Var arguments concept
#VarAgrsEx1.py-----This Program will not execute as it is
def disp(a):
    print(a)

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

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

def disp(a,b,c,d):
    print("{}\t{}\t{}\t{}".format(a,b,c,d))
```

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

```
#Program for demonstrating Var arguments concept
#VarAgrsEx2.py-----This Program will execute as it is
def disp(a):
    print(a)
```

```
disp(10) # Function call

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

disp(10,20) # Function Call

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

disp(10,20,30) # Function Call

def disp(a,b,c,d):
    print("{}\t{}\t{}\t{}".format(a,b,c,d))
```

```



---


disp(10,20,30,40) # Function Call
#Program for demonstrating Var arguments concept
#PureVarAgrsEx1.py
def disp( *kvr) : # *kvr is called Variable Length Param--Holds variable number of values
    print(kvr,type(kvr),len(kvr))

#main program
disp(10) # Function call
disp(10,20) # Function Call
disp(10,20,30) # Function Call
disp(10,20,30,40) # Function Call
disp()
disp(10,"RS","Python","DS",34.56,True)


---


#Program for demonstrating Var arguments concept
#PureVarAgrsEx2.py
def disp( *kvr) : # *kvr is called Variable Length Param--Holds variable number of values
    print("-"*40)
    print("WAY-1:Type of kvr={} and Number of"
values={}.format(type(kvr),len(kvr)))
    print("-"*40)
    for val in kvr:
        print("\t{}".format(val))
    print("-"*40)
    print("WAY-2:Type of kvr={} and Number of"
values={}.format(type(kvr),len(kvr)))
    print("-"*40)
    for i in range(0,len(kvr)):
        print("\t{}".format(kvr[i]))
    print("-"*40)
#main program
disp(10) # Function call
disp(10,20) # Function Call
disp(10,20,30) # Function Call
disp(10,20,30,40) # Function Call
disp()
disp(10,"RS","Python","DS",34.56,True)


---


#Write a python program which will calculate sum and average of variable number of values
by various person
#PureVarAgrsEx2.py
def findsumavg(sname,*k):
    print("-"*50)
    print("Name of Student:{} ".format(sname))
    print("{} , Having {} Values:".format(sname, len(k)))
    print("-"*50)
    s=0
    for v in k:
        print("\t{}\n".format(v),end=" ")
        s=s+v
    else:
        print()

```

```

        print("-"*50)
        if(len(k)!=0):
            print("Sum={}".format(s))
            print("Avg={}".format(s/len(k)))
        else:
            print("Sum={}".format(s))
        print("-"*50)

#main program
findsumavg("Naveen",10,20,30,40)
findsumavg("Muzamil",100,200,300)
findsumavg("Sarat",1.2,3.4,4.5,5.6,7.8)
findsumavg("Rossum",10,20.5)
findsumavg("KVR")


---


#Write a python program which will calculate sum and average of variable number of values by various person
#PureVarAgrsEx4.py
def findsumavg(sname,*k,cnt="HYD"):
    print("-"*50)
    print("Name of Student:{} and living in {}".format(sname,cnt))
    print("{} , Having {} Values:{}".format(sname, len(k)))
    print("-"*50)
    s=0
    for v in k:
        print("\t{}".format(v),end=" ")
        s=s+v
    else:
        print()
        print("-"*50)
        if(len(k)!=0):
            print("Sum={}".format(s))
            print("Avg={}".format(s/len(k)))
        else:
            print("Sum={}".format(s))
        print("-"*50)

#main program
findsumavg("Naveen",10,20,30,40)
findsumavg("Muzamil",100,200,300)
findsumavg("Sarat",1.2,3.4,4.5,5.6,7.8)
findsumavg("Rossum",10,20.5,cnt="Nether Lands")
findsumavg("KVR",cnt="AP")
#findsumavg("TRAVIS",cnt="Finland", 23)--SyntaxError: positional argument follows keyword argument
findsumavg("TRAVIS", 23,cnt="Finland")
#findsumavg(sname="Kinney",2,3,5,6,cnt="USA")--SyntaxError: positional argument follows keyword argument
#findsumavg(2,3,5,6,cnt="USA",sname="Kinney")--TypeError: findsumavg() got multiple values for argument 'sname'
findsumavg("Kinney",2,3,5,6,cnt="USA")


---



```

5) Key Word Variables Length Parameters (or) arguments

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

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

Syntax for function definition with Keyword Variables Length Parameters:

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

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

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

Final Syntax for defining a Function

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

X

```
#Program for demonstrating keyword variable length args
#KwdVarArgsEx1.py---This Program will not execute as it is
def dispinfo(a,b,c,d): # Function Def-1
    print("\t{}\t{}\t{}\t{}".format(a,b,c,d))

def dispinfo(k,v,r): # Function Def-2
    print("\t{}\t{}\t{}\t{}".format(k,v,r))

def dispinfo(P,Q): # Function Def-3
    print("\t{}\t{}\t{}\t{}".format(P,Q))
#main program
dispinfo(a=10,b=20,c=30,d=40) # Function Call-1
dispinfo(k=100,v=200,r=300) # Function Call-2
dispinfo(P=1.2,Q=2.3) # Function Call-3
```

```

#Program for demonstrating keyword variable length args
#KwdVarArgsEx2.py---This Program will execute as it is
def dispinfo(a,b,c,d): # Function Def-1
    print("\t{}\t{}\t{}\t{}".format(a,b,c,d))

dispinfo(a=10,b=20,c=30,d=40) # Function Call-1

def dispinfo(k,v,r): # Function Def-2
    print("\t{}\t{}\t{}".format(k,v,r))

dispinfo(k=100,v=200,r=300) # Function Call-2

def dispinfo(P,Q): # Function Def-3
    print("\t{}\t{}".format(P,Q))

dispinfo(P=1.2,Q=2.3) # Function Call-3

```

```

#Program for demonstrating keyword variable length args
#PureKwdVarArgsEx1.py

def dispinfo( **hyd): # Here **hyd is called Keyword Variable length param---dict
    print(hyd,type(hyd), len(hyd))

```

```

#main program
dispinfo(a=10,b=20,c=30,d=40) # Function Call-1
dispinfo(k=100,v=200,r=300) # Function Call-2
dispinfo(P=1.2,Q=2.3) # Function Call-3

```

```

#Program for demonstrating keyword variable length args
#PureKwdVarArgsEx2.py

```

```

def dispinfo( **hyd): # Here **hyd is called Keyword Variable length param---dict
    print("-"*50)
    print("Type of hyd=",type(hyd))
    print("-"*50)
    for k,v in hyd.items():
        print("\t{}-->{}".format(k,v))
    print("-"*50)

```

```

#main program
dispinfo(a=10,b=20,c=30,d=40) # Function Call-1
dispinfo(k=100,v=200,r=300) # Function Call-2
dispinfo(P=1.2,Q=2.3) # Function Call-3
dispinfo(P1="Ranjan")
dispinfo()

```

```

#write a python program which will compute the Total marks of different subjects who are
securing in different classes and studying in various classes.

```

```

#PureKwdVarArgsEx3.py
def findtotalmarks(sno,sname,cls,cnt="INDIA",**marks):
    print("-"*50)

```

```

print("Roll Number :{}".format(sno))
print("Student Name :{}".format(sname))
print("Class Name :{}".format(cls))
print("Country:{}".format(cnt))
print("-"*50)
totmarks=0
for sn,sm in marks.items():
    print("\t{}\t{}".format(sn,sm))
    totmarks=totmarks+sm
else:
    print("-"*50)
    print("\tTotal Marks={}".format(totmarks))
    print("-"*50)

#main program
findtotalmarks(10,"Ranjan","X",Eng=60,Hindi=70,Telugu=56,Maths=80,Science=78,Social=67)
findtotalmarks(20,"Minhaj","XII",Mathematics=70,Physics =58,Chemistry=50)
findtotalmarks(30,"Shilpa","B.Tech(CSE)",CM=70,Cpp=60,Python=50,DBMS=50)
findtotalmarks(40,"Rossum","Research")

```

#write a python program which will compute the Total marks of different subjects who are securing in different classes and studying in various classes.

```

#PureKwdVarArgsEx4.py
def findtotalmarks(sno,sname,cls,cnt="INDIA",**marks):
    print("-"*50)
    print("Roll Number :{}".format(sno))
    print("Student Name :{}".format(sname))
    print("Class Name :{}".format(cls))
    print("Country:{}".format(cnt))
    print("-"*50)
    totmarks=0
    for sn,sm in marks.items():
        print("\t{}\t{}".format(sn,sm))
        totmarks=totmarks+sm
    else:
        print("-"*50)
        print("\tTotal Marks={}".format(totmarks))
        print("-"*50)

#main program
findtotalmarks(10,"Ranjan","X",Eng=60,Hindi=70,Telugu=56,Maths=80,Science=78,Social=67)
findtotalmarks(20,"Minhaj","XII",Mathematics=70,Physics =58,Chemistry=50)
findtotalmarks(30,"Shilpa","B.Tech(CSE)",CM=70,Cpp=60,Python=50,DBMS=50)
findtotalmarks(40,"Rossum","Research")

```

#write a python program which will compute the Total marks of different subjects who are securing in different classes and studying in various classes.

```

#PureKwdVarArgsEx5.py
def findtotalmarks(sno,sname,cls, *v, cnt="INDIA",**marks):

```

```

print("*"*50)
print("Variable Length Values:{}".format(len(v)))
print("*"*50)
for val in v:
    print("\t{}".format(val))
print("*"*50)
print("-"*50)
print("Roll Number :{}".format(sno))
print("Student Name :{}".format(sname))
print("Class Name :{}".format(cls))
print("Country:{}".format(cnt))
print("-"*50)
totmarks=0
for sn,sm in marks.items():
    print("\t{}\t{}".format(sn,sm))
    totmarks=totmarks+sm
else:
    print("-"*50)
    print("\tTotal Marks={}".format(totmarks))
    print("-"*50)
#main program
findtotalmarks(10,"Ranjan","X",10,20,30,Eng=60,Hindi=70,Telugu=56,Maths=80,Science=78,Social=67)
findtotalmarks(20,"Minhaj","XII",100,200,Mathematics=70,Physics =58,Chemistry=50)
findtotalmarks(30,"Shilpa","B.Tech(CSE)",1.2,1.3,1.4,1.5,CM=70,Cpp=60,Python=50,DBM S=50)
findtotalmarks(40,"Rossum","Research",15,25)
findtotalmarks(cnt="USA",cls="VIII",sno=50,sname="Shruthi",Sub1=20, Sub2=30)
findtotalmarks(cls="V",sno=60,sname="Akshy")
findtotalmarks(cls="VIII",sno=50,sname="Raj",Sub3=70,cnt="UK")

```

Global Variables and Local Variables

Local Variables

- =>Local Variables are those which are used in side of Function Body and they are used for storing Temporary result of Processing Logic.
 - =>We can access the value of Local Variables Inside of Function Body only but not other part of the program.
-

Global Variables

- =>Global variables are those are they are used for accesssing as common values in Multiple Different function Calls
- =>Global Variables Must be defined before all Function Calls. so that we can Global Variables values in correspondinf Function Definitions.
- =>if we define Global Variables after all Function Calls then we can't access Global Variables values in correspondinf Function Definitions(NameError we get).

```
=====x=====
=====
#Program for demonstarting Local anf Global variables
#GlobalLocalVarEx1.py
lang="PYTHON" # Here lang is called Global Variable
def learnDS():
    sub1="Data Science"
    print("\tTo Develop '{}' Apps, we use '{}' Programming".format(sub1,lang))

def learnML():
    sub2="Machine Learning"
    print("\tTo Develop '{}' Apps, we to use '{}' Programming".format(sub2,lang))

def learnDL():
    sub3="Deep Learning"
    print("\tTo Develop '{}' Apps, we to use '{}' Programming".format(sub3,lang))

#main program
learnDS()
learnML()
learnDL()
=====
#Program for demonstarting Local anf Global variables
#GlobalLocalVarEx2.py
def learnDS():
    sub1="Data Science"
    print("\tTo Develop '{}' Apps, we use '{}' Programming".format(sub1,lang))

lang="PYTHON" # Here lang is called Global Variable

def learnML():
    sub2="Machine Learning"
    print("\tTo Develop '{}' Apps, we to use '{}' Programming".format(sub2,lang))

def learnDL():
    sub3="Deep Learning"
    print("\tTo Develop '{}' Apps, we to use '{}' Programming".format(sub3,lang))

#main program
learnDS()
learnML()
learnDL()
=====
#Program for demonstarting Local anf Global variables
#GlobalLocalVarEx3.py
def learnDS():
    sub1="Data Science"
    print("\tTo Develop '{}' Apps, we use '{}' Programming".format(sub1,lang))

def learnML():
    sub2="Machine Learning"
    print("\tTo Develop '{}' Apps, we to use '{}' Programming".format(sub2,lang))
```

```

def learnDL():
    sub3="Deep Learning"
    print("\tTo Develop '{}' Apps, we to use '{}' Programming".format(sub3,lang))

#main program
lang="PYTHON" # Here lang is called Global Variable
learnDS()
learnML()
learnDL()


---


#Program for demonstarting Local anf Global variables
#GlobalLocalVarEx4.py
def learnDS():
    sub1="Data Science"
    print("\tTo Develop '{}' Apps, we use '{}' Programming".format(sub1,lang))

def learnML():
    sub2="Machine Learning"
    print("\tTo Develop '{}' Apps, we to use '{}' Programming".format(sub2,lang))

def learnDL():
    sub3="Deep Learning"
    print("\tTo Develop '{}' Apps, we to use '{}' Programming".format(sub3,lang))

#main program
#learnDS()
#learnML()
#learnDL()
lang="PYTHON" # Here lang is called Global Variable

# Here we can't access lang variable Value (Global Variable ) in Those function def which
called before.

```

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

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

Syntax:

```

-----
var1=val1
var2=val2
var-n=val-n # var1,var2...var-n are called global variable names.

-----
def fun1():

-----
        global var1,var2...var-n
        # Modify var1,var2....var-n
-----
```

```

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

```

Examples:

```

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

```

#globalvarex2.py
a=10
def access1():
    global a # refering global Varaiable before its updation / Modification
    a=a+1 # Here we are modifying the global variable value then we need to use global
    print("Val of a inside of access1()=",a) # 11
#main program
print("Val of a in main before access1():",a) # 10
access1()
print("Val of a in main after access1():",a) # 11
-----
```

Examples:

```

-----#globalvarex3.py
def update1():
    global a,b # refering global Variables.
    a=a+1 #updating global Variable a
    b=b+1 #updating global Variable b
def update2():
    global a,b # refering global Variables.
    a=a*10 #updating global Variable a
    b=b*10 #updating global Variable b

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

```

print("Val of a={} and Value of b={} in main program after update2():".format(a,b))
#Val of a=20 and Value of b=30 in main program after update1():
=====X=====
====

#Program demonstrating global keyword functionality.
#GlobalKwdEx1.py
a=0 # Global Variable
def update1():
    global a
    a=a+1
#main program
print("Val of a in main program before update()=",a) # 0
update1()
print("Val of a in main program after update()=",a) # 1
=====

#Program demonstrating global keyword functionality.
#GlobalKwdEx2.py
a=10 # Global Variable
def update1():
    global a
    a=a+1

def update2():
    global a
    a=a*2

#main program
print("Val of a in main program before update1()=",a) # 10
update1()
print("Val of a in main program after update1()=",a) # 11
update2()
print("Val of a in main program after update2()=",a) # 22
=====

#Program demonstrating global keyword functionality.
#GlobalKwdEx3.py

def modification():
    global a,b # here we are refering Global Varaibles and doing modifications
    a=a+1
    b=b+2

#main program
a,b=10,20 # Here 'a' and 'b' are called Global Variables
print("Val of a in main program Before modification()=",a) # 10
print("Val of b in main program Before modification()=",b) # 20
modification()
print("Val of a in main program after modification()=",a) # 11
print("Val of b in main program after modification()=",b) # 22
=====

global and local variables and globals()
=====
```

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

=>Syntax:-

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

=====

Examples:

=====

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

```

print("Val of d=", obj.get('d'))
print("=*50)
print("\nProgrammer-defined Global Variables")
print("=*50)
print("Val of a=", globals().get('a'))
print("Val of b=", globals().get('b'))
print("Val of c=", globals().get('c'))
print("Val of d=", globals().get('d'))
print("=*50)
print("\nProgrammer-defined Global Variables")
print("=*50)
print("Val of a=", globals()['a'])
print("Val of b=", globals()['b'])
print("Val of c=", globals()['c'])
print("Val of d=", globals()['d'])
print("=*50)
=====
```

#main program
operations()

=====
Examples:

```

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

#main program
operation()

```

=====X=====
=====#program demonstarting globals()
#NOTE: globals() is used for obtaining Global Variables Names and Values ( Visible and
Invisible) in the form dict type. gloablsls() returns dict object and it contains Global
Variables Names and Values in the form of (Key,Value)
#globalsFunEx1.py
a=10
b=20
def getvalues():
    dictobj=globals()
```

```

print("type of dict obj=",type(dictobj))
print("Number of Global Var names and Values=", len(dictobj))
print("-----")
for gvn,gvv in dictobj.items():
    print("\t{}---->{}".format(gvn,gvv))
print("-----")
print("\nProgrammer-Defined Global Variable Names and Values-Way-1")
print("-----")
print("Val of a--Global Variable={}".format(dictobj['a'])) # 10
print("Val of a--Global Variable={}".format(dictobj['b'])) # 20
print("-----")
print("\nProgrammer-Defined Global Variable Names and Values-Way-2")
print("-----")
print("Val of a--Global Variable={}".format(dictobj.get('a')))
print("Val of a--Global Variable={}".format(dictobj.get('b')))
print("-----")
print("\nProgrammer-Defined Global Variable Names and Values-Way-3")
print("-----")
print("Val of a--Global Variable={}".format( globals()['a'] ) )
print("Val of b--Global Variable={}".format( globals()['b'] ) )
print("-----")
print("\nProgrammer-Defined Global Variable Names and Values-Way-4")
print("-----")
print("Val of a--Global Variable={}".format( globals().get('a') ))
print("Val of b--Global Variable={}".format( globals().get('b')))
```

```

#main program
getvalues()


---


#program demonstarting globals()
#globalsFunEx2.py
a=10
b=20
c=30
d=40 # Here a,b,c,d are called Global Variable Names
def operation():
    x1=10
    x2=20
    x3=30
    x4=40 # Here x1,x2,x3,x4 are called Local Variable Names
    res=a+b+c+d+x1+x2+x3+x4
    print("sum of local and global var vals=",res)
```

```

#main program
operation()


---


#program demonstarting globals()---globals()
#globalsFunEx3.py
a=10
b=20
c=30
```

```

d=40 # Here a,b,c,d are called Global Variable Names
def operation():
    a=100
    b=200
    c=300
    d=400 # Here a,b,c,d are called Local Variable Names
    res=a+b+c+d+globals()['a']+globals()['b']+globals()['c']+globals()['d']
    print("sum of local var vals=",res)
    print("=====OR=====")
    res=a+b+c+d+globals().get('a')+globals().get('b')+globals().get('c')+globals().get('d')
    print("sum of local var vals=",res)

#main program
operation()

```

Anonymous Functions OR Lambda Functions

- =>Anonymous Functions are those which does not contain Name Explicitly.
 - =>The purpose of Anonymous Functions is that " To Perform Instant Operations".
 - =>Instant Operations are those Which are used at that Point Time Only But No Longer interested to use next Part of the project".
 - =>To define Anonymous Functions , we use lambda keyword and hence Anonymous Functions are called Lamda Functions.
 - =>Anonymous Functions contains Single executable Statement Only But never contains Multiple Executable Statements.
 - =>Anonymous Functions automatically or Implcitly returns the value (No Need to use return statement)
-

Syntax: varname=lambda params-list : statement

Explanation

- =>Here Varname is an object of <class, 'function'> and varname indirectly treated as Anonymous Function Name.
 - =>Here lambda is Keyword used for Defining Anonymous Functions
 - =>Params-list represents list of Variable Names used for Holding / Storing the inputs coming from Functions.
 - =>Gere Stetement represents Single Executable Statement and whose Value returns Automatically or Implicitly.
-

Question: Define a Function for adding two values

Normal Function---Definition	Anonymous Function
<pre> def sumop(a,b): c=a+b return c </pre>	<pre> sumop=lambda a,b:a+b </pre>
Main program	main program

```
-----  
-----  
res=sumop(10,20) # Function Call  
print(res)---30  
res=sumop(3,4)  
print(res)
```

```
#Program defining Function for addition of two values  
#AnonymousFunEx1.py  
def sumop(a,b): # Normal Function  
    c=a+b  
    return c  
  
addop=lambda x,y: x+y # Anonymous Function definition
```

```
#main program  
a,b=int(input("Enter First Value:")),int(input("Enter Second Value:"))  
res=sumop(a,b)  
print("Type of sumop--Normal Function=",type(sumop))  
print("Sum by using Normal Fun={} ".format(res))  
print("-"*50)  
r=addop(a,b)  
print("Type of addop--Anonymous Function=",type(addop))  
print("Sum by using Anonymous Function={} ".format(r))
```

```
#Write a python program which will accept two numerical values and find the biggest and  
smallest amoung them by using anonymous functions  
#BigSmall.py  
big=lambda a,b: a if a>b else b # anonymous function  
small=lambda a,b: a if a<b else b # anonymous function
```

```
#mian program  
x=int(input("Enter First Value:"))  
y=int(input("Enter Second Value:"))  
bv=big(x,y)  
sv=small(x,y)  
print("Big({},{})={}".format(x,y,bv))  
print("Small({},{})={}".format(x,y,sv))
```

```
#Write a python program which will accept two numerical values and find the biggest and  
smallest amoung them by using anonymous functions  
#EqualBigSmall.py  
big=lambda a,b: a if a>b else b if b>a else "BOTH VALUES ARE EQUAL" # anonymous  
function  
small=lambda a,b: a if a<b else b if b<a else "BOTH VALUES ARE EQUAL" # anonymous  
function
```

```
#mian program  
x=int(input("Enter First Value:"))  
y=int(input("Enter Second Value:"))  
bv=big(x,y)  
sv=small(x,y)  
print("Big({},{})={}".format(x,y,bv))
```

```

print("Small({},{})={}".format(x,y,sv))
#WAPP which will accept list of numerical values and find max and min
#MaxMinValues.py
def readvalues():
    lst=[]
    n=int(input("Enter How Many Values:"))
    if(n<=0):
        return lst
    else:
        print("Enter {} Values:".format(n))
        for i in range(1,n+1):
            lst.append(int(input()))
    return lst

maxvalue=lambda lst:max(lst) # Anonymous Functions
minvalue=lambda lst:min(lst) # Anonymous Functions

#Main Program
lst=readvalues()
print("Content of lst=",lst)
if(len(lst)==0):
    print("Empty List and can't find max and min")
else:
    maxv=maxvalue(lst)
    minv=minvalue(lst)
    print("max({})={}".format(lst,maxv))
    print("minv({})={}".format(lst,minv))

```

List comprehension

=>The purpose of List comprehension is that to read the values dynamically from key board separated by a delimiter (space, comma, colon..etc) .

=>List comprehension is the most effective way for reading the data for list instead traditional reading the data and also we can perform Various Operations.

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

=>here expression represents either type casting or mathematical expression

Examples:

```

print("Enter List of values separated by space:") # [10,2,222,50,10,4,55,-3,0,22]
lst= [float(val) for val in input().split() ]
print("content of lst",lst)

```

Examples:

```

lst=[4,3,7,-2,6,3]
newlst=[ val*2 for val in lst ]
print("new list=",newlst) # [ 8, 6, 14,-4,12,6 ]
#ListComprehenEx1.py

```

```
lst=[10,20,30,40,50,60,70,80,90]
print('Original Content=',lst)
lst1= [ val+1 for val in lst ] # List Comprehension
print('Modified Content=',lst1)
print("-----")
set1={val*2 for val in lst } # set Comprehension
print(type(set1))
print('Modified Content=',set1)
tpl1=(val**0.2 for val in lst ) # Not Tuple Comprehension
tpl2=tuple(tpl1) # Convert generator object into tuple obj
print(type(tpl2))
print('Modified Content=',tpl2)


---


#ListComprehenEx2.py
lst=[10,20,30,40,50,60,70,80,90,101]
print('Original Content=',lst)
lst1=[ val for val in lst if (val%3==0)      ]
print("Multiples of 3=",lst1)
lst2=[ val for val in lst if (val%4==0)      ]
print("Multiple of 4=",lst2)


---


#ListComprehenEx3.py
lst=[-10,20,-30,40,50,-60,70,-80,90,101,0,12]
print('Original Content=',lst)
lst1=[ val for val in lst if (val>0)      ]
print("+Ve Values=",lst1)
lst2=[ val for val in lst if (val<0)      ]
print("-ve values=",lst2)


---


#ListComprehenEx4.py
print("Enter List of values separated by space:") # 10 20 30 40 50 50 60 70 80 90
lst=[int (val) for val in input().split() ]
print("Content of list=",lst)
print("-----")
print("Enter List of Names separated by Comma:")
lst1=[val for val in input().split(",") ]
print("List of Names=",lst1)


---


#write a pyhton program which will accept list of numerical value and obtain sqaure of each number
#ListComprehenEx5.py
print("Enter List of value separated by hash symbol:")
lst=[int(val) for val in input().split("#")] # 10#24#45#24#78#12
print("Content of list=",lst)
sqrlst=[val**2 for val in lst]
print("Square List=",sqrlst)


---


#ListComprehenEx6.py
print("Enter List of Names separated by Comma:")
lst1=[val for val in input().split(",") ]
print("List of Names=",lst1)
print("-----")
nameslist=[name for name in lst1 if (len(name)>=3) and (len(name)<=6)]
print("Names list whose length is between 3 and 6=",nameslist)
```

Special Functions in Python

=>In Python Programming, we have 3 Types of Special Functions. They are

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

1) filter()

=>The purpose of filter() is that " To filter out some elements from given list of elements based on some condition".

=>Syntax: varname=filter(functionName , IterableObject)

=>Here varname is an object of <class, 'filter'>. Programmatically we can convert an object of filter into Sequence , List, set and dict type by using Type Casting Functions.

=>filter() is one of the pre-defined special function.

=>Function Name can be either Normal Function or Anonymous Function.

=>Iterable Object can any Sequence , List, set and dict type

=>The Execution Process of filter() is that " Each Element of Iterable object passing to Specified Function Name, Function Name Takes that value, applied to the condition, If the condition is True then that Element filtered otherwise the element will be neglected". This Process will be repeated until all elements of Iterable object completed.

```
#Program for obtaining list of Possitive Values by using filter()
#FilterEx1.py
def positive(n):
    if(n>0):
        return True
    else:
        return False

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

#main program
lst=[10,-20,30,-40,50,-56,34,-24,0,-56]
x=filter(positive,lst)
print("Type of x=",type(x)) # <class, 'filter'>
print("content of x=",x) # content of x= <filter object at 0x000001E57B8263E0>
#Convert Filter object into list / set / tuple
listobj=list(x)
print("Given Data=",lst)
print("Possitive Elements=",listobj)
```

```
y=filter(negative,lst)
print("Type of y=",type(y)) # <class, 'filter'>
#Convert Filter object into list / set / tuple
tplobj=tuple(y)
print("Negative Elements=",tplobj)
```

```
#Program for obtaining list of Possitive Values by using filter()
#FilterEx2.py
poss=lambda k: k>0 # Anonymous Functions
negs=lambda k: k<0 # Anonymous Functions
```

```
#main program
print("Enter List of values separated by space:")
lst=[int(val) for val in input().split()]
pslist=list(filter(poss,lst))
nglist=tuple(filter(negs,lst))
print("Given data=",lst)
print("Positive Elements=",pslist)
print("Negative Elements=",nglist)
```

```
#Program for obtaining list of Possitive Values by using filter()
#FilterEx3.py
print("Enter List of values separated by space:")
lst=[int(val) for val in input().split()]
pslist=list(filter(lambda k: k>0 , lst ))
nglist=tuple(filter(lambda k: k<0,lst))
print("Given data=",lst)
print("Positive Elements=",pslist)
print("Negative Elements=",nglist)
```

```
#WAPP which will accept a line of text and count the number of vowels by filtering them
```

```
#FilterEx4.py
```

```
def vowels(ch):
    if ch.lower() in ['a','e','i','o','u']:
        return True
    else:
        return False
```

```
#main program
line=input("Enter Line of Text:")
print("\nGiven Line: {}".format(line)) # python is an oop lang
vlst=list(filter(vowels,line))
print("Vowles Found={}".format(vlst))
print("No. of Vowels={}".format(len(vlst)))
```

```
#FilterEx5.py
```

```
vowels=lambda ch: ch.upper() in ['A','E','I','O','U']
```

```
#main program
line=input("Enter Line of Text:")
print("\nGiven Line: {}".format(line)) # python is an oop lang
vlst=list(filter(vowels,line))
print("Vowles Found={}".format(vlst))
```

```

print("No. of Vowels={}".format(len(vlst)))


---


#FilterEx6.py
line=input("Enter Line of Text:")
print("\nGiven Line: {}".format(line)) # python is an oop lang
vlst=list(filter(lambda ch: ch.upper() in ['A','E','I','O','U'] , line))
print("Vowles Found={}".format(vlst))
print("No. of Vowels={}".format(len(vlst)))
print("=====OR=====")
vl=[ str(ch) for ch in line if ch.upper() in ['A','E','I','O','U'] ]
print("Vowles Found={}".format(vlst))
print("No. of Vowels={}".format(len(vlst)))


---


#FilterEx7.py
def consonants(ch):
    if (ch.isalpha() and ch.lower() not in ['a','e','i','o','u']):
        return True

#main program
line=input("Enter Line of Text:")
print("\nGiven Line: {}".format(line)) # python is an oop lang
cons=list(filter(consonants, line))
print("List of Cons=",cons)
print("Number of Cons=",len(cons))
vlst=list(filter(lambda ch: ch.upper() in ['A','E','I','O','U'] , line))
print("Vowles Found={}".format(vlst))
print("No. of Vowels={}".format(len(vlst)))

```

===== 2) map() =====

=>map() is used for obtaining new Iterable object from existing iterable object by applying old iterable elements to the function.

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

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

=>here 'varname' is an object of type <class, map> and we can convert into any iterable object by using type casting functions.

=>"functionName" represents either Normal function or anonymous functions.

=>"Iterable_object" represents Sequence, List, set and dict types.

=>The execution process of map() is that " map() sends every element of iterable object to the specified function, process it and returns the modified value (result) and new list of elements will be obtained". This process will be continued until all elements of Iterable_object completed.

```

#Program for map()
#MapEx1.py
def hike(sal):
    return (sal+sal*(20/100))

```

```
#main program
oldsallist=[10,15,12,20,18,8]
m=map(hike,oldsallist)
#print("type of m=",type(m))# <class, 'map'>
#print("content of m=",m) # <map object at 0x00000181D0666380>
#Type Cast map object into list
newsallist=list(m)
print("Old Salaries=",oldsallist)
print("New Salaries=",newsallist)
```

```
#Program for map()
#MapEx2.py
print("Enter List of Old Salaries separated by Comma:")
oldsal=[int(sal) for sal in input().split(",")]
newsal=list(map(lambda sal:sal*1.2,oldsal))
print("Old Salaries=",oldsal)
print("New Salaries:")
for val in newsal:
    print("{}".format(round(val,2)),end=" ")
print()
```

```
#Program for map()
#MapEx1.py
def hike(sal):
    return (sal+sal*(20/100))
```

```
#main program
oldsallist=[10,15,12,20,18,8]
m=map(hike,oldsallist)
#print("type of m=",type(m))# <class, 'map'>
#print("content of m=",m) # <map object at 0x00000181D0666380>
#Type Cast map object into list
newsallist=list(m)
print("Old Salaries=",oldsallist)
print("New Salaries=",newsallist)
```

```
#Program for map()
#MapEx2.py
print("Enter List of Old Salaries separated by Comma:")
oldsal=[int(sal) for sal in input().split(",")]
newsal=list(map(lambda sal:sal*1.2,oldsal))
print("Old Salaries=",oldsal)
print("New Salaries:")
for val in newsal:
    print("{}".format(round(val,2)),end=" ")
print()
```

```
#Program for addition of two list elements
#MapEx3.py
```

```
def addition(x,y):
    return (x+y)
```

```
#main program
print("Enter List of Values Separated By Space for List1:")
lst1=[int(val) for val in input().split()]
print("Enter List of Values Separated By Space for List2:")
lst2=[int(val) for val in input().split()]
lst3=list(map(addition,lst1,lst2))
print("Content of lst1=",lst1)
print("Content of lst2=",lst2)
print("Sum List=",lst3)
```

#Program for addition of two list elements

#MapEx4.py

```
print("Enter List of Values Separated By Space for List1:")
lst1=[int(val) for val in input().split()]
print("Enter List of Values Separated By Space for List2:")
lst2=[int(val) for val in input().split()]
lst3=list(map(lambda k,v:k+v , lst1 , lst2))
```

```
print("Content of lst1=",lst1)
print("Content of lst2=",lst2)
print("Sum List=",lst3)
```

reduce()

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

=>Syntax:-

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

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

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

Internal Flow of reduce()

step-1:- Initially, reduce() selects First Two values of Iterable object and place them in First var and Second var .

step-2:- The function-name(vlambda or normal function) utilizes the values of First var and Second var and applied to the specified logic and obtains the result.

Step-3:- reduce () places the result of function-name in First variable and reduce()
selects the succeeding element of Iterable object and places in second variable.

Step-4: Repeat Step-2 and Step-3 until all elements completed in
Iterable object and returns the result of First Variable.

#Program for finding sum of list of Values

#reduceeex1.py

import functools

lst=[5,10,6,3,14,8,2]

s=functools.reduce(lambda x,y:x+y,lst)

print("Sum({})={}".format(lst,s))

#Program for finding sum of list of Values

#reduceeex2.py

import functools

def sumop(x,y):

```

    return (x+y)



---


#Main Program,
print("List of Values Separated By Space:")
lst=[int(val) for val in input().split()]
s=functools.reduce(sumop,lst)
print("Sum({})={}".format(lst,s))


---


#write a python program which will accept list of value and find the max and min value
#reduceeex3.py
import functools
print("List of Values Separated By Space:")
lst=[int(val) for val in input().split()] # lst=23 45 78 -2 23 1 2
bv=functools.reduce(lambda k,v: k if k>v else v, lst)
sv=functools.reduce(lambda k,v: k if k<v else v, lst)
print("\nMax({})={}".format(lst,bv))
print("\nMin({})={}".format(lst,sv))


---


#FilterReduceEx.py
import functools

print("List of Values Separated By Space:")
lst=[int(val) for val in input().split()] # lst=[10,-20,34,56,-4,-5,6,12,-6]
psnums=list(filter(lambda x: x>0 , lst))
pslistsum=functools.reduce(lambda x,y:x+y,psnums)

nnnums=list(filter(lambda x: x<0 , lst))
nnlistsum=functools.reduce(lambda x,y:x+y,nnnums)

print("-"*50)
print("List of values :{}".format(lst))
print("\nList of Possitive Values :{}".format(psnums))
print("Sum of Possitive Values :{}".format(pslistsum))
print("\nList of Negative Values :{}".format(nnnums))
print("Sum of Negative Values :{}".format(nnlistsum))

print("-"*50)


---

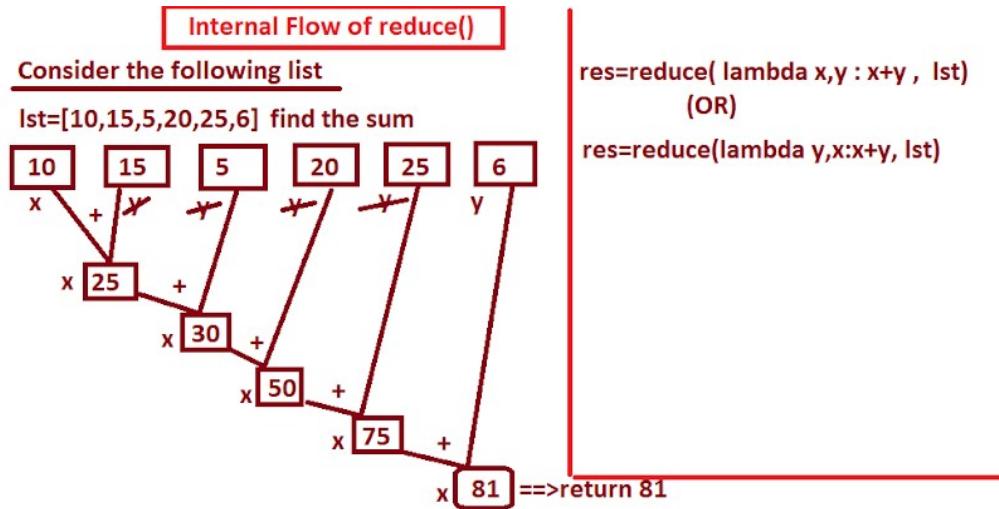

#FilterMapReduceEx.py
import functools
print("List of Salaries of Employees Separated By Space:")
#lst=[int(val) for val in input().split() if int(val)>=0 and int(val)<=1000 ] OR
lst=[int(val) for val in input().split() if 1000>=int(val)>=0 ]
print("-"*50)
print("\nList of Salaries:",lst)
print("-"*50)
#Filter for Those salaries which are in range of 0 to 500
sal_0_500=list(filter(lambda sal:500>=sal>=0,lst))
print("\nList of Slaries Ranges from 0-500={}".format(sal_0_500))
hike_sal_0_500=list(map(lambda sal:sal*1.1,sal_0_500))
print("List of Hiked Salaries Ranges from 0-500={}".format(hike_sal_0_500))

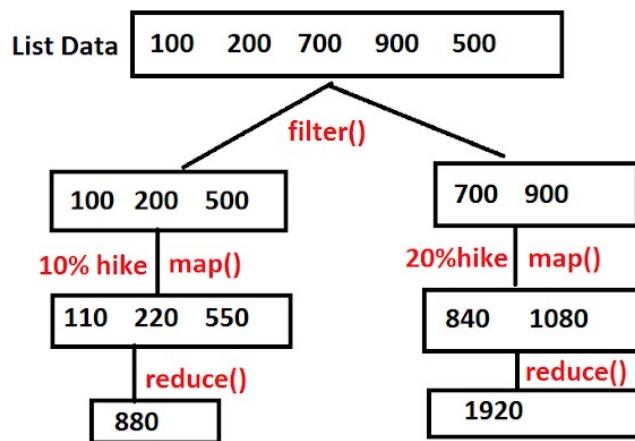
```

```

sum_hike_sal_0_500=functools.reduce(lambda x,y:x+y,hike_sal_0_500)
print("Total Salary paid by Company within the range 0-
500={}".format(sum_hike_sal_0_500))
print("-"*50)
#Filter for Those salaries which are in range of 501 to 1000
sal_501_1000=list(filter(lambda sal:1000>=sal>=501,lst))
print("\nList of Salaries Ranges from 501--1000={}".format(sal_501_1000))
hike_sal_501_1000=list(map(lambda sal:sal*1.2,sal_501_1000))
print("List of Hiked Salaries Ranges from 501--1000={}".format(hike_sal_501_1000))
sum_hike_sal_501_1000=functools.reduce(lambda x,y:x+y,hike_sal_501_1000)
print("Total Salary paid by Company within the range 501-
1000={}".format(sum_hike_sal_501_1000))
print("-"*50)

```





Modules in Python

Index

- =>Purpose of Modules
 - =>Definition of Module
 - =>Types of Modules
 - =>Number of Approaches to Re-use the Modules
 - a) By Using import statement
 - b) By Using from ... import statement
 - =>Programming Examples
 - =>Re-Loading the Modules
 - =>Programming Examples
-

Modules in Python

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

=>Definition of Module:

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

=>Types of Modules:

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

- 1) Pre-defined (or) Built-in Modules
 - 2) Programmer or user or custom-defined modules.
-

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

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

Examples: math cmath functools sys calendar os
re threading pickle random.....etc

=>Out of many pre-defined modules, in python programming one implicit pre-defined module imported to every python program called "builtins".

2) Programmer or user or custom-defined modules:

=>These modules are developed by Python Programmers and they are available in Python Project and they are used by other python programmers who are in project development to deal with common requirements.

=>Examples:- aop mathsinfo icicietc

Development of Programmer-Defined Module

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

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

=>After developing step-1, step-2 and step-3 , we must save on some file name with an extension .py (FileName.py) and it is treated as module name.

=>When a file name treated as a module name , internally Python execution environment creates a folder automatically on the name of __pycache__ and it contains module name on the name of "filename.cpython-310.pyc ".

Examples:

____pycache____ <----Folder Name

aop.cpython-310.pyc <---Module Name
mathsinfo.cpython-310.pyc<--Module Name
icici.cpython-310.pyc<-----Module Name

Number of approaches to re-use Modules

=>We know that A Module is a collection of variables, Functions and Classes.

=>To re-use the features(Variable Names, Function Names and Class Names) of module, we have 2 approaches.They are

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

1) By using import statement:

=>'import' is a keyword

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

=>we can use import statement in 4 ways.

=>Syntax-1: import module name

=>This syntax imports single module

Example: import icici
 import aop
 import mathsinfo

=>Syntax-2: import module name1, module name2....Module name-n

=>This syntax imports multiple modules

Example: import icici , aop, mathsinfo

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

=>This syntax imports single module and aliased with another name

Example: import icici as i
import aop as a
import mathsinfo as m

=>Syntax-4: import module name1 as alias name, module name2 as alias

=>This syntax imports multiple modules and aliased with another names

Example: import icici as i, aop as a , mathsinfo as m

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

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

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

=>Here "from" "import" are the key words

=>The purpose of from.... import statement is that " To refer or access the variable names, function names and class names in current program directly without writing module name as alias name of Module name."

=> we can use from.... import statement in 3 ways.

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

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

Example: from calendar import month
from aop import addop,subop
from mathinfo import pi,e
from icici import bname,loc, calsimpleint

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

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

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

Syntax-3: from module name import *

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

=>This syntax is not recommended to use bcoz it imports required Features of Module and also import un-interested features also imported and leads more main memory space.

Example: from calendar import *
from aop import *
from mathsinfo import *

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

Variable Name
Function Name
Class Name

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

=====X=====

#Program for demonstrating the need of modules
#MathsInfo.py--File Name & acts module name
PI=3.14
E=2.71 # here PI and E are called Global Variables

#Program for demonstrating the need of Modules
#aop.py---file name & acts as Module name
def addop(a,b):
 print("sum({},{})={}".format(a,b,a+b))
def subop(a,b):
 print("sub({},{})={}".format(a,b,a-b))
def mulop(a,b):
 print("mul({},{})={}".format(a,b,a*b))

#icici.py----File Name & acts module name
bname="ICICI"

```
addr="HYD" # Here bname and addr are called Global variables
def calsimpleint():
    p=float(input("\nEnter Principle Amount:"))
    t=float(input("Enter Time:"))
    r=float(input("Enter Rate of Interest:"))
    #cal si
    si=(p*t*r)/100
    print("Simple Interest={}".format(si))


---


#Programmer1.py
print("Val of PI=",PI)
print("Val of E=",E)


---


#Programmer2.py
addop(10,20) # Function Call
mulop(4,5) # Function Call


---


#Programmer3.py
import MathsInfo
import aop
print("Val of PI=",MathsInfo.PI)
print("Val of E=",MathsInfo.E)
aop.addop(10,20)


---


#Programmer4.py
import icici
print("Bank Name:",icici.bname)
print("Bank Address:",icici.addr)
icici.calsimpleint()


---


#Progammer5.py
from calendar import *
print(month(2022,12))
print(calendar(2023))


---


#ImportStmtsSyntax-1.py---
import aop
import icici
aop.addop(100,200)
icici.calsimpleint()


---


#ImportStmtsSyntax-2.py
import aop,icici
aop.addop(100,200)
icici.calsimpleint()


---


#ImportStmtsSyntax-3.py
import aop as a
import icici as i
a.addop(100,200)
i.calsimpleint()


---


#ImportStmtsSyntax-4.py
import aop as a , icici as i
a.addop(10,20)
i.calsimpleint()


---


#FromImportStmtsSyntax-1.py
from aop import mulop
```

```

from icici import bname,calsimpleint
mulop(100,200)
print("Bank Name:",bname)
calsimpleint()


---


#FromImportStmtsSyntax-2.py
from aop import mulop as m
from icici import bname as bn,calsimpleint as cs
m(100,200)
print("Bank Name:",bn)
cs()


---


#FromImportStmtsSyntax-3.py
from aop import *
from icici import *
mulop(100,200)
print("Bank Name:",bname)
calsimpleint()

```

=====

realoding a modules in Python

=====

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

(OR)

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

=>Purpose / Situation:

=>`reload()` reloads a previously imported module.

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

X

#shares.py---file and treated as module name

```

def sharesinfo():
    d={"Tech":19,"Pharma":11,"Auto":1,"Finance":00}
    return d

```

#main program

#sharesdemo.py

import shares

import time

import importlib

def disp(d):

print("-"*50)

print("\tShare Name\tValue")

print("-"*50)

for sn,sv in d.items():

print("\t{}\t:{}".format(sn,sv))

else:

```

        print("-"*50)
#main program
d=shares.sharesinfo()
disp(d)
time.sleep(15)
importlib.reload(shares) # relodaing previously imported module
d=shares.sharesinfo() # obtaining changed / new values of previously imported module
disp(d)


---


#Shares.py--File Name and Module Name
def sharesinfo():
    d1={"IT":1000,"Fin":1001,"Auto":1111,"Pharma":7}
    return d1


---


#SharesDemo.py
import Shares,time
import importlib
def dispdata(d):
    print("-"*50)
    print("\tShare Name\tShare Value:")
    print("-"*50)
    for sn,sv in d.items():
        print("\t{}\t{}".format(sn,sv))
    print("-"*50)

#main program
dictobj= Shares.sharesinfo()
dispdata(dictobj)
print("I am from going to sleep for 15 secs--first time")
time.sleep(15)
print("I am coming out of sleep ")
importlib.reload(Shares) #-----imp.reload(Shares)
dictobj= Shares.sharesinfo()
dispdata(dictobj)
print("I am from going to sleep for 15 secs --second time")
time.sleep(15)
print("I am coming out of sleep ")
importlib.reload(Shares)
dictobj= Shares.sharesinfo()
dispdata(dictobj)


---


=====
```

Packages in Python

Index

-
- =>Purpose of Package
 - =>Definition of Package
 - =>Development of Package
 - =>Number of Approaches to re-use Packages
 - a) By Using sys.path.append()
 - b) By using PYTHONPATH Environmental Variable
 - =>Programming Examples
-

Package in Python

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

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

=>The Package Concept is a collection of Modules.

=>The purpose of Packages is that to provide code re-usability across the folders / drives / environments.

=>To deal with the package, we need to learn the following.

- a) create a package
 - b) re-use the package
-

a) create a package:

=>To create a package, we use the following steps.

- i) create a Folder
- ii) place / write an empty python file called `__init__.py`
- iii) place / write the module(s) in the folder where it is considered as

Package

Name

Example:

bank <---- Package Name

 __init__.py <---- Empty Python File
 simpleint.py <--- Module Name
 aop.py----Module Name
 icici1.py---Module Name
 welcome.py <--- Module Name

b) re-use the package

=>To reuse the modules of the packages across the folders / drives / environments, we have to two approaches. They are

- i) By using sys module
 - ii) by using PYTHONPATH Environmental Variable Name
-

i) By using sys module:

Syntax:

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

=>sys is pre-defined module

=>path is a pre-defined object / variable present in sys module

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

Example:

```
sys.path.append("E:\\KVR-PYTHON-6PM\\ACKAGES\\BANK")
               (or)
sys.path.append("E:\\KVR-PYTHON-6PM\\ACKAGES\\BANK")
               (or)
sys.path.append("E:\\KVR-PYTHON-6PM\\ACKAGES\\BANK")
```

ii) by using PYTHONPATH Environmental Variables:

=>PYTHONPATH is one of the Environmental Variable

=>Search for Environmental Variable

Steps for setting :

Var name : PYTHONPATH

Var Value : E:\\KVR-PYTHON-7AM\\PACKAGES\\BANK

The overall path

PYTHONPATH= E:\\KVR-PYTHON-

11AM\\PACKAGES\\BANK

#kvrmath.py---File name and Module name

def kvsqrt(n):

 print("sqrt({})={}".format(n,n**0.5))

#MathCal.py---File Name and Module Name

def fact(n):

 if(n<0):

 print("{} Not possible to calculate Factorial".format(n))

 else:

 f=1

 for i in range(1,n+1):

 f=f*i

 else:

 print("fact({})={}".format(n,f))

#LetterCount.py----Module Name

def LetterCount():

 line=input("Enter a Line of text:")#aaabbzzzeeee

 d={} # Empty dict

 for ch in line:

 if ch in d:

 d[ch]=d[ch]+1 # OR d[ch]=d.get(ch)+1

 else:

 d[ch]=1

 else:

 print("-"*50)

 for let,noc in d.items():

 print("\t{}-->{}".format(let,noc))

 print("-" * 50)

```
#d={'a':3,'b':2,'z':4,'e':4}
```

Exception Handling

Index

=>Purpose of Exception Handling

=>Types of Errors

- i) Compile Time Errors
- ii) Logical Errors
- iii) Runtime Errors

=>Definition of Exception

=>Definition of Exception handling

=>Types of Exceptions.

- a) Pre-defined Exceptions
- b) Programmer-Defined Exceptions

=>Handling Exceptions

=>Keywords for Handling Exceptions

- 1. try
- 2. except
- 3. else
- 4. finally
- 5. raise

=>Syntax for Handling Exceptions

=>Programming Examples

=>Development of Programmer-Defined Exceptions

=>Programming Examples

=>ATM Case Study

Building Blocks of Exception Handling

1. When the application user or end user enters Invalid Input then we get Run time Errors.

(Invalid Input---->Run time Error)

2. By default Runtime Errors always generates Technical Error Messages.

3. Definition of Exception: Every Runtime Error is called Exception

(Invalid Input---->Runtime Error----->Exception)

Hence Every Valid Input gives Result.

Every Invalid Input gives Exception.

4. By default All exceptions gives Technical Error Messages.

5. Definition of Exception Handling:

=>The Process of converting Technical Error Messages into User-Friendly Error Messages is called Exception Handling

6. When Exception Occurs , Internally PVM perform 3 steps
 - a) PVM Terminates the program execution Abnormally.
 - b) PVM comes out of Program flow
 - c) PVM generates Technical Error Message by default.
 7. To Perform Step-(a),Step-(b) and Step-(c) by PVM, PVM creates an object of appropriate exception class .
 8. When an exception occurs, PVM creates an object of appropriate exception class. Hence Every Invalid Input gives Exception and Every Exception considered as object of appropriate exception class
 9. Hence Every Exception is python is Object of appropriate exception class.
-

Types of Errors in Exception Handling

=>The Purpose of Exception Handling is that " To Build Robust (Strong) Applications ".

=>In Real Time, To develop any project, we must choose one programming language and by using that language, we develop, compile and execute the programs. During this process we get 3 types of Errors. They are

1. Compile Time Errors
 2. Logical Errors
 3. Runtime Errors
-

1. Compile Time Errors

=>Compile Time Errors are those which are occurring during Compilation Process.

=>Compile Time Errors occurs due to Syntaxes are not followed.

=>Compile Time Errors must be solved by Programmers at development time.

2. Logical Errors

=>Logical Errors are those which are occurring during Run time or Execution Process.

=>Logical Errors occurs due to Wrong Representation of Logic.

=>Logical Errors gives Wrong Results or Inconsistent Results.

=>Logical Errors must be solved by Programmers at development time.

3. Runtime Errors (Implementation Errors)

=>Runtime Errors are those which are occurring during Run time or Execution Process.

=>Runtime Errors occurs due to Wrong or Invalid Input Entered by End User or Application

User.

=>By default Runtime Error gives Technical Error Messages and these messages Understandable by Programmers but not by End Users. So Industry is recommended to generate User-Friendly Error Messages by using Exception Handling.

Handling the exception in Python

=>Handling the exception in Python is nothing but Converting Technical Error Messages into User-Friendly Error Messages.

=>To do this conversion, In Python Programming we have 5 keywords. They are

1. try
 2. except
 3. else
 4. finally
 5. raise
-

Syntax for handling the exceptions

```
try:  
    Block of statements  
    Generating Exceptions  
except <exception-class-name-1>:  
    Block of statements generates  
    User-Friendly Error Messages  
except <exception-class-name-2>:  
    Block of statements generates  
    User-Friendly Error Messages  
  
-----  
except <exception-class-name-n>:  
    Block of statements generates  
    User-Friendly Error Messages  
else:  
    Block of statements generates  
    Results  
finally:  
    Block of statements executes  
    compulsorily
```

Detailed Explanation of all Block---tomorrow
try block:

except

else

finally

Types of Exceptions in Python

=>In Python Programming, we have 2 types of exceptions. They are

1. Pre-Defined or Built-in Exceptions
 2. Programmer / User / Custom-Defined Exceptions
-

1. Pre-Defined or Built-in Exceptions

=>These exceptions are defined and developed by Python Developers and they are available as part of Python Software and used by python Programmers for dealing Universal Problems.

=>Some of the Universal Problems are

- a) Division by Zero (ZeroDivisionError)
 - b) Invalid number format (ValueError)
 - c) Invalid Indices (IndexError)
 - d) Wrong module name (ModuleNotFoundError).....etc
-

2. Programmer / User / Custom-Defined Exceptions

=>These exceptions are defined and developed by Python Programmers and they are available as part of Python Project and used by other python Programmers who are in the project for dealing with Common Problems.

=>Some of the Common Problems are

- a) Attempting to enter Invalid PIN
 - b) Attemmpting to enter Wrong User Name and Password
 - c) Attempting to withdraw more amount than existing
-etc
-

#Program for accepting two integer values and find their div

```
#Div1.py
print("Program Execution Strated:")
s1=input("Enter First Value:")
s2=input("Enter Second Value:")
a=int(s1) #-----ValueError
b=int(s2) #-----ValueError
print("First Value:{} ".format(a))
print("Second Value:{} ".format(b))
c=a/b #-----ZeroDivisionError
print("Div:{} ".format(c))
print("Program Execution Ended:")
```

#Program for accepting two integer values and find their div

```
#Div2.py
try:
    print("Program Execution Strated:")
```

```

s1=input("Enter First Value:")
s2=input("Enter Second Value:")
a=int(s1) #-----ValueError
b=int(s2) #-----ValueError
c=a/b #-----ZeroDivisionError
except ZeroDivisionError:
    print("\nDON'T ENTER ZERO FOR DEN...")
except ValueError:
    print("\nDON'T ENTER STRS, ALNUMS AND SYMBOLS")
else:
    print("First Value:{}\n".format(a))
    print("Second Value:{}\n".format(b))
    print("Div:{}\n".format(c))
    print("Program Execution Ended:")
finally:
    print("\ni am from finally block:")

```

Explanation for the keywords in handling exceptions

1) try:

=>It is the block in which we write block statements generates exceptions. In otherwords, whatever the

statements are generating exceptions, those statements must written within try block and 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

=>After executing appropriate except block, PVM never goes to try block to execute rest of the

statements.

=>Every try block must be immediately followed by except block (otherwise we get SyntaxError)

=>Every try block must contain atleast one except block and recommended to define / write multiple

except blocks for generating multiple user-friendly error messages.

2)except

1) It is the block in which we write block of statements gives User-Friendly Error Messages. In

Otherwords, except block supresses the Technical error Messages and Generates User-Freindly Errors Mesasges and this block is called Exception Processing Block.

NOTE:- Handling Exception= try block + except block

2) except block will execute when there is an exception in try block.

3) Even we write multiple except block, PVM execute appropriate except block depends on type of

exception occurs in try block.

- 4) The place of writing except block is after try block and before else block(if present)
-

3. else

- 1) It is the block, In which we define Block of statements generates results and this block is known as

Result generated Block

- 2) else block will execute where is no exception in try block.

- 3) Writing else block is Optional

- 4) The place of writing else is after except block and before finally (if present)
-

4. finally

1. It is the Block, In which we block of statements used for Relinquishing (Close / release / clean-up/

give-up) the Resources (Files, Databases) which are obtained in try block.

2. Writing / Defining finally Block is Optional

3. finally block will execute compulsorily (if we write)

4. The place of writing finally is after else block (if it present)
-
-
-
-

Various Forms of except blocks

=>except block contains Variouis Forms. They are

Form-1: This Syntax handle one exception at a time and generates user-friendly Error message one

at a time.

Syntax: try:

 except <exception-class-name>:

Examples: Refer Div2.py Program

Form-2:The feature is called "Multi Exception Handling Block"

Here with Single except block, we can handle multiple specific exceptions and generates multiple user-friendly error messges.

try:

```
except (exception-class-1, exception-class-2, ... exception-class-n):
```

Multiple User-friendly Error Messages

Examples: Refer Div3.py Program

Form-3: Handling the Single Specific Exception with alias name
here alias name of exception can capture the Tech error of corresponding exception.

Syntax: try:

```
    except <exception-class-name-1> as alias name:
```

```
    except <exception-class-name-2> as alias name:
```

Examples: Refer Div4.py Program

Form-4 : default except block

=> This default block will execute when no exception matches with exception exceptions.

Syntax: try:

```
    except :
```

Examples: Refer Div5.py Program

Standard Syntax-1:

```
try:  
    Block of statements  
    Generating Exceptions  
except <exception-class-name-1>:  
    Block of statements generates  
    User-Friendly Error Messages  
except <exception-class-name-2>:  
    Block of statements generates  
    User-Friendly Error Messages
```

```

-----
except <exception-class-name-n>:
    Block of statements generates
    User-Friendly Error Messages
except : #default Exception block
    default error messages
else:
    Block of statements generates
    Results
finally:
    Block of statements executes
    compulsorily

```

(OR)

try:

```

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

except (exception-class-1,exception-class-2,...exception-class-n):

Multiple User-friendly Error Messages

```

except : #default Exception block
    default error messages
else:
    Block of statements generates
    Results
finally:
    Block of statements executes
    compulsorily

```

Examples: Refer Div6.py Program

```

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

#Program for accepting two integer values and find their div
#Div2.py
try:
    print("Program Execution Started:")
    s1=input("Enter First Value:")
    s2=input("Enter Second Value:")
    a=int(s1) #-----ValueError
    b=int(s2) #-----ValueError
    c=a/b #-----ZeroDivisionError
except ZeroDivisionError:
    print("\nDON'T ENTER ZERO FOR DEN...")
except ValueError:
    print("\nDON'T ENTER STRS, ALNUMS AND SYMBOLS")
```

```
else:  
    print("First Value:{}".format(a))  
    print("Second Value:{}".format(b))  
    print("Div:{}".format(c))  
    print("Program Execution Ended:")  
finally:  
    print("\ni am from finally block:")
```

#Program for accepting two integer values and find their div

#Div3.py

```
try:  
    print("Program Execution Strated:")  
    s1=input("Enter First Value:")  
    s2=input("Enter Second Value:")  
    a=int(s1) #-----ValueError  
    b=int(s2) #-----ValueError  
    c=a/b #-----ZeroDivisionError  
except (ZeroDivisionError,ValueError): # Multi Exception Handling Block  
    print("\nDON'T ENTER ZERO FOR DEN...")  
    print("DON'T ENTER STRS, ALNUMS AND SYMBOLS")  
else:  
    print("First Value:{}".format(a))  
    print("Second Value:{}".format(b))  
    print("Div:{}".format(c))  
    print("Program Execution Ended:")  
finally:  
    print("\ni am from finally block:")
```

#Program for accepting two integer values and find their div

#Div4.py

```
try:  
    print("Program Execution Strated:")  
    s1=input("Enter First Value:")  
    s2=input("Enter Second Value:")  
    a=int(s1) #-----ValueError  
    b=int(s2) #-----ValueError  
    c=a/b #-----ZeroDivisionError  
except ZeroDivisionError as z:  
    print(z) # division by zero  
except ValueError as k:  
    print(k) # invalid literal for int() with base 10: '10abc'  
else:  
    print("First Value:{}".format(a))  
    print("Second Value:{}".format(b))  
    print("Div:{}".format(c))  
    print("Program Execution Ended:")  
finally:  
    print("\ni am from finally block:")
```

#Program for accepting two integer values and find their div

#Div5.py---(Not recommended in real time)

```
try:  
    print("Program Execution Strated:")
```

```

s1=input("Enter First Value:")
s2=input("Enter Second Value:")
a=int(s1) #-----ValueError
b=int(s2) #-----ValueError
c=a/b #-----ZeroDivisionError
except:
    print("Oooops., some thing went wrong!!!")
else:
    print("First Value:{}\n".format(a))
    print("Second Value:{}\n".format(b))
    print("Div:{}\n".format(c))
    print("Program Execution Ended:")
finally:
    print("\ni am from finally block:")


---


#Program for accepting two integer values and find their div
#Div6.py---Kvr Programmer--defined code in 2022 Oct
try:
    print("Program Execution Strated:")
    s1=input("Enter First Value:")
    s2=input("Enter Second Value:")
    a=int(s1) #-----ValueError
    b=int(s2) #-----ValueError
    c=a/b #-----ZeroDivisionError
    #Code in 2023 adding Sandeep
    s="PYTHON"
    print(s[10])
except ZeroDivisionError :
    print("\nDON'T ENTER ZERO FOR DEN...")
except ValueError :
    print("\nDON'T ENTER STRS, ALNUMS AND SYMBOLS")
except IndexError:
    print("Index wrong plz check:")
except: # Default except block ---must be written at last
    print("Oooops , some thing went wrong!!!")
else:
    print("First Value:{}\n".format(a))
    print("Second Value:{}\n".format(b))
    print("Div:{}\n".format(c))
    print("Program Execution Ended:")
finally:
    print("\ni am from finally block:")


---


=====
```

Development of Programmer-Defined Exceptions

=>These exceptions are defined and developed by Python Programmers and they are available as part of Python Project and used by other python Programmers who are in the project for dealing with Common Problems.

=>Some of the Common Problems are

- a) Attempting to enter Invalid PIN
- b) Attemmpting to enter Wrong User Name and Password

c) Attempting to withdraw more amount than existing
.....etc

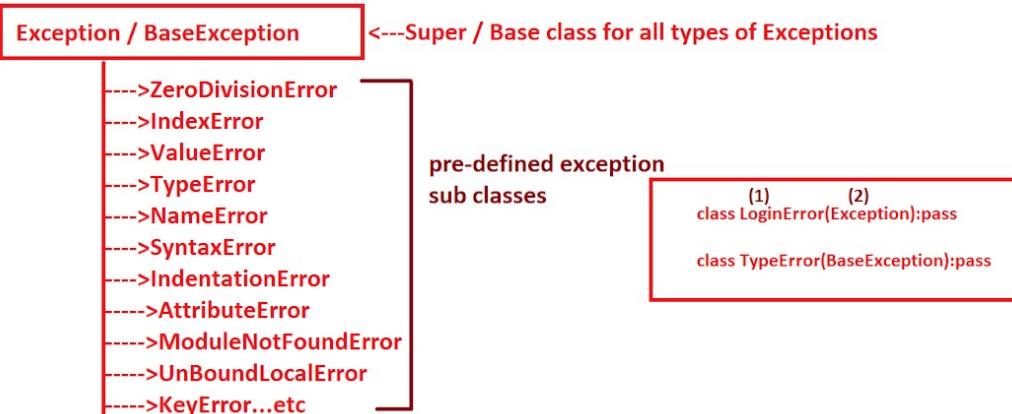
=>Steps for Developing Programming Exceptions.

- Step-1: Choose the Programmer-defined class name
Step-2: The Programmer-defined class name must Inherit from Base class exception called "Exception" or BaseException"
Step-3: The above development must be saved on some file name with an extension.py (Module Name)
-

Example: class LoginError(Exception):pass
 class InSuffBal(BaseException):pass

=>Here LoginError and InSuffBal are comes under Programmer-defined Exception sub classes.

Exception handling Hierarchy Chart



=====
raise key word
=====

=>raise keyword is used for hitting / raising / generating the exception provided some condition must be satisfied.

=>raise keyword always used inside of Function Definition only.

=>PVM uses raise keyword implicitly for hitting pre-defined Exception where as Programmer makes the PVM to use raise keyword explicitly for Hitting or Generating Programmer-defined Exceptions.

=>Syntax:- if (Test Cond):
 raise <exception-class-name>

=>Syntax:- def functionname(list of formal parms if any):

if (Test Cond):
 raise <exception-class-name>

Examples:

```
from kvr import KvrDivisionError  
def division(a,b):  
    if(b==0):  
        raise KvrDivisionError  
    else:  
        return (a/b)
```

#kvr.py--File Name and treated as module name---Step-3

step-1 Step-2

```
class KvrDivisionError(Exception):pass # Phase-I--development of exception
```

#division.py--File Name and Module name

```
from kvr import KvrDivisionError  
def division(a,b): # Common Function ---Phase-II  
    if(b==0):  
        raise KvrDivisionError # Hitting or generating the exception  
    else:  
        return (a/b)
```

#Main Program

#DivisionDemo.py

```
from division import division
```

```
from kvr import KvrDivisionError
```

```
try:
```

```
    a=int(input("Enter Value of a:"))  
    b=int(input("Enter Value of b:"))  
    res=division(a,b)#Function Call
```

```
except KvrDivisionError: # Phase-3--Handling the exception
```

```
    print("\nDon't Enter Zero for Den...")
```

```
except ValueError:
```

```
    print("\nDon't enter strs, alnums and symbols")
```

```
else:
```

```
    print("Division=",res)
```

#MulTabExcept.py--Phase-I

```
class NegativeNumberError(Exception):pass
```

```
class ZeroError(BaseException):pass
```

#MulTable.py--File Name and Module Name--Phase-II

```
from MulTabExcept import NegativeNumberError,ZeroError
```

```
def table(n):
```

```
    if(n<0): # Hit the exception--NegativeNumberError
```

```
        raise NegativeNumberError
```

```
    elif(n==0): # Hit the exception --ZeroError
```

```

        raise ZeroError
    elif(n>0):
        print("*50")
        print("Mul Table for :{}".format(n))
        print("= * 50")
        for i in range(1,11):
            print("\t{}x{}={}".format(n,i,n*i))
    else:
        print("= * 50")


---


#MulTableDemo.py---main program--Phase-III
from MulTable import table
from MulTabExcept import NegativeNumberError,ZeroError
n=int(input("Enter a Number:"))
try:
    table(n) #
except NegativeNumberError:
    print("Don't Enter -Ve Number")
except ZeroError:
    print("Don't enter Zero for Mul table")
except:
    print("Some thing went wrong!!!!")
finally:
    print("I am from finally Block")


---



```

ATM CASE STUDY

```

#AtmExcept.py--File Name and Module name
class DepositError(Exception):pass
class WithdrawError(BaseException):pass
class InSuffFundError(Exception):pass


---


#AtmMenu.py--File Name and module name
def menu():
    print("-*50")
    print("\tATM OPerations")
    print("-*50")
    print("\t1.Deposit")
    print("\t2.Withdraw")
    print("\t3.Bal Enq")
    print("\t4.Exit")
    print("-*50")


---


#AtmOperations.py--File and module
from AtmExcept import DepositError,WithdrawError,InSuffFundError
bal=500.00 # Global variables
def deposit():
    damt=float(input("Enter how much amount u want to deposit:"))#implcitly raises
ValueError
    if(damt<=0): # Exception Involved
        raise DepositError # Hitting or raising exception explicitly
    if(damt>0):
        global bal

```

```

        bal=bal+damt
        print("Ur Account xxxxxxxx123 credited with INR:{}.".format(damt))
        print("Now Account xxxxxxxx123 Balanace after
depositINR:{}.".format(bal))

def withdraw():
    global bal
    wamt=float(input("Enter how much amount u want to Withdraw:"))#implcitly
raises ValueError
    if(wamt<=0):
        raise WithdrawError # Hitting or raising exception explcitly
    elif((wamt+500)>bal):
        raise InSuffFundError # Hitting or raising exception explcitly
    else:
        bal=bal-wamt
        print("Ur Account xxxxxxxx123 debited with INR:{}.".format(wamt))
        print("Now Account xxxxxxxx123 Balanace after withdraw
INR:{}.".format(bal))

def balenq():
    print("Ur Account xxxxxxxx123 BalanceINR:{}.".format(bal))


---


#AtmDemo.py----File Name and Main Program
import sys
from AtmMenu import menu
from AtmOperations import deposit,withdraw,balenq
from AtmExcept import DepositError,WithdrawError,InSuffFundError
while(True):
    menu()
    try:
        ch=int(input("Enter Ur Choice:"))
        match(ch):
            case 1:
                try:
                    deposit()
                except ValueError:
                    print("Don't enter strs, alnums and symbols as
deposit amt:")
            case 2:
                try:
                    withdraw()
                except ValueError:
                    print("Don't enter strs, alnums and symbols as
withdraw amt:")
            except DepositError:
                print("Don't Deposit -Ve and Zero Value:")
            except WithdrawError:
                print("Don't withdraw -Ve and Zero Value:")
            except InSuffFundError:
                print("Ur Account does not have suff Funds--")

```

Read Python Notes")

```

        case 3:
            balenq()
        case 4:
            print("Thx for using this program")
            sys.exit()
        case _:
            print("Ur Selection of Operation is Wrong-try again")
    except ValueError:
        print("Don't enter strs, alnums and symbols for choice of Operations:")

```

MAX STRING PALINDROMES

```

#Maxpalindromeex1.py
lst=[ 1, 232, 5545455, 999999, 1212 , 8558 ]
ml=0
cl=0
pos=-1
for i in range(0,len(lst)):
    cv=str(lst[i])
    if(cv==cv[::-1]):
        cl=len(cv)
    if(cl>ml):
        ml=cl
        pos=i
else:
    print("Max palindrome Value={}".format(lst[pos]))

```

```

#Maxpalindromeex2.py
lst=[ "mom","madam","liril","malayalam","racecar"]
ml=0
cl=0
pos=-1
for i in range(0,len(lst)):
    cv=str(lst[i])
    if(cv==cv[::-1]):
        cl=len(cv)
    if(cl>ml):
        ml=cl
        pos=i
else:
    print("Max palindrome Value={}".format(lst[pos]))

```

```

#Maxpalindromeex3.py
lst=[ 1, 232, 5545455, 999999, 1212 , 8558 ]
lst=[ "mom","car","madam","liril","mala","racecar"]
ltp=[(val,len(val)) for val in lst if val==val[::-1] ]
print(ltp) # [('mom', 3), ('madam', 5), ('liril', 5), ('malayalam', 9), ('racecar', 7)]
d=dict(ltp)
val=max(d.values()) # val=9
maxpalin=[ k for k,v in d.items() if v==val]
print("\nMax Length palindrome=",maxpalin)

```

#Maxpalindromeex4.py

```
lst=[ 1, 232, 5545455, 999999, 1212 , 8558 ]
ltp=[(str(val),len(str(val))) for val in lst if str(val)==str(val)[::-1] ]
print(ltp) #
d=dict(ltp)
val=max(d.values())
maxpalin=[ k for k,v in d.items() if v==val]
print("\nMax Lengh palindrome=",maxpalin[0])
```

Files in Python

Index

=>Purpose of Files

=>Types of Applications

- a) Non-Persistent Applications
- b) Persistent Applications

=>Definition of File, Stream

=>Operations on Files

- a) Write Operation
- b) Read Operations

=>File Opening Modes

- 1) r 2) w 3) a
- 4) r+ 5) w+ 6) a+
- 7) x 8) x+

=>Number of Approaches to Open the File

- a) By Using open()
- b) By using " with open() as "

=>Writing the Data to The file

- 1) write()
- 2) writelines()

=>Reading the data from the files

- 1) read()
- 2) readlines()

=>Programming Examples

=>Pickling(Object Serialization) and Un-pickling (Object De-Serialization)

=>pickle module

=>Implementation of pickling and un-pickling

=>Programming Examples

=>Working CSV Files (Excel Sheet of MS)

=>csv module

=>Operations on CSV Files

- a) csv.reader
- b) csv.writer
- c) csv.DictReader
- d) csv.DictWriter

=>Programming Examples

=>OS Modules

=>OS Operations

=>Programming Examples

Types of Application in Files

=>The purpose of Files in any programming language is that " To maintain Data Persistency".

=>The Process of storing the data permanently is called Data Persistency.

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

- 1) Non-Persistent Applications
- 2) Persistent Applications

=>In Non-Persistent Applications development, we read the data from Keyboard , stored in main memory(RAM) in the form objects, processed and whose results displayed on Monitor. Examples: ALL our previous examples comes under Non-Persistent Applications.

=>We know that Data stored in Main Memory is temporary.

=>In Persistent Applications development, we read the data from Keyboard , stored in main memory(RAM) in the form objects, processed and whose results stored Permanently.

=>In Industry, we have two ways to store the Data Permanently. They are

- 1) By using Files
- 2) By Using DataBase Softwares (Oracle, MySQL, MongoDB, DB2, PostgreSQL, SQL Server, SQLite3..etc)

X

Data Persistence by Files of Python

Def. of File:

=>A File is a collection of Records.

=>Files Resides in Secondary Memory(HDD).

=>Technically, File Name is a named location in Secondary Memory.

=>All the objects data of main memory becomes records in File of Secondary memory and records of file of secondary memory becomes the objects in main memory.

Def. of Stream:

=>The Flow of Data between object(s) of Main Memory and Files of Secondary memory is called

Stream.

Operations on Files

=>On the files, we can perform Two Types of Operations. They are

- 1) Write Operation.
- 2) Read Operation.

1) Write Operation:

=>The purpose of write operation is that " To transfer or save the object data of main memory as record in the file of secondary memory".

=>Steps:

- 1) Choose the File Name
- 2) Open the File Name in Write Mode
- 3) Perform cycle of Write Operations.

=>While we are performing write operations, we get the following exceptions.

- a) IOError
- b) OSError
- c) FileExistError

2) Read Operation:

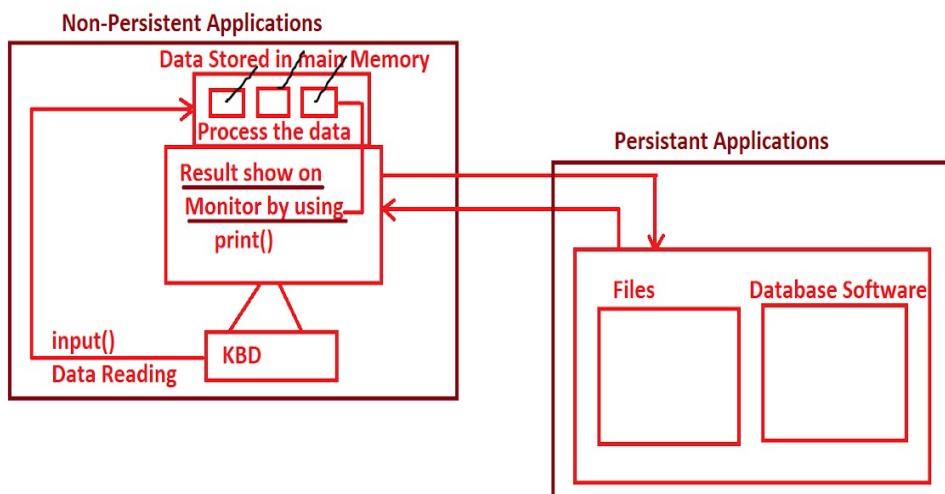
=>The purpose of read operation is that " To transfer or read the record from file of secondary memory into the object of main memory".

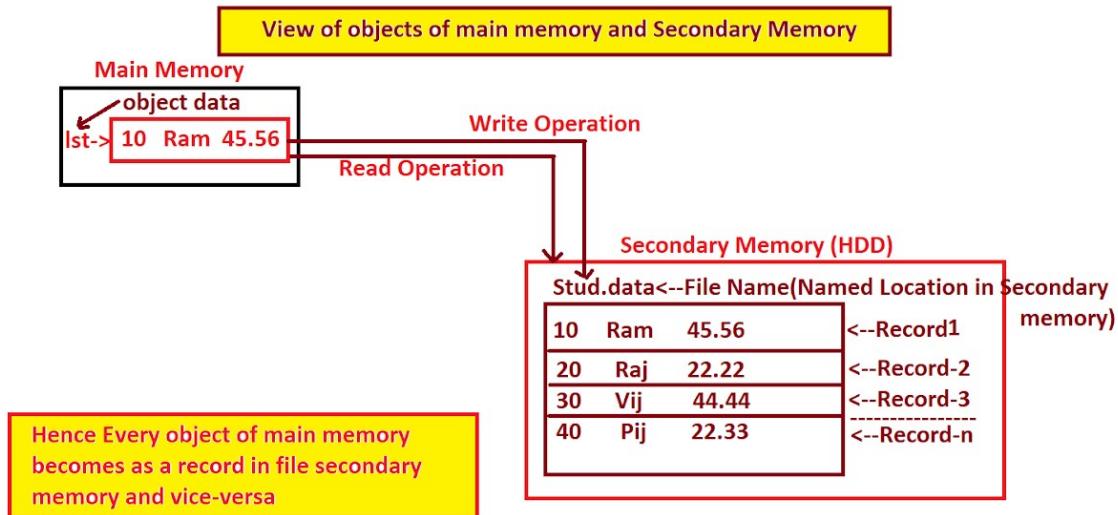
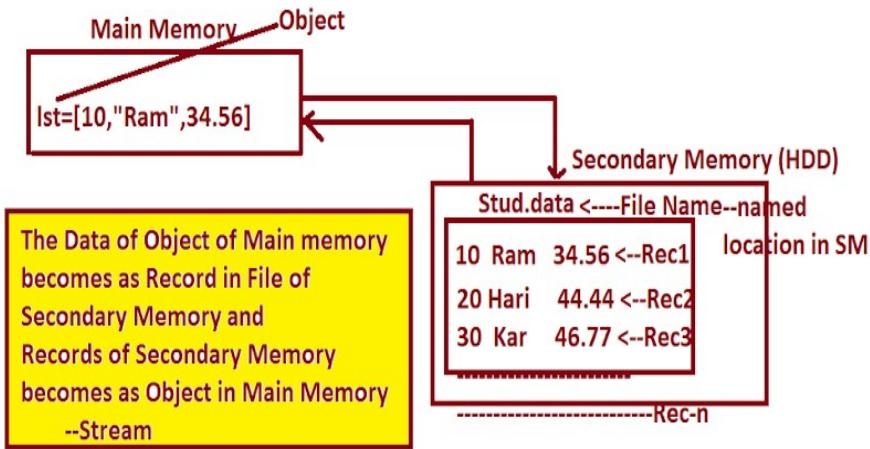
=>Steps

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

=>While we performing read operations, we get the following exceptions.

- a) FileNotFoundError
- b) EOFError





===== File Opening Modes =====

=>The purpose of File Opening Modes is that "In which type file mode we are opening the file".

=>In Python Programming, we have 8 File Opening Modes. They are

- 1. r 2. w 3. a
- 4. r+ 5. w+ 6. a+
- 7. x 8. x+

1) r

=>This mode is used for opening the file name in READ Mode.

=>If we open the file name in 'r' mode and if the file name does not exist then we get FileNotFoundError.

=>"r" mode is default mode of all file opening modes.

2) w

=>This mode is used for Creating File Name and Opening the file name in WRITE Mode always.

=>If we are opening NEW FILE in "w" mode then that File Created Newly and Opened in write mode

=>If we are opening EXISTING FILE in "w" then EXISTING FILE Data OVERLAPPED with new Data.

3) a

=>This mode is used for Creating File Name and Opening the file name in WRITE Mode always.

=>If we are opening NEW FILE in "a" mode then that File Created Newly and Opened in write mode

=>If we are opening EXISTING FILE in "a" then EXISTING FILE Data APPENDED with new Data.

4) r +

=>This mode is used for opening the file name in READ Mode.

=>If we open the file name in 'r+' mode and if the file name does not exist then we get FileNotFoundError.

=>With "r+" mode, First we Must read the data and Later we can perform Write Operation.

5) w +

=>This mode is used for Creating File Name and Opening the file name in WRITE Mode always.

=>If we are opening NEW FILE in "w" mode then that File Created Newly and Opened in write mode

=>If we are opening EXISTING FILE in "w" then EXISTING FILE Data OVERLAPPED with new Data.

=>With "w+" mode , Additionally we can perform Read Operation After Performing Write Operation.

6) a+

=>This mode is used for Creating File Name and Opening the file name in WRITE Mode always.

=>If we are opening NEW FILE in "a+" mode then that File Created Newly and Opened in write mode

=>If we are opening EXISTING FILE in "a+" then EXISTING FILE Data APPENDED with new Data.

=>With "a+" mode , Additionally we can perform Read Operation After Performing Write Operation.

7) x

=>This mode is used for Creating File Name and eXclusively Opening the file name in WRITE Mode

 always.

=>Once we open File Name in "x" mode , we can Perform Write Operations only.

=>If we open existing file in "x" mode then we get FileNotFoundError

8) x+

=>This mode is used for Creating File Name and eXclusively Opening the file name in WRITE Mode

 always.

=>Once we open File Name in "x+" mode , we can Perform Write Operations First and Later we can

 Perform Read Operations also

=>If we open existing file in "x+" then we get FileNotFoundError

=====

Opening Files

=====

=>To perform any type of Operation, we must open the file.

=>In Python Programming, To open the file, we have two approaches. They are

-
1. By using `open()`
 2. By using " with `open()` as "
-

1) By using `open()`

=>Syntax: `varname=open("File Name","File Mode")`

Explanation

=>varname represents File Pointer and always points to the content of file and it is an object of type TextIOWrapper

=>open() is one of the pre-defined Function, which is used for opening the specified File Name

 in Specified File Mode

=>File Name represents Name of the file

=>File Mode represents r,w,a,r+,w+,a+,x,x+

=>When we open the file with open() then it is recommended to close the file manually by using close(). In otherwords open() does not provide auto-closability of Files.

2) By using " with open() as "

Syntax:- with open("File Name","File Mode") as Varname:

Other statemenets out-of "with open() as " statements

Explanation:

=>Here "with" and "as" are are the key words

=>open() is one of the pre-defined Function, which is used for opening the specified File Name

 in Specified File Mode

=>varname represents File Pointer and always points to the content of file and it is an object of type TextIOWrapper

=>File Name represents Name of the file

=>File Mode represents r,w,a,r+,w+,a+,x,x+

=>The advantage of " with open() as " is that "Auto-Closability of File. In Otherwords, as long as PVM is executing Indentation block of " with open() as " then File Name is Active(Open) and Once PVM comes out-of Indentation block of " with open() as" then File Name closed automatically (no need to use close()).

X

Writing the data to the File

=>To write the data to the file, we have 2 pre-defined Functions in file pointer object. They are

- 1) write()
 - 2) writelines()
-
-

1)write()

--
=>Syntax: filepointerobj.write(strdata)
=>This Function is used for writing any type of Data to the file in the form of str
=>Examples: refer FileWriteEx1.py

2)writelines()

--
=>Syntax: filepointerobj.writelines(Iterable-object)
=>This Function is used for writing any type of Iterable Object to the file in the form of str
=>Examples: refer FileWriteEx2.py

#Program for demonstarting to open the file name in different modes

#FileOpenEx1.py

try:

 fp=open("stud1.data") # Here we ar eopening "stud.data" in read mode.

except FileNotFoundError:

 print("File does not exist")

else:

 print("File Opened Successfully in Read Mode:")

 print("Type of fp=",type(fp)) # Type of fp= <class '_io.TextIOWrapper'>

 print("Is File Closed--else block:", fp.closed)

 print("File Mode Name=",fp.mode)

finally:

 print("\nI am from finally block:")

 fp.close() # manual closing of file

 print("Is File Closed in finally block:", fp.closed)

#Program for demonstarting to open the file name in different modes

#FileOpenEx2.py

fp=open("stud1.data","w")

print("File Created Successfully in Write Mode:")

print("Type of fp=",type(fp))

#Program for demonstarting to open the file name in different modes

#FileOpenEx3.py

try:

 with open("stud3.data","r") as fp:

 print("File Opened Successfully in Read Mode:")

 print("Type of fp=",type(fp)) # Type of fp= <class '_io.TextIOWrapper'>

 print("Is File Closed--within with-open() as Indentation block:", fp.closed)

False

 print("\ni am out-of with-open() as Indentation block:")

 print("Is File Closed--out-of with-open() as Indentation block:", fp.closed) # True

except FileNotFoundError:

 print("File does not exist:")

```
#Program for demonstarting to open the file name in different modes
#FileOpenEx4.py
with open("stud4.data","a+") as fp:
    print("-"*50)
    print("File Opened in Write Mode:")
    print("Name of File =",fp.name)
    print("File Opening Mode=",fp.mode)
    print("Is File Writable=",fp.writable())
    print("Is File Readable=",fp.readable())
    print("-"*50)
```

```
#Program for demonstarting to open the file name in different modes
#FileOpenEx5.py
try:
    with open("kvr.data","x") as fp:
        print("-"*50)
        print("File Opened in Write Mode:")
        print("Name of File =",fp.name)
        print("File Opening Mode=",fp.mode)
        print("Is File Writable=",fp.writable())
        print("Is File Readable=",fp.readable())
        print("-"*50)
except FileNotFoundError:
    print("File Name alerady Exist:")
```

```
#Program for demonstarting to open the file name in different modes
#FileOpenEx6.py
try:
    with open("hyd.data","x+") as fp:
        print("-"*50)
        print("File Opened in Write Mode:")
        print("Name of File =",fp.name)
        print("File Opening Mode=",fp.mode)
        print("Is File Writable=",fp.writable())
        print("Is File Readable=",fp.readable())
        print("-"*50)
except FileNotFoundError:
    print("File Name alerady Exist:")
```

```
#Program for writing Address of a Person to file file---write()
#FileWriteEx1.py
with open("Addr1.data","a") as fp:
    fp.write("Travis El Olihpant\n")
    fp.write("FNO:13-14, Hill Side\n")
    fp.write("Numpy Foundation\n")
    fp.write("Nethet Lands-56\n")
    print("Data Written to the file")
```

```
#Program for writing Address of a Person to file file---writelines()
#FileWriteEx2.py
obj={10,20,30,10,20,30}
with open("Addr2.data","a") as fp:
    fp.writelines(str(obj)+"\n")print("Data Written to the file")
```

Reading the Data from File

=>To read the data from the file, we have 2 pre-defined functions present File Pointer object. They are

- 1) `read()`
 - 2) `readlines()`
-

1) `read()`

=>Syntax:- `varname=filepointerobj.read()`

=>This Function is used for reading the entire data from file and placed in LHS Varname in the form of

str.

Examples:

2) `readline()`

=>Syntax:- `varname=filepointerobj.readlines()`

=>This Function is used for reading the entire data from file the form of Line by line and placed in LHS

Varname in the form of list object.

Examples:

Types of Files

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

- a) Text Files
- b) Binary Files

1) Text File:

=>A Text File always contains Alphabets, Digits and Special Symbols.

=>Text Files always denoted by a letter "t"

=>The default file in python is text file.

Examples: `.py .java .c .cpp .txt .doc....etc`

2) Binary File:

=>A BinaryFile always contains Data in Binary Format (Pixles)

=>Binary Files always denoted by a letter "b"

Examples: images (.jpg, .jpeg, .png, .gif)
 audio and video files
 PDF documents with Images.

#Write a python program which will count number of lines , number of words and number of characters in any file.

#FileInfo.py

```
try:  
    fname=input("Enter File Name:")  
    nl=0  
    nw=0  
    nc=0  
    with open(fname,"r") as fp:  
        lines=fp.readlines()  
        for line in lines:  
            print(line,end="")  
            nl=nl+1  
            nw=nw+len(line.split())  
            nc=nc+len(line)  
    else:  
        print('*'*40)  
        print('Number of Lines:{}'.format(nl))  
        print("Number of Words:{} ".format(nw))  
        print("Number of Chars:{} ".format(nc))  
        print('*'*40)  
except FileNotFoundError:  
    print("File does not Exist");
```

#Write a python program which will copy an image

#ImageFileCopyEx1.py
sfile=input("Enter Source File:")
try:

```
    with open(sfile,"rb") as rp:  
        dfile=input("Enter Destination File:")  
        with open(dfile,"wb") as wp:  
            sfilldata=rp.read()  
            wp.write(sfilldata)  
            print("\nSource File Data Copied into Destination File:")
```

except FileNotFoundError:
 print("File does not Exist");

#Write a python program which will copy the content of one file into another file (text file)

#TextFileCopyEx1.py
sfile=input("Enter Source File:")
try:

```
    with open(sfile,"r") as rp:  
        dfile=input("Enter Destination File:")  
        with open(dfile,"a") as wp:  
            sfilldata=rp.read()  
            wp.write(sfilldata)  
            print("\nSource File Data Copied into Destination File:")
```

```
except FileNotFoundError:  
    print("File does not Exist");  
# Write a python program which will read the data from the file and display on the console  
#FileReadEx1.py----read()  
try:  
    filename=input("Enter any file name:")  
    with open(filename) as fp:  
        print("Initila Position of fp=",fp.tell())  
        print("-"*50)  
        filedatal=fp.read()  
        print(filedatal)  
        print("-"*50)  
        print("Now Position of fp=",fp.tell())  
    except FileNotFoundError:  
        print("File does not Exist");  
# Write a python program which will read the data from the file and display on the console  
#FileReadEx2.py----readline()  
try:  
    filename=input("Enter any file name:")  
    with open(filename) as fp:  
        print("Initial Position of fp=",fp.tell())  
        print("-"*50)  
        lines=fp.readlines()  
        for line in lines:  
            print(line,end="")  
        print("-"*50)  
        print("Now Position of fp=",fp.tell())  
    except FileNotFoundError:  
        print("File does not Exist");  
# Write a python program which demonstartes tell() and seek()  
#RandomFileAccessEx.py----tell() and seek()  
try:  
    with open("addr1.data") as fp:  
        print("Initila Position of fp before reading=",fp.tell()) # 0  
        print(fp.read())  
        print("Position of fp after reading=",fp.tell()) # ----163  
        print("-"*40)  
        fp.seek(6)  
        print("Now after Seek--Position of fp before reading=",fp.tell()) #  
        print(fp.read(10))  
        print("Position of fp after reading=",fp.tell()) # 16  
  
    except FileNotFoundError:  
        print("File does not Exist");
```

Pickling and Un-Pickling
(OR)
Object Serialization or Object De-Serialization

Pickling (Object Serialization)

- =>Let us assume there exist an object which contains multiple values. To save or write object data of main memory into the file of secondary memory by using write() and writelines() , they transfers the values in the form of value by value and it is one of the time consuming process(multiple write operations).
 - =>To Overcome this time consuming process, we must use the concept of Pickling.
 - =>The advantage of pickling concept is that with single write operation , we can save or write entire object data of main memory into the file of secondary memory.
 - =>Definition of Pickling:
-

=>The Process saving or transferring entire object content of main memory into the file of secondary memory by performing single write operation is called Pickling.

=>Pickling concept participates in Write Operations.

Steps for implementing Pickling Concept:

- =>import pickle module, here pickle is one of the pre-defined module
- =>Choose the file name and open it into write mode.
- =>Create an object with collection of values (Iterable object)
- =>use the dump() of pickle module. dump() save the content of any object into the file with single write operation.

Syntax: pickle.dump(object , filepointer)

=>NOTE That pickling concept always takes the file in Binary Format.

Un-Pickling (Object De-Serialization)

- =>Let us assume there exists a record with multiple values in a file of secondary memory. To read or transfer the entire record content from file of secondary memory, if we use read(), readlines() then they read record values in the form of value by value and it is one of the time consuming process(multiple read operations).

=>To overcome this time consuming process, we must use the concept of Un-pickling.

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

=>Definition of Un-Pickling:

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

=>Un-Pickling concept participates in Read Operations.

Steps for implementing Un-Pickling Concept:

=>import pickle module
=>Choose the file name and open it into read mode.
=>Use the load() of pickle module. load() is used for transferring or loading the entire record content from file of secondary memory into object of main memory.
 Syntax: objname=pickle.load(filepointer)
=>NOTE That Un-pickling concept always takes the file in Binary Format.

#WAPP which will read emp no, emp name and salary and save them in a file---Program--(A)

```
#EmpPickEx1.py
import pickle,sys
with open("emp.data","ab") as fp:
    while(True):
        print("-"*50)
        #read emp values from key board
        eno=int(input("Enter Employee Number:"))
        ename=input("Enter Employee Name:")
        sal=float(input("Enter Emp Salary:"))
        #place emp values in Iterable Object--list
        l1=list() #empty list
        l1.append(eno)
        l1.append(ename)
        l1.append(sal)
        #Save list object data into file
        pickle.dump(l1,fp)
        print("-"*50)
        print("\tEmployee Record Saved Successfully in File:")
        print("-"*50)
        ch=input("Do u want to insert another record(yes/no):")
        if(ch.lower() == "no"):
            print("Thx for using this program")
            sys.exit()
```

#WAPP which will read emp records from the file---Program--(B)

```
#EmpUnPick.py
import pickle
try:
    with open("emp.data","rb") as fp:
        print("-"*50)
        print("\tEmpno\tName\tSal")
        print("-"*50)
        while(True):
            try:
                obj=pickle.load(fp) # Which ever object was pickled
                and same type of object would be un-pickled
                for val in obj:
                    print("\t{}{}".format(val),end="")
                print()
            except EOFError:
                print("-"*50)
                break
```

```

except FileNotFoundError:
    print("File does not exist")


---


#Write a python program which will accept number of student in the college.Accept
individual details of student such as student number, student name and marks save the student
details as a record in a file
#StudPick.py
import pickle
def savestudrecords():
    nos=int(input("Enter How many students u have:"))
    if(nos<=0):
        print("{} is invalid number of students:".format(nos))
    else:
        with open ("stud.data","ab") as fp:
            for i in range(1,nos+1):
                print("-"*50)
                print("Enter {} Student Information:".format(i))
                print("-"*50)
                sno=int(input("Enter Student Number:"))
                sname=input("Enter Student Name:")
                marks=float(input("Enter Student Marks:"))
                l=list();
                l.append(sno)
                l.append(sname)
                l.append(marks)
                pickle.dump(l,fp);
                print("-"*50)
                print("{} Student Record Saved in a File".format(i))

#main program
savestudrecords()


---


#write a python program which will read the student record from the file
#StudUnpick.py
import pickle,sys
try:
    with open("stud.data","rb") as fp:
        print("-"*50)
        print("\tStno\tName\tMarks")
        print("-"*50)
        while(True):
            try:
                studrec=pickle.load(fp)
                for val in studrec:
                    print("\t{}".format(val),end="")
                else:
                    print()
            except EOFError:
                print("-"*50)
                break
except FileNotFoundError:
    print("File does not exist")

```

Working with CSV Files in Python

- =>CSV stands for Comma Separated Values.
- =>A CSV File is one of the simple file format used to store tabular data, such as a spreadsheet or database.
- =>A CSV file stores tabular data (numbers and text) in plain text.
- =>Each line of the CSV file is a data record. Each record consists of one or more fields, separated by commas.
- =>Python provides an in-built module called csv to work with CSV files.
- =>There are 2 classes provided by this module for writing the data to CSV File. They are
 - 1) By using csv.writer class object
 - 2) By Using csv.DictWriter class object

1) By using csv.writer class object

- =>The csv.writer class object is used to insert data to the CSV file.
- =>To create an object of csv.writer class object, we use writer() and present in csv module.
- =>csv.writer class object provides two Functions for writing to CSV file.
- =>They are
 - 1) writerow()
 - 2) writerows()

1) writerow(): This Function writes a single row at a time.
Field row / Header can also be written using this Function.

Syntax:- csvwriterobj.writerow(fields Row / Data Row)

2) writerows(): This Function is used to write multiple rows at a time.
This can be used to write rows list.

Syntax: Writing CSV files in Python
csvwriterobj.writerow(data rows)
here data rows can be list, tuple set,frozenset only

2) By Using csv.DictWriter class object

- =>The csv.DictWriter class object is used to insert dict data to the CSV file.
- =>To create an object of csv.DictWriter class object, we use DictWriter() and present in csv module.
- =>csv.DictWriter class object provides two Functions for writing to CSV.

- 1) writeheader()
- 2) writerows()

1) writeheader():

- =>writeheader() method simply writes the first row of your csv file using the pre-specified fieldnames.

Syntax: DictWriterObj.writeheader()

2) writerows():

=>writerows() method simply writes all the values of (Key,Value) from dict object in the form of separate rows[Note: it writes only the values(not keys)]

Syntax:- DictWriterObj.writerow(dictobject)

Reading the data from CSV File

=>There are various ways to read a CSV file that uses either the CSV module or the pandas library.

=>The csv Module provides two classes for reading information from CSV file .

- 1) csv.reader
 - 2) csv.DictReader
-

1) csv.reader():

=>This Function is used for creating an object of csv.reader class and It helps us to read the data records from csv file.

=>Syntax:- csvreaderobj=csv.reader(filepointer)

2) csv.DictReader():

=>This Function is used for creating an object of csv.DictReader class and It helps us to read the data from csv file where it contains dict data(Key,Value).

=>Syntax:- csvdictreaderobj=csv.DictReader(filepointer)

```
# importing the csv module
#csvdictwriteex1.py
import csv

# my data rows as dictionary objects
mydict =[ {'branch': 'COE', 'cgpa': '9.0', 'name': 'Nikhil', 'year': '2'},
          {'branch': 'COE', 'cgpa': '9.1', 'name': 'Sanchit', 'year': '2'},
          {'branch': 'IT', 'cgpa': '9.3', 'name': 'Aditya', 'year': '2'},
          {'branch': 'SE', 'cgpa': '9.5', 'name': 'Sagar', 'year': '1'},
          {'branch': 'MCE', 'cgpa': '7.8', 'name': 'Prateek', 'year': '3'},
          {'branch': 'EP', 'cgpa': '9.1', 'name': 'Sahil', 'year': '2'} ]

# field names
csvfields = ['name', 'branch', 'year', 'cgpa']

# name of csv file
filename = "univ2.csv"

# writing to csv file
with open(filename, 'w') as fp:
    # creating a csv dict writer object
    dictwriter = csv.DictWriter(fp, fieldnames = csvfields)
    # writing headers (field names)
    dictwriter.writeheader()
```

```

# writing data rows
dictwriter.writerows(mydict)
print("\nDict Data Written to the csv file--verify")


---


#Program for Reading the Data from CSV File
#CsvReadEx1.py
try:
    with open("E:\\KVR-PYTHON-11AM\\FILES\\NOTES\\student.csv","r") as fp:
        csvdata=fp.read()
        print(csvdata)
except FileNotFoundError :
    print("CSV File does not exist:")



---


#Note: This Type of Reading the data from File is called NON-CSV Reading


---


# Python program to demonstrate writing to CSV File
#csvwriteex1.py
import csv
# field names OR Header Names
recfields = ['Name', 'Branch', 'Year', 'CGPA']
# data rows of csv file
rows = [ ['Nikhil', 'COE', '2', '9.0'],
         ['Sanchit', 'COE', '2', '9.1'],
         ['Aditya', 'IT', '2', '9.3'],
         ['Sagar', 'SE', '1', '9.5'],
         ['Prateek', 'MCE', '3', '7.8'],
         ['Sahil', 'EP', '2', '9.1'] ]
# name of csv file
csvfilename = "univ1.csv"
# writing data to csv file
with open(csvfilename, 'w') as fp:
    # creating a csv writer object
    csvwriter = csv.writer(fp)
    # writing the fields
    csvwriter.writerow(recfields)
    # writing the data rows
    csvwriter.writerows(rows)
    print("\nCSV file Created and Verify")


---


# Python program to demonstrate to write single record
# writing single record to CSV file
#csvwriteex2.py
import csv

# data record of csv file
row = ('Rajan', 'ECE', '3', '9.9')

# name of csv file
filename = "univ1.csv"
# writing to csv file
with open(filename, 'a') as fp:
    # creating a csv writer object
    cw = csv.writer(fp)

```

```
# writing the data row to the csv file
cw.writerow(row)
print("\nSingle Record Written to the CSV File:")


---


#Program for Reading the Data from CSV File
#To read the data from csv file, we must create an object csv.reader object
# csv.reader(filepointer) which gives an object of csv.reader
#PureCsvReadEx1.py
import csv
try:
    with open("E:\\KVR-PYTHON-11AM\\FILES\\NOTES\\student.csv","r") as fp:
        csvr=csv.reader(fp)
        for record in csvr:
            for val in record:
                print("\t{}{}".format(val),end="")
        print()
except FileNotFoundError :
    print("CSV File does not exist:")


---


#Program for Reading the Data from CSV File
#To read the data from csv file, we must create an object csv.reader object
# csv.reader(filepointer) which gives an object of csv.reader
#PureCsvReadEx2.py
import csv
try:
    with open("univ1.csv","r") as fp:
        csvr=csv.reader(fp)
        for record in csvr:
            for val in record:
                print("\t{}{}".format(val),end="")
        print()
except FileNotFoundError :
    print("CSV File does not exist:")


---


#Program for Reading the Data from CSV File
#To read the data from csv file, we must create an object csv.reader object
# csv.reader(filepointer) which gives an object of csv.reader
#PureCsvReadEx3.py
import csv
try:
    with open("univ2.csv","r") as fp:
        dictcsvr=csv.DictReader(fp)
        for record in dictcsvr:
            for key,val in record.items():
                print("\t{}-->{}{}".format(key,val))
        print()
except FileNotFoundError :
    print("CSV File does not exist:")
```

PYTHON WITH DATABASE COMMUNICATION

===== Python DataBase Communication (PDBC) =====

=>Even we achieved the Data Persistence by using Files, Files has the following Limitations.

- provide
security in the form of User Name and Password.
1. Files of any language does not contain security bcoz Files are unable to
2. Files are unable to store large amount of data
3. File are differing from One OS to another OS (Files are OS
dependent)
4. Querying and Processing the data from Files is Very Complex bcoz file
data is
organized w.r.t Indices and identifying the indices is very complex.
5. Files does not contain Column Names (Except CSV Files)

=>To Overcome the limitation of files and to achieve the Data Persistence, we must use the concept of any RDBMS DataBase Softwares (Oracle, MYSQL, Mongo DB, DB2, SQL Server, Postgey SQL, SQLITE3.....etc).

- DataBase
1. All RDBMS DataBase Softwares Provides Security bcoz RDBMS
Softwares considers User names and Password.
2. All RDBMS DataBase Softwares stores large amount of data
3. All RDBMS DataBase Softwares Arch Remains Same on all types of
OSes (OS
Independent)
4. Querying and Processing the data from All RDBMS DataBase
Softwares is Very
Simple bcoz data of All RDBMS DataBase Softwares organized in
the of Tables with Column Names.
5. The Data Present in any RDBMS DataBase Softwares organized in
the of Tables
with Column Names.

=>If Python Program want to communicate with any RDBMS DataBase Softwares then we must use a PRE-DEFINED MODULE and such PRE-DEFINED MODULE does not exist in Python Software.

=>Some Third Party Software Vendors(Ex: "Anthony Tuininga") developed a Module for Python Programmers to communicate with RDBMS DataBase Softwares and placed in github and Third Party Software Modules must be installed.

=>To install any Third Party Software Modules in python , we use a tool called pip and it is present in C:\Users\KVR\AppData\Local\Programs\Python\Python310\Scripts folder.

=>Syntax : pip install Module Name (at any Windows command prompt)

=>If Python Program want to communicate with Oracle Database, then we must install

`cx_Oracle` Module.

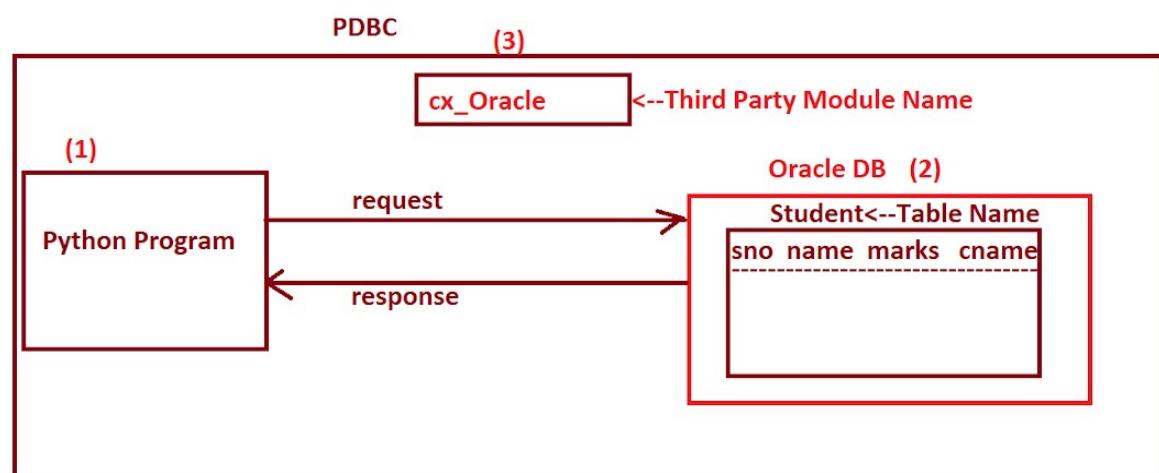
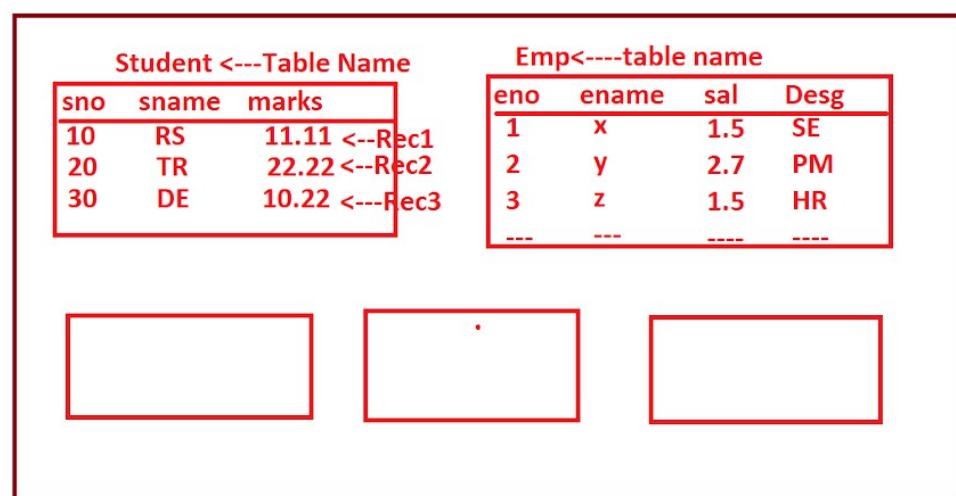
=>Examples : `pip install cx_Oracle`

=>If Python Program want to communicate with MySQL Database, then we must install `mysql-connector` or `mysql-connector-python` Module.

=>Examples : `pip install mysql-connector`

=>Examples : `pip install mysql-connector-python`

Oracle Database view <----Collection tables<---Collection of Records



Communication between Python Program and Oracle Database

=>In order to write python program to communicate with Oracle Database, we must follow 6 steps. They are

1. import cx_Oracle module
 2. Python Program must get the connection from Oracle DB
 3. Create an object of Cursor
 4. Python program must Prepare the Query and Execute the Query in Oracle DB
 5. Python Program Process the Result of the Query.
 6. Python Program closes the connection.
-

Explanation:

1. import cx_Oracle module:
-

=>If a python Program want to perform any database operations(insert , delete , update record , read records..etc) then we must import a pre-defined third party module "cx_Oracle".

=>We know that a module is a collection of Variables, Function Name and Class Names.

Examples: import cx_Oracle

2. Python Program must get the connection from Oracle DB
-

=>To do any Data Base Operations, First python program must get the connection from Oracle.

=>To get the connection from any Database, we use connect() which is present in cx_Oracle module.

=>Syntax:- varame=cx_Oracle.connect("Connection URL")

=>Here connection URL Represents " UserName/Password@DNS/Serviceid "

(OR)

"

UserName/Password@IPAddress/Serviceid "

=>Here Varname is an object of <class, cx_Oracle.Connection>

=>Here "UserName" represents User Name of Oracle Data Base (Ex: scott)

=>here "password" represents Password of Oracle Data Base (Ex: tiger)

=>here DNS(Domain Naming Service) represents name of the machine where Database Software Installed. The default Name of Every Machine is "localhost".

=>Here IPAddress (Internet Protocol Address) represents An address of Physical Machine where Database software Installed. The default IP Address of Every Machine is 127.0.0.1 (Loop back address)

=>Here "serviceid" represents on which name Oracle data base Installed in current working machine. To find Service Id in Oracle Data base, we use the following at SQL Environment

```
SQL> select * from global_name;
```

GLOBAL_NAME
ORCL <-----Service id

=>When we use / write Invalid Connection URL then we get cx_Oracle.DatabaseError as an exception and must handle.

3. Create an object of Cursor

=>The purpose of creating an object of Cursor is that "To carry the Query from Python Program, hand over to Database, and obtains Result from Database and Gives to Python Program".

=>To create an object of Cursor, we use cursor() which is present in Connection Object.

=>Syntax: varname=conobj.cursor()

=>Here Varname represents an object of <class, cx_Oracle.Cursor>

4. Python program must Prepare the Query and Execute the Query in Oracle DB

=>A Query is a statement or Request or Question to database software for obtaining data base results.

=>To execute the query in any Database software from Python Program, we use execute() which is

present in cursor object.

=>Syntax: cursorobj.execute("Query")

=>Here Query is of type str and In any database software we have different Queries (DDL,DML,DRL)

5. Python Program Process the Result of the Query.

=>After Executing DML statements, the result of DML statements is present in cursor object. To extract the result from cursor object, we use "rowcount" attribute of cursor object. "rowcount" attribute gives number of updated / deleted / inserted in the the data base.

=>After Executing DRL statements, the result of DRL statements is present in cursor object. To extract the from cursor object, have 3 Functions in cursor object. They are

- a) fetchone()
- b) fetchmany(no. of records)
- c) fetchall()

=>fetchone() is used for obtaining One Record at a Time in the form of tuple. if no records found then this function returns None.

=>fetchmany(no. of records) is used for obtaining specified number of records.

case-1: if specified number of records==0 then this function obtains all records
case-2: if specified number of records<=Total Number of Records then this
function gives specified number of records
case-3: if specified number of records>Total Number of Records then this function
obtains all records
case-4: if specified number of records<0 then this function never gives any records.

=>fetchall() is used for obtaining all the records from cursor object.

#Write a python program which will obtain the connection from oracle database.

```
#TestOracleConEx1.py
import cx_Oracle # step-1
try:
```

```
    con=cx_Oracle.connect("scott/tiger@localhost/orcl") # Step-2
    print("\nType of con=",type(con))
    print("Python Program obtains connection from Oracle DB")
except cx_Oracle.DatabaseError as d:
    print(d)
```

#Write a python program which will obtain the connection from oracle database.

```
#TestOracleConEx2.py
import cx_Oracle # step-1
try:
```

```
    con=cx_Oracle.connect("scott/tiger@127.0.0.1/orcl") # Step-2
    print("\nType of con=",type(con))
    print("Python Program obtains connection from Oracle DB")
except cx_Oracle.DatabaseError as d:
    print(d)
```

#Write a python program which will create an object of cursor.

```
#CursorEx1.py
import cx_Oracle # 1
try:
```

```
    con=cx_Oracle.connect("scott/tiger@localhost/orcl") # 2
    print("\nConnection Obtained from Oracle DB")
    cur=con.cursor() # 3
    print("\nType of cur=",type(cur))
    print("Cursor object created:")
except cx_Oracle.DatabaseError as db:
    print(db)
```

===== Types of Queries in Database Softwares =====

=>In any database, we have 3 types of Queries. They are

1. DDL(Data Definition Language) Queries
 2. DML (Data Manipulation Language) Queries
 3. DRL (Data Retrieval Language) Queries
-

1. DDL(Data Definition Language) Queries

=>The purpose of DDL Queries is that to deal with Physical Level of Database software such as creation of Tables, altering column sizes, adding new Columns etc.

=>In any Database software , we have 3 types of DDL Queries. They are

1. create
 2. alter
 3. drop
-

1) create:

=>This Query is used for creating a table in Oracle Database

=>Syntax:-

```
SQL> create table <table-name> ( col name1 database data type, col  
name2 database data type, .....col name-n database data type )
```

```
SQL> create table employee(eno number(2) primary key ,ename varchar2(10) not null , sal  
number (6,2) not null);
```

2. alter :

=>This Query is used for alter the table structure such as modifying (modify) the column sizes and adding (add) new columns.

Syntax1:- SQL> alter table <table-name> modify (existing col name1 database data type....
existing col name-n database data type)

Syntax2:- SQL> alter table <table-name> add (new col name1 database data type....
new col name-n database data type)

Examples: SQL> alter table employee add(cname varchar2(10));
SQL> alter table employee modify(ename varchar2(20), sal
number(8,2));

3) drop :

=>This query is used for removing the table from Database Software

=>Syntax:- SQL> drop table <table-name>

=>Example:- SQL> drop table employee

#Write a python program which will add new column name called company name to employee table

#AlterwithAddEx.py

```
import cx_Oracle
```

```
def alterwithadd():
```

```
    try:
```

```
con=cx_Oracle.connect("scott/tiger@127.0.0.1/orcl")
cur=con.cursor()
aq="alter table employee add(cname varchar2(10) not null)"
cur.execute(aq)
print("Employee Table Altered")
except cx_Oracle.DatabaseError as db:
    print("Problem in Database:",db)
```

#main programs
alterwithadd()

#Write a python Program Which will Change the column sizes of employee table(eno,ename)

#AlterwithModifyEx.py

```
import cx_Oracle
```

```
def alterwithmodify():
```

```
    try:
```

```
        con=cx_Oracle.connect("scott/tiger@127.0.0.1/orcl")
        cur=con.cursor()
        aq="alter table employee modify(eno number(3),name varchar2(15))"
        cur.execute(aq)
        print("Employee Table Altered")
    except cx_Oracle.DatabaseError as db:
        print("Problem in Database:",db)
```

#main programs
alterwithmodify()

#Write a python program which will create employee table in oracle database.

#TableCreateEx1.py

```
import cx_Oracle
```

```
def tablecreate():
```

```
    try:
```

```
        con=cx_Oracle.connect("scott/tiger@localhost/orcl")
        cur=con.cursor()
        #prepare the query and execute
        ctq="create table employee(eno number(2) primary key, name
varchar2(10) not null,sal number(5,2) not null )"
        cur.execute(ctq)
        print("Employee Table Created Successfully")
    except cx_Oracle.DatabaseError as db:
        print("Problem in Database:",db)
```

#main program
tablecreate()

===== 2. DML (Data Manipulation Language) Queries =====

=>The purpose of DML operations is that To manipulate the table such Inserting the records, deleting the records and updating the records.

=>In RDBMS database softwares, we have 3 types of DML Operations. They are

1. insert
2. delete
3. update

=>When we execute any DML Operation through python program, we must use commit() for permanent change / update / modification and to roll back we use rollback().

=>commit() and rollback() are present in connection object.

-
1. insert:

=>This query is used for inserting a record in table of database.

=>Syntax:- SQL> insert into <table-name> values(val1 for column1, val2 for column2.....

val-1

Example: SQL> insert into student values (20,'DR',33.45,'C');

SQL>insert into student values (10,'RS',23.45,'Python');
SQL> commit ;

-
2. delete

=>This query is used for deleting a record .

=>Syntax1: delete from <table name>

=>Syntax2: delete from <table-name> where cond list;

=>Example: SQL> delete from student where sno=70;

3. update

=>This query is used for updating the record values

=>Syntax1:

SQL> update <table-name> set col1=val1,col2=val2.....col-n=val-n;

=>Syntax2:

SQL> update <table-name> set col1=val1,col2=val2.....col-n=val-n
where cond list;

Examples: SQL> update student set marks=marks+marks*0.02;

Examples: SQL> update student set marks=marks+marks*0.05,crs='Django' where sno=90;

#Write a python program which will access employees number employees name,salary and designation from keyboard and insert as a record in employee table

#RecordInsertEx1.py

import cx_Oracle

def insertRecord():

try:

con=cx_Oracle.connect("scott/tiger@localhost/orcl")

```
cur=con.cursor()
iq="insert into employee values(40,'Kinney',1.2,'Pandas')"
cur.execute(iq)
con.commit()
print("Record Inserted Successfully in Employee Table")
except cx_Oracle.DatabaseError as db:
    print("Problem in Database:",db)
```

#main program
insertRecord()

#Write a python program which will access employees number employees name,salary and designation from keyboard and insert as a record in employee table

```
#RecordInsertEx2.py
import cx_Oracle
def insertRecord():
    try:
        con=cx_Oracle.connect("scott/tiger@localhost/orcl")
        cur=con.cursor()
        #accept employee values from KBD
        eno=int(input("Enter Employee Number:"))
        ename=input("Enter Employee Name:")
        empsal=float(input("Enter Employee Salary:"))
        cname=input("Enter Employee Company Name:")
        #Prepare the query and execute
        iq="insert into employee values(%d,'%s',%f,'%s' ) "
        cur.execute("insert into employee values(%d,'%s',%f,'%s' ) "
        %(eno,ename,empsal,cname) )
        #OR cur.execute(iq %(eno,ename,empsal,cname) )
        con.commit()
        print("{} Record Inserted Successfully in Employee
Table".format(cur.rowcount ))
    except cx_Oracle.DatabaseError as db:
        print("Problem in Database:",db)
    except ValueError:
        print("Don't eneter alnums,strs and symbols for empno, salary")
```

#main program
insertRecord()

#Write a python program which will access employees number employees name,salary and designation from keyboard and insert as a record in employee table

```
#RecordInsertEx3.py
import cx_Oracle,sys
def insertRecord():
    while(True):
        try:
            con=cx_Oracle.connect("scott/tiger@localhost/orcl")
            cur=con.cursor()
            print("-"*50)
            #accept employee values from KBD
            eno=int(input("Enter Employee Number:"))
```

```

ename=input("Enter Employee Name:")
empsal=float(input("Enter Employee Salary:"))
cname=input("Enter Employee Company Name:")
#Prepare the query and execute
iq="insert into employee values(%d,'%s',%f,'%s' ) "
cur.execute("insert into employee values(%d,'%s',%f,'%s' ) "
%(eno,ename,empsal,cname) )
#OR cur.execute(iq %(eno,ename,empsal,cname) )
con.commit()
print("-"*50)
print("{} Record Inserted Successfully in Employee
Table".format(cur.rowcount ))
print("-"*50)
ch=input("Do u want to insert another record(yes/no):")
if(ch.lower() == "no"):
    print("Thx for using this program")
    sys.exit()
print("-"*50)
except cx_Oracle.DatabaseError as db:
    print("Problem in Database:",db)
except ValueError:
    print("Don't eneter alnums,strs and symbols for empno, salary")

```

```

#main program
insertRecord()
#Write a python program which will delete the record from employee table based employee
number
#RecordDeleteEx.py
import cx_Oracle,sys
def deleteRecord():
    while(True):
        try:
            con=cx_Oracle.connect("scott/tiger@localhost/orcl")
            cur=con.cursor()
            print("-"*50)
            #accept employee values from KBD
            empno=int(input("Enter Employee Number for deleting the
Record:"))
            #Prepare the query and execute
            cur.execute("delete from employee where eno=%d" %empno)
            con.commit()
            if(cur.rowcount>0):
                print("{} Record deleted
Successfully".format(cur.rowcount))
            else:
                print("Employee Record does not exist:")
            print("-"*50)
            ch=input("Do u want to delete another record(yes/no):")
            if(ch.lower() == "no"):

```

```

        print("Thx for using this program")
        sys.exit()
        print("-"*50)
    except cx_Oracle.DatabaseError as db:
        print("Problem in Database:",db)
    except ValueError:
        print("Don't enter alnums,strs and symbols for empno, salary")

#main program
deleteRecord()


---


#Write a python program which will update salary by 50% for those employee whose salary
is less than 1.0
#RecordUpdateEx.py
import cx_Oracle
def updateRecord():
    try:
        con=cx_Oracle.connect("scott/tiger@localhost/orcl")
        cur=con.cursor()
        uq="update employee set sal=sal+sal*(50/100) where sal<1.0"
        cur.execute(uq)
        con.commit()
        print("{} Record updated Successfully in Employee
Table".format(cur.rowcount ))
    except cx_Oracle.DatabaseError as db:
        print("Problem in Database:",db)
#main program
updateRecord()


---



```

===== 3. DRL (Data Retrieval Language) Queries =====

=>DRL (Data Retrieval Language) Queries are used for Reading the records from table.
=>To read the records from table, we use "select"
=>In Otherwords "select" comes under DRL (Data Retrieval Language) Query.
=>Syntax1: SQL>select col1,col2,....col-n from <table-name>
=>Syntax2: SQL>select col1,col2,....col-n from <table-name> where cond list
=>Syntax3: SQL>select * from <table-name>
=>Syntax4: SQL>select * from <table-name> where cond list

=>Once the select query executed, all records are present in the object of cursor in Python.
=>To get the records from cursor object, we have 3 functions. They are

- 1) fetchone()
 - 2) fetchmany(no. of records)
 - 3) fetchall()
-

1) fetchone():

=>This function is used obtaining One Record at a time, where cursor object pointing

2) fetchmany(no. of records)

=>fetchmany(no. of records) is used for obtaining specified number of records.
case-1: if specified number of records==0 then this function obtains all records
case-2: if specified number of records<=Total Number of Records then this
function gives specified number of records
case-3: if specified number of records>Total Number of Records then this function
obtains all records
case-4: if specified number of records<0 then this function never gives any records.

3) fetchall()

=>fetchall() is used for obtaining all the records from cursor object.

#program for selecting or reading records from employee table---fetchone()

#SelectRecordEx1.py

```
import cx_Oracle
def selectrecords():
    con=cx_Oracle.connect("scott/tiger@localhost/orcl")
    cur=con.cursor()
    cur.execute("select * from employee")
    print("-"*50)
    while(True):
        rec=cur.fetchone()
        if(rec!=None):
            for val in rec:
                print("\t{}".format(val),end="")
            print()
        else:
            print("-"*50)
            break
```

#main program

selectrecords()

#program for selecting or reading records from employee table---fetchmany()

#SelectRecordEx2.py

```
import cx_Oracle
def selectrecords():
    con=cx_Oracle.connect("scott/tiger@localhost/orcl")
    cur=con.cursor()
    cur.execute("select * from employee order by name ASC") # No. records in
employee tab=6
    print("-"*50)
    records=cur.fetchmany()
    for record in records:
        for val in record:
            print("\t{}".format(val),end="")
        print()
    print("-"*50)
```

#main program

selectrecords()

#program for selecting or reading records from employee table---fetchall()

#SelectRecordEx3.py

```

import cx_Oracle
def selectrecords():
    con=cx_Oracle.connect("scott/tiger@localhost/orcl")
    cur=con.cursor()
    cur.execute("select * from employee order by name") # No. records in employee
tab=6
    print("-"*50)
    records=cur.fetchall()
    for record in records:
        for val in record:
            print("\t{}{}".format(val),end="")
            print()
    print("-"*50)
#main program
selectrecords()



---


#program for displaying col Names of a table
#ColNamesEx1.py
import cx_Oracle
def colnames():
    con=cx_Oracle.connect("scott/tiger@localhost/orcl")
    cur=con.cursor()
    cur.execute("select * from employee")
    colnamesinfo=cur.description
    print("*50)
    for colname in colnamesinfo:
        print(colname[0],end="\t")
    print()
    print("*50)

#main program
colnames()



---


#program for displaying col Names of a table
#ColNamesEx2.py
import cx_Oracle
def colnames():
    con=cx_Oracle.connect("scott/tiger@localhost/orcl")
    cur=con.cursor()
    cur.execute("select * from emp")
    print("*80)
    for cname in [colname[0] for colname in cur.description]:
        print("\t{}{}".format(cname),end="")
    print()
    print("*80)

#main program
colnames()



---


#Write a python program which will accept any table name and print all the records along
with column names.
#CompleteTable.py
import cx_Oracle

```

```

def tablerecords():
    try:
        con=cx_Oracle.connect("scott/tiger@localhost/orcl")
        cur=con.cursor()
        #accept the table name from keyboard
        tname=input("Enter table Name:")
        cur.execute("select * from %s" %tname)
        #Display column Names
        print("*"*80)
        for cname in [cname[0] for cname in cur.description]:
            print("\t{}".format(cname),end="")
        print()
        print("*"*80)
        #display records
        records=cur.fetchall()
        for record in records:
            for val in record:
                print("\t{}".format(val),end="")
            print()
        print("*"*80)
    except cx_Oracle.DatabaseError as db:
        print("prob in DB:",db)

```

#main program
tablerecords()

#Write a python program which will obtain connection from MySQL databases
#MySQLConnTest.py
import mysql.connector
con=mysql.connector.connect(host="localhost",
 user="root",
 passwd="root",

print("Type of con=",type(con))
print("Python Program Obtains Connection from MySQL")

#Write a python program which will obtain connection and cursor objects in MySQL
databases
#MySQLCursorTest.py
import mysql.connector
try:
 con=mysql.connector.connect(host="localhost",

print("\nType of con=",type(con))
print("Python Program Obtains Connection from MySQL")
cur=con.cursor()
print("\nType of cur=",type(cur))
print("Python Program Created Cursor object:")
except mysql.connector.DatabaseError as db:
 print("Prob in MySQL DB:",db)

```
#Write a Python program which will create a database in the name of batch 11am in my SQL
database.
#mysqldbcreate.py
import mysql.connector
try:
    con=mysql.connector.connect(host="localhost",
                                user="root",
                                passwd="root")
    print("Python Program obtains connection MYSQL")
    cur=con.cursor()
    #create a database name --->batch11am
    dq="create database HydPython"
    cur.execute(dq)
    print("Database created Successfully")
except mysql.connector.DatabaseError as db:
    print("Prob inb MySQL DB:",db)

#Write a python program which will create employee table in batch11am database of
MySQL
#TableCreateEx.py
import mysql.connector
def createtable():
    try:
        con=mysql.connector.connect(host="localhost",
                                    user="root",
                                    passwd="root")
        cur=con.cursor()
        #prepare the query and execute
        tq="create table student(sno int primary key, name varchar(10) not null,
marks float not null)"
        cur.execute(tq)
        print("Table created successfully")
    except mysql.connector.DatabaseError as db:
        print("Prob inb MySQL DB:",db)

#main program
createtable()

#Write a python program which will insert employee records in MYSQL by reading the
values from keyboard
#RecordInsertEx.py
import mysql.connector,sys
def insertRecord():
    while(True):
        try:
            con=mysql.connector.connect(host="localhost",
                                        user="root",
                                        passwd="root")
            cur=con.cursor()
            print("-"*50)
            print("Enter Employee Record")
            sno=int(input("Enter Sno:"))
            name=input("Enter Name:")
            marks=float(input("Enter Marks:"))
            cur.execute("insert into student values(%d,'%s',%f)"%(sno,name,marks))
            con.commit()
            print("Record Inserted")
        except mysql.connector.DatabaseError as db:
            print("Prob inb MySQL DB:",db)
        ch=input("Do you want to insert more record(y/n):")
        if(ch=='n' or ch=='N'):
            break
```

```

#accept employee values from KBD
eno=int(input("Enter Employee Number:"))
ename=input("Enter Employee Name:")
empsal=float(input("Enter Employee Salary:"))
#Prepare the query and execute
cur.execute("insert into employee values(%d,'%s',%f) "
%(eno,ename,empsal))
con.commit()
print("-"*50)
print("{} Record Inserted Successfully in Employee
Table".format(cur.rowcount))
print("-"*50)
ch=input("Do u want to insert another record(yes/no):")
if(ch.lower() == "no"):
    print("Thx for using this program")
    sys.exit()
print("-"*50)
except mysql.connector.DatabaseError as db:
    print("Problem in Database:",db)
except ValueError:
    print("Don't enter alnums,strs and symbols for empno, salary")

#main program
insertRecord()

#WAPP which will Delete a record based on employee number from employee table of
mysql
#RecordDeleteEx.py
import mysql.connector,sys
def deleteRecord():
    while(True):
        try:
            con=mysql.connector.connect(host="localhost",
                                         user="root",
                                         password="123456",
                                         database="Employee")
            cur=con.cursor()
            print("-"*50)
            #accept employee values from KBD
            empno=int(input("Enter Employee Number for deleting the
Record:"))
            #Prepare the query and execute
            cur.execute("delete from employee where eno=%d" %empno)
            con.commit()
            if(cur.rowcount>0):
                print("{} Record deleted
Successfully".format(cur.rowcount))
            else:
                print("Employee Record does not exist:")
            print("-"*50)
            ch=input("Do u want to delete another record(yes/no):")
        except mysql.connector.Error as e:
            print("Error occurred: ",e)

```

```

        if(ch.lower() == "no"):
            print("Thx for using this program")
            sys.exit()
        print("-"*50)
    except mysql.connector.DatabaseError as db:
        print("Problem in Database:",db)
    except ValueError:
        print("Don't enter alnums,strs and symbols for empno, salary")

#main program
deleteRecord()


---


#write a python program which will update name of the employee and give 50% hike for that
employee based on employee number
#RecordUpdateEx.py
import mysql.connector
def updateRecord():
    try:
        con=mysql.connector.connect(host="localhost",
                                     user="root",
                                     password="123456",
                                     database="Employee")
        cur=con.cursor()
        eno=int(input("Enter Employee Number for updating Name and Sal:"))
        empname=input("Enter Ur Correct Name:")
        uq="update employee set name=%s, sal=sal*(50/100) where"
        eno=%d"
        cur.execute(uq %(empname,eno))
        con.commit()
        if(cur.rowcount>0):
            print("{} Record updated Successfully in Employee
Table".format(cur.rowcount ))
        else:
            print("Employee Record does not exist:")
    except mysql.connector.DatabaseError as db:
        print("Problem in Database:",db)

#main program
updateRecord()


---


#Write a python program which will accept any table name and print all the records along
with column names.
#CompleteTable.py
import mysql.connector
def tablerecords():
    try:
        con=mysql.connector.connect(host="localhost",
                                     user="root",
                                     password="123456",
                                     database="Employee")
        cur=con.cursor()
        #accept the table name from keyboard
        tname=input("Enter table Name:")
        cur.execute("select * from "+tname)
        print("Table "+tname+" has "+str(cur.rowcount)+" rows")
        print("The columns are "+str(cur.column_names))
        for row in cur:
            print(row)
    except mysql.connector.DatabaseError as db:
        print("Problem in Database:",db)

```

```
cur.execute("select * from %s" %tname)
#Dipplay column Names
print("*"*80)
for cname in [colname[0] for colname in cur.description]:
    print("\t{}".format(cname),end="")
print()
print("*"*80)
#display records
records=cur.fetchall()
for record in records:
    for val in record:
        print("\t{}".format(val),end="")
    print()
print("*"*80)
except mysql.connector.DatabaseError as db:
    print("prob in DB:",db)
#main program
tablerecords()
```

Index Object Oriented Principles or Features or Concepts(10 days)

Index:

=>What are the advantages of OOPs

=>List of Object Oriented Principles

1. Classes
 2. Objects
 3. Data Encapsulation
 4. Data Abstraction
 5. Inheritance
 6. Polymorphism
 7. Message Passing(already)
-

1. Classes

=>Importance and purpose of Classes concept

=>Syntax for Defining Class

=>Types of Data Members

- a) Instance Data Members
- b) Class Level Data Members

=>Types of Methods

- a) Instance Method
- b) Class Level Method
- c) Static Method

=>What is "self" and "cls"

=>Programming Examples

2. Object

=>Importance and purpose of Object Concept

=>Syntax for creating Object

=>Programming Examples

=>PRAMMING eXamples related to pickling and Data base communication with Classes and objects.

=>Constructors in OOPs

=>Importance and purpose of Constructors

=>Types of Constructors

- a) Default Constructors
- b) Parameterized Constructors

=>Rules for Constructors

=>Programming Examples

=>Detstructrorts in OOPs with Garbage Collector

=>Importance and purpose of Detstructrorts

=>Syntax for defining Detstructrorts

=>Internal flow of Detstructrorts

=>relation between Detstructrorts and Garbage Collector

=>gc module
=>Programming Examples

3. Data Encapsulation and Data Abstraction
=>Importance and purpose of Data Encapsulation
=>Importance and purpose of Data Abstraction
=>Implementation of data encapsulation and Data Abstraction
=>Programming Examples

5. Inheritance
=>Importance and purpose of Inheritance
=>Types of Inheritances
 a) single
 b) multi level
 c) hierarchical
 d) multiple
 e) Hybrid
=>Syntax for Inheritance
=>Programming Examples

Method Overriding in OOPs
=>Importance and purpose of Method Overriding
=>Memory management in Method Overriding
=>Programming Examples

6. Polymorphism
=>Importance and purpose of Polymorphism
=>Difference between Polymorphism and Inheritance
=>Method Overriding with Polymorphism
=>super() and class name approaches in Polymorphism
=>Programming Examples

===== Object Oriented Principles or Features or Concepts =====

=>In real time, to develop any project or application, we must choose a language and it can satisfy two types of principles. They are

1. Functional(Procedure) Oriented Principles----C,Pascal,
cobol,8086,oracle7.3,PYTHON
2. Object Oriented Principles-----PYTHON C++ JAVA, .NET, from Oracle8
onwadrs.....

=>Even though, PYTHON programming Belongs to both Functional and Object Oriented Programming language and internally every thing is treated as object.

"Every Thing is an object " --Benefits
(OR)

Advantages of Object Oriented Principles

-
1. Objects allows us to store Large Volume of Data (Platform Independent)
 2. The Data is visiting between two machines in the form of Cipher Text (encrypted Format). So that we can achieve the Security
 3. The Large Volume of Data can be transferred between multiple machines all at once in the form of objects and obtains effective communication.
 4. With Objects we can build high effective Re-Usable Applications (Redundancy of the Code is Minimized).
 5. The Data is always available around Objects (Effective Memory Usage) and functions can operate on the objects.
-

List of Object Oriented Principles

=> To say a language is Object Oriented, It has to satisfy the following Principles.

1. Classes
 2. Objects
 3. Data Encapsulation
 4. Data Abstraction
 5. Inheritance
 6. Polymorphism
 7. Message Passing (already discussed)
-

Classes

- => The Purpose of Classes Concept is that "To Develop Programmer-Defined Data Type + To develop any real Time Application in OOPs"
- => The Purpose of Developing Programmer-Defined Data Type is that "To Store Customized Data and To Perform Customized Operations".
- => To Develop Programmer-Defined Data Type by using Classes concept, we use "class" keyword
- => In Python Programming, All Class Names are treated as Programmer-Defined Data Type.
- => Every Program in OOPs, Must starts with Classes concept.
-

Definition:

=> A Class is a collection Variables (Data Members) and Methods .

=> When we define a class, Memory space is not created for Data Members and Methods But Whose Memory Space is created when we create an object w.r.t Class Name.

Syntax for Defining a class in Python

```
class <class-name>:  
    Class Level Data Members  
        def InstanceMethodName(self,List of formal params if any):  
            -----  
            Specify Instance Data Members---Perform Specific Operations  
            -----  
  
        @classmethod  
        def classlevelname(cls,list of Formal params if any)  
            -----  
            Specify Class Level Data Members--Perform Common Operations  
            -----  
  
        @staticmethod  
        def staticmethodname(list of formal params if any):  
            -----  
            Performs Utility / Universal Operations  
            -----
```

Types of Data Members in a Class

=>In class of Python, we can define Two Types of Data Members. They are

1. Instance Data Members
 2. Class Level Data Members
-

1. Instance Data Members

=>Instance Data Members are those which are used for Storing Specific Values

=>Instance Data Members are those whose memory space created Each Every Time when an object is
 created.

=>Instance Data Members Must be Accessed w.r.t Object Name or self

 ObjName . InstanceData Member Name

 (OR)

 self.InstanceData Member Name

=>To Add the Instance Data Members to the corresponding object, we use 3 approaches.
They are

- a) By Using Object Name
 - b) By Using Instance Methods
 - c) By using Constructors.
-

2. Class Level Data Members

=>Class Level Data Members are those which are used for Common Values.

=>Class Level Data Members are those Whose Memory space created One time irrespective
 of Number of Objects are Created.

=>Class Level Data Members can be accessed w.r.t Class Name.

Class Name.Class Level Data Member Name

(OR)

cls.Class Level Data Member Name

(OR)

ObjectName.Class Level Data Member Name

(OR)

self.Class Level Data Member Name

=>To Add the Class Level Data Members we use 2 approaches. They are

a) Inside of Class Definition

b) Inside of Class Level Method.

#Program for storing sno,sname ,marks in an object of Programmer-defined class

#StudEx1.py----Instance Data members

class Student:pass

```
#main program
s1=Student()
s2=Student()
print("Content of s1 before adding the data={} and Number of
values={}".format(s1.__dict__, len(s1.__dict__)))
print("Content of s2 before adding the data={} and Number of
values={}".format(s2.__dict__, len(s2.__dict__)))
print("-"*50)
#add Instance Data Members to s1
s1.sno=10
s1.name="RS"
s1.marks=22.22
print("Content of s1 after adding the data={} and Number of values={}".format(s1.__dict__,
len(s1.__dict__)))
#add Instance Data Members to s2
s2.stno=20
s2.sname="TR"
s2.smarks=42.22
print("Content of s2 after adding the data={} and Number of values={}".format(s2.__dict__,
len(s2.__dict__)))
```

#Program for storing sno,sname ,marks in an object of Programmer-defined class

#StudEx2.py----Instance Data members and Class Level Data Members

class Student:

 crs="PYTHON" # here crs is called Class Level Data member

```
#main program
```

s1=Student()

s2=Student()

print("Content of s1 before adding=",s1.__dict__)

print("Content of s2 before adding=",s2.__dict__)

#Read Instance Data members such as sno,sname and marks to s1

print("-"*50)

```

print("First Student Information")
print("-"*50)
s1.sno=int(input("Enter Student Number:"))
s1.sname=input("Enter Student Name:")
s1.marks=float(input("Enter Student Marks:"))
print("-"*50)
print("Second Student Information")
print("-"*50)
s2.sno=int(input("Enter Student Number:"))
s2.sname=input("Enter Student Name:")
s2.marks=float(input("Enter Student Marks:"))
print("-"*50)
print("\nContent of s1 after adding=",s1.__dict__)
print("Content of s2 after adding=",s2.__dict__)


---


#Program for storing sno,sname ,marks in an object of Programmer-defined class
#StudEx3.py---Instance Data members and Class Level Data Members
class Student:
    crs="PYTHON" # here crs is called Class Level Data member

#main program
s1=Student()
s2=Student()
print("Content of s1 before adding=",s1.__dict__)
print("Content of s2 before adding=",s2.__dict__)
#Read Instance Data members such as sno,sname and marks to s1
print("-"*50)
print("First Student Information")
print("-"*50)
s1.sno=int(input("Enter Student Number:"))
s1.sname=input("Enter Student Name:")
s1.marks=float(input("Enter Student Marks:"))
print("-"*50)
print("Second Student Information")
print("-"*50)
s2.sno=int(input("Enter Student Number:"))
s2.sname=input("Enter Student Name:")
s2.marks=float(input("Enter Student Marks:"))

print("-"*50)
print("First Student Information:")
print("-"*50)
print("\tStudent Number:{} ".format(s1.sno))
print("\tStudent Name:{} ".format(s1.sname))
print("\tStudent Marks:{} ".format(s1.marks))
print("\tStudent Course:{} ".format(Student.crs))# accessing Class Level Data member w.r.t
Class Name
print("-"*50)
print("Second Student Information:")
print("-"*50)
print("\tStudent Number:{} ".format(s2.sno))

```

```
print("\tStudent Name:{} ".format(s2.sname))
print("\tStudent Marks:{} ".format(s2.marks))
print("\tStudent Course:{} ".format(Student.crs))## accessing Class Level Data member w.r.t
Class Name
print("-"*50)


---


#Program for storing sno,sname ,marks in an object of Programmer-defined class
#StudEx4.py---Instance Data members and Class Level Data Members
class Student:
    crs="PYTHON" # here crs is called Class Level Data member

#main program
s1=Student()
s2=Student()
print("Content of s1 before adding=",s1.__dict__)
print("Content of s2 before adding=",s2.__dict__)
#Read Instance Data members such as sno,sname and marks to s1
print("-"*50)
print("First Student Information")
print("-"*50)
s1.sno=int(input("Enter Student Number:"))
s1.sname=input("Enter Student Name:")
s1.marks=float(input("Enter Student Marks:"))
print("-"*50)
print("Second Student Information")
print("-"*50)
s2.sno=int(input("Enter Student Number:"))
s2.sname=input("Enter Student Name:")
s2.marks=float(input("Enter Student Marks:"))

print("-"*50)
print("First Student Information:")
print("-"*50)
print("\tStudent Number:{} ".format(s1.sno))
print("\tStudent Name:{} ".format(s1.sname))
print("\tStudent Marks:{} ".format(s1.marks))
print("\tStudent Course:{} ".format(s1.crs)) # accessing Class Level Data member w.r.t object
Name
print("-"*50)
print("Second Student Information:")
print("-"*50)
print("\tStudent Number:{} ".format(s2.sno))
print("\tStudent Name:{} ".format(s2.sname))
print("\tStudent Marks:{} ".format(s2.marks))
print("\tStudent Course:{} ".format(s2.crs))# accessing Class Level Data member w.r.t object
Name
print("-"*50)
```

Types of Methods in class of Python

=>In a class of Python, we can define 3 types of Methods. They are

1. Instance Methods
 2. Class Level Methods
 3. Static Methods.
-

1. Instance Methods

=>Instance Methods are used for performing Specific Operation or Operation on Instance Data

Members of Object and these Methods are also called Object Level Methods.

=>Instance Methods always Takes "self" as a First Formal Parameter.

=>The Syntax for Instance Method

```
def InstanceMethodName(self,list of formal params if any)
```

Specific Operations
Specify Instance Data Members

=>All Instance Methods Must be accessed w.r.t Object Name or self

ObjectName.InstanceMethod Name()
(OR)
self.InstanceMethod Name()

=>What is "self"?

=>"self" is one of the First Formal Parameter in Instance Methods

=>"self" contains Reference / Address of Current Class Object

=>"self" can be used Inside of Corresponding Instance Method Body Only But not
Possible to access other part of Program

2. Class Level Methods

=>Class Level Methods are used for performing Common Operation on Class Level Data Members.

=>In Order to define Class Level Methods, whose definition must be preceded with a pre-defined decorator called `@classmethod` and takes "cls" as a First Formal Parameter.

=>The Syntax for Class Level Method is

```
@classmethod  
def classlevelmethod(cls,list of formal params if any):
```

Specify Class Level Data Members
Common Operations

=>All Class Level Methods can be accessed w.r.t Class Name or cls or object name or self

```
classname.classlevelmethod()  
      (OR)  
cls.classlevelmethod()  
      (OR)  
objname.classlevelmethod()  
      (OR)  
self.classlevelmethod()
```

=>What is "cls"?

=>"cls" is one of the First Formal Parameter in Class Level Methods

=>"cls" contains Current Class name

=>"cls" can be used Inside of Corresponding Class Level Method Body Only But not Possible to access other part of Program

3. Static Methods

=>Static Methods are used for performing Universal Operations or Utility Operations

=>Static Methods definition must be preceded with a predefined decorator called

 @staticmethod and it never takes "cls" or "self" but always takes object of other classes.

=>The Syntax for Static Method is

```
@staticmethod  
def staticmethodname(list of Formal Params):
```

Utility Operation / Universal Operations

=>Static Methods can be accessed w.r.t Class Name or object name.

```
ClassName.static method name()  
      (OR)  
ObjectName.static method name()
```

#Program for Reading sno,sname ,marks in an object of Programmer-defined class w.r.t Instance Methods

#InstanceMethodEx1.py

class Student:

```
    def getstuddata(self):  
        print("-"*40)  
        self.sno=int(input("Enter Student Number:"))  
        self.sname=input("Enter Student Name:")  
        self.marks=float(input("Enter Student Marks:"))  
        print("-"*40)  
    def dispstuddata(hyd):  
        print("-"*40)  
        print("\tStudent Number:{} ".format(hyd.sno))  
        print("\tStudent Name:{} ".format(hyd.sname))
```

```

print("\tStudent Marks:{}\n".format(hyd.marks))
print("-"*40)

#main program
s1=Student()
s2=Student()
print("Content of s1 before reading=",s1.__dict__)
print("Content of s2 before reading=",s2.__dict__)
print("-"*50)
print("Enter First Student Information:")
s1.getstuddata()
print("Enter Second Student Information:")
s2.getstuddata()
print("Details of First Student:")
s1.dispstuddata()
print("Details of Second Student:")
s2.dispstuddata()


---


#Program for Reading sno,sname ,marks in an object of Programmer-defined class w.r.t
Instance Methods
#InstanceMethodEx2.py
class Student:
    crs="PYTHON" # Class Level Data Member
    def getstuddata(self):
        print("-"*40)
        self.sno=int(input("Enter Student Number:"))
        self.sname=input("Enter Student Name:")
        self.marks=float(input("Enter Student Marks:"))
        print("-"*40)
        self.dispstuddata() # calling Instance method from another instance
method

    def dispstuddata(self):
        print("-"*40)
        print("\tStudent Number:{}\n".format(self.sno))
        print("\tStudent Name:{}\n".format(self.sname))
        print("\tStudent Marks:{}\n".format(self.marks))
        print("\tStudent Course name:{}\n".format(self.crs))
        print("-"*40)

#main program
s1=Student()
s2=Student()
print("Content of s1 before reading=",s1.__dict__)
print("Content of s2 before reading=",s2.__dict__)
print("-"*50)
print("Enter First Student Information:")
s1.getstuddata()
print("Enter Second Student Information:")
s2.getstuddata()

```

```
#Program for demonstrating Class Level Methods and Class Level Data members
#ClassLevelMethodEx1.py
class Student:
```

```
    @classmethod
    def getcourse(cls):
        cls.crs="PYTHON"
    @classmethod
    def getDeveloper(cls):
        Student.dev="Rossum"
```

```
#main program
```

```
Student.getcourse() # calling Class Level method w.r.t class name
Student.getDeveloper() # calling Class Level method .w.r.t classname
s1=Student()
s2=Student()
print(s1.crs,s1.dev)
print(s2.crs,s2.dev)
```

```
#Program for demonstrating Class Level Methods and Class Level Data members
```

```
#ClassLevelMethodEx2.py
```

```
class Student:
```

```
    @classmethod
    def getcourse(cls):
        cls.crs="PYTHON"
    @classmethod
    def getDeveloper(cls):
        Student.dev="Rossum"
```

```
#main program
```

```
s1=Student()
s2=Student()
s1.getcourse() # calling Class Level method w.r.t object name
s2.getDeveloper() # calling Class Level method w.r.t object name
```

```
print(s1.crs,s1.dev)
print(s2.crs,s2.dev)
```

```
#Program for demonstrating Class Level Methods and Class Level Data members
```

```
#ClassLevelMethodEx3.py
```

```
class Student:
```

```
    @classmethod
    def getcourse(cls):
        cls.crs="PYTHON"
        cls.getDeveloper() # Calling Class Level Method w.r.t cls

    @classmethod
    def getDeveloper(cls):
        Student.dev="Rossum"
```

```
#main program
Student.getcourse() # Calling Class Level Method w.r.t Class Name
s1=Student()
s2=Student()

print(s1.crs,s1.dev)
print(s2.crs,s2.dev)
```

```
#Program for demonstrating Class Level Methods and Class Level Data members
```

```
#ClassLevelMethodEx4.py
```

```
class Student:
```

```
    @classmethod
```

```
    def getcourse(cls):
```

```
        cls.crs="PYTHON"
```

```
        cls.getDeveloper("ROSSUM") # Calling Class Level Method w.r.t cls
```

```
    @classmethod
```

```
    def getDeveloper(cls,dname):
```

```
        Student.dev=dname
```

```
    def getstudentdet(self,sno,sname,marks):
```

```
        self.sno=sno
```

```
        self.sname=sname
```

```
        self.marks=marks
```

```
    def dispstuddata(self):
```

```
        self.getcourse() # calling Class Level Method Name w.r.t self
```

```
        print("-"*40)
```

```
        print("\tStudent Number:{} ".format(self.sno))
```

```
        print("\tStudent Name:{} ".format(self.sname))
```

```
        print("\tStudent Marks:{} ".format(self.marks))
```

```
        print("\tStudent Course Name:{} ".format(self.crs))
```

```
        print("\tCourse Dev By:{} ".format(self.dev))
```

```
        print("-"*40)
```

```
#main program
```

```
s1=Student()
```

```
s2=Student()
```

```
s1.getstudentdet(10,"RS",11.11)
```

```
s2.getstudentdet(20,"TR",21.11)
```

```
s1.dispstuddata()
```

```
s2.dispstuddata()
```

```
#StaticMethodEx1.py
```

```
class Student:
```

```
    def getstuddet(self):
```

```
        self.sno=int(input("\nEnter Student Number:"))
```

```
        self.sname=input("Enter Student Name: ")
```

```
class Employee:
```

```
    def getempdet(self):
```

```

        self.eno=int(input("\nEnter Employee Number:"))
        self.ename=input("Enter Employee Name:")
        self.sal=input("Enter Employee Salary:")

class Teacher:
    def getteacherdet(self):
        self.tno=int(input("\nEnter Teacher Number:"))
        self.ename=input("Enter Teacher Name:")
        self.subject=input("Enter Teacher Subject:")

class Hyd:
    @staticmethod
    def dispobjectdata(kvr,pinfo):
        print("-"*50)
        print("Information about:{}{}".format(pinfo))
        for k,v in kvr.__dict__.items():
            print("\t{} {}".format(k,v))
        print("-"*50)

#main program
s=Student()
e=Employee()
t=Teacher()
s.getstuddet()
e.getempdet()
t.getteacherdet()
#calling Static Method w.r.t Class Name
Hyd.dispobjectdata(s,"Student")
Hyd.dispobjectdata(e,"Employee")
Hyd.dispobjectdata(t,"Teacher")


---


#StaticMethodEx2.py
class Student:
    def getstuddet(self):
        self.sno=int(input("\nEnter Student Number:"))
        self.sname=input("Enter Student Name:")

class Employee:
    def getempdet(self):
        self.eno=int(input("\nEnter Employee Number:"))
        self.ename=input("Enter Employee Name:")
        self.sal=input("Enter Employee Salary:")

class Teacher:
    def getteacherdet(self):
        self.tno=int(input("\nEnter Teacher Number:"))
        self.ename=input("Enter Teacher Name:")
        self.subject=input("Enter Teacher Subject:")

class Hyd:

```

```

@staticmethod
def dispobjectdata(kvr,pinfo):
    print("-"*50)
    print("Information about:{} ".format(pinfo))
    for k,v in kvr.__dict__.items():
        print("\t{} {}".format(k,v))
    print("-"*50)

#main program
s=Student()
e=Employee()
t=Teacher()
s.getstuddet()
e.getempdet()
t.getteacherdet()
#calling Static Method w.r.t Object Name
H=Hyd()
H.dispobjectdata(s,"Student")
H.dispobjectdata(e,"Employee")
H.dispobjectdata(t,"Teacher")


---


#StaticMethodEx3.py
class Student:
    def getstuddet(self):
        self.sno=int(input("\nEnter Student Number:"))
        self.sname=input("Enter Student Name:")
        self.dispobjectdata(self,"Student") # Calling Static Method w.r.t self
    @staticmethod
    def dispobjectdata(kvr,pinfo):
        print("-"*50)
        print("Information about:{} ".format(pinfo))
        for k,v in kvr.__dict__.items():
            print("\t{} {}".format(k,v))
        print("-"*50)

class Employee:
    def getempdet(self):
        self.eno=int(input("\nEnter Employee Number:"))
        self.ename=input("Enter Employee Name:")
        self.sal=input("Enter Employee Salary:")
        Student.dispobjectdata(self,"Employee") #Calling Static Method w.r.t
    class name
class Teacher:
    def getteacherdet(self):
        self.tno=int(input("\nEnter Teacher Number:"))
        self.ename=input("Enter Teacher Name:")
        self.subject=input("Enter Teacher Subject:")
        Student.dispobjectdata(self,"Teacher") #Calling Static Method w.r.t class
    name
#main program
s=Student()

```

```
e=Employee()
t=Teacher()
s.getstuddet()
e.getempdet()
t.getteacherdet()
```

```
"""write a python program which will accept student details such as student number student
name,marks in three subjects.
calculate the total marks
calculate the Percentage and give the grades.(fail/pass)
```

give the grade = #fail provided student secured less than 40 in any of the three subjects.

give the grade = #distinction provided student totall marks lies within 250 - 300

give the grade = #First class provided total marks lies within 200- 249

give the grade= second class provided total marks lies within 150- 199

give the grade= third class provided total marks lies within 120- 149

save the students result in the database."""

```
#StudentOOPsDataBase.py
```

```
import cx_Oracle
```

```
class Student:
```

```
    def getstuddet(self):
```

```
        self.sno=int(input("Enter Student Number:"))
```

```
        self.sname=input("Enter Student Name:")
```

```
        #validation of C Marks
```

```
        while(True):
```

```
            self.cm=int(input("Enter Marks in C(100):"))
```

```
            if(self.cm>=0) and (self.cm<=100):
```

```
                break
```

```
        #validation of C++ Marks
```

```
        while(True):
```

```
            self.cppm=int(input("Enter Marks in C++(100):"))
```

```
            if(self.cppm>=0) and (self.cppm<=100):
```

```
                break
```

```
        #validation of Python Marks
```

```
        while(True):
```

```
            self.pym=int(input("Enter Marks in PYTHON(100):"))
```

```
            if(self.pym>=0) and (self.pym<=100):
```

```
                break
```

```
    def compute(self):
```

```
        self.totmarks=self.cm+self.cppm+self.pym
```

```
        self.percent=(self.totmarks/300)*100
```

```
        #decide Grade
```

```
        if(self.cm<40) or (self.cppm<40) or (self.pym<40):
```

```
            self.grade="FAIL"
```

```
        else:
```

```
            if(self.totmarks>=250) and (self.totmarks<=300):
```

```
                self.grade="DISTINCTION"
```

```
            elif(self.totmarks>=200) and (self.totmarks<=249):
```

```
                self.grade="FIRST"
```

```
            elif(self.totmarks>=150) and (self.totmarks<=199):
```

```

        self.grade="SECOND"
    elif(self.totmarks>=120) and (self.totmarks<=149):
        self.grade="THIRD"

def savestuddata(self):
    #We must write PDBC Code
    con=cx_Oracle.connect("scott/tiger@localhost/orcl")
    cur=con.cursor()
    cur.execute("insert into result values(%d,'%s',%d,%d,%d,%d,%f,'%s')"
%(self.sno,self.sname,self.cm,self.cppm,self.pym,self.totmarks,self.percent,self.grade) )
    con.commit()
    print("Student Record Saved Successfully in Result Table:")

```

```

#main program
s=Student()
s.getstuddet()
s.compute()
s.savestuddata()

```

"""\nwrite a python program which will accept student details such as student number student name,marks in three subjects.\n calculate the total marks\n calculate the Percentage and give the grades.(fail/pass)

give the grade = #fail provided student secured less than 40 in any of the three subjects.
give the grade = #distinction provided student totall marks lies within 250 - 300
give the grade = #First class provided totall marks lies within 200- 249
give the grade= second class provided totall marks lies within 150- 199
give the grade= third class provided totall marks lies within 120- 149
save the students result in the database."""

```

#StudentOOPsDataBaseMySQL.py
import mysql.connector
class Student:
    def getstuddet(self):
        self.sno=int(input("Enter Student Number:"))
        self.sname=input("Enter Student Name:")
        #validation of C Marks
        while(True):
            self.cm=int(input("Enter Marks in C(100):"))
            if(self.cm>=0) and (self.cm<=100):
                break
        #validation of C++ Marks
        while(True):
            self.cppm=int(input("Enter Marks in C++(100):"))
            if(self.cppm>=0) and (self.cppm<=100):
                break
        #validation of Python Marks
        while(True):
            self.pym=int(input("Enter Marks in PYTHON(100):"))
            if(self.pym>=0) and (self.pym<=100):
                break

```

```

def compute(self):
    self.totmarks=self.cm+self.cppm+self.pym
    self.percent=(self.totmarks/300)*100
    #decide Grade
    if(self.cm<40) or (self.cppm<40) or (self.pym<40):
        self.grade="FAIL"
    else:
        if(self.totmarks>=250) and (self.totmarks<=300):
            self.grade="DISTINCTION"
        elif(self.totmarks>=200) and (self.totmarks<=249):
            self.grade="FIRST"
        elif(self.totmarks>=150) and (self.totmarks<=199):
            self.grade="SECOND"
        elif(self.totmarks>=120) and (self.totmarks<=149):
            self.grade="THIRD"

def savestuddata(self):
    #We must write PDBC Code
    try:
        con=mysql.connector.connect(host="localhost",
                                     user="root",
                                     password="123456",
                                     database="student")
        cur=con.cursor()
        cur.execute("insert into result
values(%d,'%s',%d,%d,%d,%f,%s')"
                    %(self.sno,self.sname,self.cm,self.cppm,self.pym,self.totmarks,self.percent,self.grade) )
        con.commit()
        print("Student Record Saved Successfully in Result Table:")
    except mysql.connector.DatabaseError as db:
        print("Prob in DB",db)

#main program
s=Student()
s.getstuddet()
s.compute()
s.savestuddata()


---


#Student.py-----File Name and acts as Module Name
import mysql.connector
class Student:
    def getstuddet(self):
        self.sno=int(input("Enter Student Number:"))
        self.sname=input("Enter Student Name:")
        #validation of C Marks
        while(True):
            self.cm=int(input("Enter Marks in C(100):"))
            if(self.cm>=0) and (self.cm<=100):
                break
        #validation of C++ Marks

```

```

while(True):
    self.cppm=int(input("Enter Marks in C++(100):"))
    if(self.cppm>=0) and (self.cppm<=100):
        break
#validation of Python Marks
while(True):
    self.pym=int(input("Enter Marks in PYTHON(100):"))
    if(self.pym>=0) and (self.pym<=100):
        break
def compute(self):
    self.totmarks=self.cm+self.cppm+self.pym
    self.percent=(self.totmarks/300)*100
    #decide Grade
    if(self.cm<40) or (self.cppm<40) or (self.pym<40):
        self.grade="FAIL"
    else:
        if(self.totmarks>=250) and (self.totmarks<=300):
            self.grade="DISTINCTION"
        elif(self.totmarks>=200) and (self.totmarks<=249):
            self.grade="FIRST"
        elif(self.totmarks>=150) and (self.totmarks<=199):
            self.grade="SECOND"
        elif(self.totmarks>=120) and (self.totmarks<=149):
            self.grade="THIRD"

def savestuddata(self):
    #We must write PDBC Code
    try:
        con=mysql.connector.connect(host="localhost",

```

```

        cur=con.cursor()
        cur.execute("insert into result
values(%d,%s,%d,%d,%d,%f,%s)")
        %(self.sno,self.sname,self.cm,self.cppm,self.pym,self.totmarks,self.percent,self.grade) )
        con.commit()
        print("Student Record Saved Successfully in Result Table:")
    except mysql.connector.DatabaseError as db:
        print("Prob in DB",db)

```

```

#StudDemo.py
from Student import Student
s=Student()
s.getstuddet()
s.compute()
s.savestuddata()

```

Index of Constructors in Class of Python

Index

- =>Purpose of Constructors
 - =>Definition of Constructors
 - =>Rules / Properties of Constructors
 - =>Types of Constructors
 - a) Default Constructors
 - b) Parameterized Constructors
 - =>Programming Examples
-

Constructors in Python

- =>The purpose of Constructors in Python is that "To Initialize the object".
 - =>Initializing the object is nothing but placing our own values in the object without leaving an object empty.
-

=>Definition of Constructor:

-
-
- =>A Constructor is a special Method which is automatically or implicitly called by PVM during object creation and whose purpose is to initialize the object without leaving an object empty.
-

Syntax for Defining Constructor:

```
def __init__(self, list of formal params if any):
```

```
Block of Statements--Initialization
```

Rules or Properties of Constructor:

- 1. The Name of the constructor is def __init__(self)
 - 2. The Constructors automatically or implicitly called by PVM during object creation
 - 3. Constructors should not return any value (It can return only None value)
 - 4. Constructors participates in Inheritance Process.
 - 5. Constructors can be Overridden (can re-defined)
-

Types Constructors in Python

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

1. Default or Parameter-Less Constructor
 2. Parameterized Constructor
-

1. Default or Parameter-Less Constructor

=>A Constructor is said to be Default iff it never takes any argument(s) or Formal Param(s)

=>The purpose of Default or Parameter-Less Constructor is that "To Initialize multiple objects of same class with same values".

=>Syntax:

```
def __init__(self):  
    _____  
        Block of Stmts--Initialization  
    _____
```

=>Examples

```
#DefConstEx1.py  
class Test:  
    def __init__(self):  
        print("i am from Default Constructor:")  
        self.a=10  
        self.b=20  
        print("Value of a: {}".format(self.a))  
        print("Value of b: {}".format(self.b))  
  
#main program  
t1=Test()  
t2=Test()  
t3=Test()
```

2. Parameterized Constructor

=>A Constructor is said to be Parameterized iff it always takes any argument(s) or Formal Param(s)

=>The purpose of Parameterized Constructor is that "To Initialize multiple objects of same class with Different values".

=>Syntax:

```
def __init__(self, list of formal params):  
    _____  
        Block of Stmts--Initialization  
    _____
```

Examples:

```
#ParamConstEx1.py
class Test:
    def __init__(self,a,b):
        print("i am from Parameterized Constructor:")
        self.a=a
        self.b=b
        print("Value of a:{}".format(self.a))
        print("Value of b:{}".format(self.b))
```

```
#main program
t1=Test(10,20)
t2=Test(100,200)
t3=Test("RS","PYTHON")
```

Note:

Note: In Class of Python, we can't define both default and Parameterized constructors bcoz PVM can remember only latest constructor (due to its interpretation Process) . To full fill the need of both default and parameterized constructors , we define single constructor with default parameter mechanism.

Examples:

```
#DefultParamConstEx1.py
class Test:
    def __init__(self,a=1,b=2):
        print("i am from Default /Parameterized Constructor:")
        self.a=a
        self.b=b
        print("Value of a:{}".format(self.a))
        print("Value of b:{}".format(self.b))
```

```
#main program
t1=Test() # Object Creation calls Default Constructor
t2=Test(100,200) # Object Creation calls Parameterized Constructor
```

#Program for demostrating the need of Constructor

```
#Non-ConstEx1.py
class Student:
    def getstudvalues(self):
        self.sno=10
        self.sname="Rossum"
```

```
#main program
s=Student() # Object Creation
```

```
print("Initial Content of s=",s.__dict__) # {}
#To place the data inside of object s, we must call getstudvalues() explicitly
s.getstudvalues()
print("Content of s after calling method=",s.__dict__) # {}


---


#Program for demonstrating the need of Constructor
#ConstEx1.py
class Student:
    def __init__(self): # default or Parameterless constructor
        self.sno=10
        self.name="Rossum"
```

```
#main program
s=Student() # Object Creation---PVM Calls implicitly constructor
print("Initial Content of s=",s.__dict__) # {}
s1=Student() # Object Creation---PVM Calls implicitly constructor
print("Initial Content of s1=",s1.__dict__) # {}
s2=Student() # Object Creation---PVM Calls implicitly constructor
print("Initial Content of s2=",s2.__dict__)


---


#Program for demonstrating the need of Constructor
#ConstEx2.py
class Student:
    def __init__(self,sno,sname): # Parametrized Constructor
        self.sno=sno
        self.sname=sname
```

```
#main program
s1=Student(10,"Rossum") # Object Creation---PVM Calls implicitly constructor
print("Initial Content of s1=",s1.__dict__) # {}
s2=Student(20,"Travis") # Object Creation---PVM Calls implicitly constructor
print("Initial Content of s2=",s2.__dict__)
s3=Student(30,"Kinney") # Object Creation---PVM Calls implicitly constructor
print("Initial Content of s3=",s3.__dict__)


---


```

```
#DefaultConstEx1.py
class Test:
    def __init__(self):
        print("\nI am from default constructor")
        self.a=10
        self.b=20
        print("Val of a=",self.a)
        print("Val of b=",self.b)
```

```
#main program
t1=Test() # Object Creating-----calls default constructor
t2=Test() # Object Creating-----calls default constructor
t3=Test() # Object Creating-----calls default constructor


---


```

```

#ParamConstEx1.py
class Test:
    def __init__(self,a,b):
        print("\nI am from Parameterized constructor")
        self.a=a
        self.b=b
        print("Val of a=",self.a)
        print("Val of b=",self.b)

#main program
t1=Test(10,20) # Object Creating----calls Parameterized constructor
t2=Test(100,200) # Object Creating-----callsParameterized constructor
t3=Test(1000,2000) # Object Creating-----calls Parameterized constructor


---


#DefaultParamConstEx1.py
class Test:
    def __init__(self,a=1,b=2):
        print("\nI am from default / Parameterized constructor")
        self.a=a
        self.b=b
        print("Val of a=",self.a)
        print("Val of b=",self.b)

#main program
t1=Test() # Object Creating----calls Default constructor
t2=Test(10,20) # Object Creating-----calls Parameterized constructor
t3=Test(100)# Object Creating-----calls Parameterized constructor
t3=Test(b=100)# Object Creating-----calls Parameterized constructor
t4=Test(b=100,a=200)# Object Creating-----calls Parameterized constructor
t5=Test(b=100,a="KVR")# Object Creating-----calls Parameterized constructor
t6=Test("Python",b="Java")# Object Creating-----calls Parameterized constructor


---


#write a python program which will read student values such as sno,sname and marks. save the details of the student in a file by using pickling and read the student record values from the file by using unpickling.impliment this example by using classes and objects.
#Student.py--File Name and Module Name
class Student:
    def __init__(self,sno,sname,marks):
        self.sno=sno
        self.sname=sname
        self.marks=marks
    def dispstuddata(self):
        print("\t{}\t{}\t{}".format(self.sno,self.sname,self.marks))


---


#StudOopsPickEx.py
import sys,pickle
from Student import Student
class StudPick:
    def savestuddata(self):
        with open("oopsstud.data","ab") as fp:
            while(True):

```

```

        print("-"*50)
        sno=int(input("Enter Student Number:"))
        sname=input("Enter Student Name:")
        marks=float(input("Enter Student Marks:"))
        s=Student(sno,sname,marks) # Calling PC of Student
Class
        pickle.dump(s,fp)
        print("Student Data Saved in a File:")
        print("-"*50)
        ch=input("Do u want to Insert another
record(yes/no):")
        if(ch.lower() == "no"):
            print("Thx for using this program")
            print("-"*50)
            sys.exit()

#main Program
sp=StudPick()
sp.savestuddata()


---


#StudOopsUnPickEx.py
import pickle
class StudUnPick:
    def readrecords(self):
        with open("oopsstud.data","rb") as fp:
            print("-"*50)
            print("\tSNO\tNAME\tMARKS")
            print("-"*50)
            while(True):
                try:
                    obj=pickle.load(fp)
                    obj.dispstuddata()
                except EOFError:
                    print("-"*50)
                    break

#main program
so=StudUnPick()
so.readrecords()


---


=====
```

Destructors in Python and Garbage Collector

- =>We know that Garbage Collector is one of the in-built program in python, which is running behind of every python program and whose role is to collect un-used memory space and it improves the performance of python based applications.
- =>Every Garbage Collector Program is internally calling its Own Destructor Functions.
- =>The destructor function name in python is `def __del__(self)`.

=>By default ,The destructor always called by Garbage Collector when the program execution completed for de-allocating the memory space of objects which are used in that program. Where as constructor called By PVM implicitly when object is created for initlizing the object.

=>When the program execution is completed, GC calls its own destructor to de-allocate the memory space of objects present in program and it is called automatic Garbage Collection.

=>Hence , We have THREE programming conditions for calling GC and to make the garbage collector to call destructor Function.

a) By default (or) automatically GC calls destructor, when the program execution completed.

b) Make the object reference as None for calling Forcefull Garbage Collection
Syntax : objname=None

c) delete the object by using del operator for calling Forcefull Garbage Collection
Syntax:- del objname

=>Syntax:

```
def __del__(self):
```

=>No Need to write destructor in class of Python bcoz GC contains its own Destructor

Garbage Collector

=>Garbage Collector contains a pre-defined module called "gc"

=>Here gc contains the following Functions.

- 1) isenabled()
- 2) enable()
- 3) disable()

=>GC is not under control of Programmer but it always maintained and mangaged by OS and PVM.

NOTE: Python Programmers need not to write destructor method / function and need not to deal with Garbage Collection Process by using gc module bcoz PVM and OS takes care about Automatic Garbage Collection Process.

=====x=====

#Non-DestEx.py

class Student:

```
def __init__(self,sno,sname):
    self.sno=sno
    self.sname=sname
    print("\t{}\t{}".format(self.sno,self.sname))
```

```
#main program
print("\nProgram Execution Started")
s1=Student(10,"RS")# Object Creation
s2=Student(20,"TR")# Object Creation
print("Program Execution Ended")


---


#DestEx1.py
import sys
class Student:
    def __init__(self,sno,sname):
        print("I am from PC")
        self.sno=sno
        self.sname=sname
        print("\t{}\t{}".format(self.sno,self.sname))

    def __del__(self):
        global totmem
        print("GC calls __del__")
        print("At Present Memory Space:{} ".format(totmem))
        print("Now Memory space:",sys.getsizeof(self))
        totmem=totmem-sys.getsizeof(self)
        print("Remaing Memory Space:{} ".format(totmem))
```

```
#main program
print("\nProgram Execution Started")
s1=Student(10,"RS")# Object Creation
s2=Student(20,"TR")# Object Creation
totmem=sys.getsizeof(s1)+sys.getsizeof(s2)
print("Now Memory space in main program:{} ,totmem")
print("Program Execution Ended")


---


```

```
#DestEx2.py
import time
class Student:
    def __init__(self,sno,sname):
        print("I am from PC")
        self.sno=sno
        self.sname=sname
        print("\t{}\t{}".format(self.sno,self.sname))

    def __del__(self):
        print("GC calls __del__")
```

```
#main program
print("\nProgram Execution Started")
s1=Student(10,"RS")# Object Creation
print("Now we are No Longer interested in maintaing S1 object memory space:")
time.sleep(5)
s1=None # Calling GC Forcefully and it inturns calls Destructor
time.sleep(5)
s2=Student(20,"TR")# Object Creation
```

```
print("Program Execution Ended")
time.sleep(5)


---


#DestEx3.py
import time
class Student:
    def __init__(self,sno,sname):
        print("I am from PC")
        self.sno=sno
        self.sname=sname
        print("\t{}\t{}".format(self.sno,self.sname))

    def __del__(self):
        print("GC calls __del__")

#main program
print("\nProgram Execution Started")
s1=Student(10,"RS")# Object Creation
print("Now we are No Longer interested in maintaining S1 object memory space:")
time.sleep(5)
s1=None # Calling GC Forcefully and it inturns calls Destructor
time.sleep(5)
s2=Student(20,"TR")# Object Creation
print("Now we are No Longer interested in maintaining S2 object memory space:")
time.sleep(5)
s2=None # Calling GC Forcefully and it inturns calls Destructor
print("Program Execution Ended")


---


#DestEx4.py
import time
class Student:
    def __init__(self,sno,sname):
        print("I am from PC")
        self.sno=sno
        self.sname=sname
        print("\t{}\t{}".format(self.sno,self.sname))

    def __del__(self):
        print("GC calls __del__")

#main program
print("\nProgram Execution Started")
s1=Student(10,"RS")# Object Creation
s3=Student(30,"KVR") # Object Creation
print("Now we are No Longer interested in maintaining S1 object memory space:")
time.sleep(5)
del s1 # Calling GC Forcefully and it inturns calls Destructor
time.sleep(5)
s2=Student(20,"TR")# Object Creation
print("Now we are No Longer interested in maintaining S2 object memory space:")
time.sleep(5)
del s2 # Calling GC Forcefully and it inturns calls Destructor
```

```
print("Program Execution Ended")
time.sleep(5)


---


#DestEx5.py
import time
class Student:
    def __init__(self,sno,sname):
        print("I am from PC")
        self.sno=sno
        self.sname=sname
        print("\t{}\t{}".format(self.sno,self.sname))

    def __del__(self):
        print("GC calls __del__")

#main program
print("\nProgram Execution Started")
s1=Student(10,"RS")# Object Creation
s2=s1 #Deep Copy
s3=s1 #Deep Copy
print(id(s1),id(s2),id(s3))
time.sleep(5)
print("\nProgram Execution Ended")
time.sleep(5)
#Here GC calls __del__(self) only once even we have 3 object and they have same address..


---


#DestEx6.py
import time
class Student:
    def __init__(self,sno,sname):
        print("I am from PC")
        self.sno=sno
        self.sname=sname
        #print("\t{}\t{}".format(self.sno,self.sname))

    def __del__(self):
        print("GC calls __del__")

#main program
print("\nProgram Execution Started")
s1=Student(10,"RS")# Object Creation
s2=s1 #Deep Copy
s3=s1 #Deep Copy
print("Now we are No Longer interested in maintaining S1 object memory space:")
time.sleep(5)
s1=None # GC will not call __del__(self) bcoz still s2 and s3 to same memory space
print("Now we are No Longer interested in maintaining S2 object memory space:")
time.sleep(5)
del s2 # GC will not call __del__(self) bcoz still s3 to same memory space
print("\nProgram Execution Ended")
time.sleep(5)
#Here GC calls __del__(self)
```

```

#DestEx7.py
import time
class Student:
    def __init__(self,sno,sname):
        print("I am from PC")
        self.sno=sno
        self.sname=sname
        #print("\t{}\t{}".format(self.sno,self.sname))

    def __del__(self):
        print("GC calls __del__")

#main program
print("\nProgram Execution Started")
s1=Student(10,"RS")# Object Creation
s2=s1 #Deep Copy
s3=s1 #Deep Copy
print("Now we are No Longer interested in maintaining S1 object memory space:")
time.sleep(5)
s1=None # GC will not call __del__(self) bcoz still s2 and s3 to same memory space
print("Now we are No Longer interested in maintaining S2 object memory space:")
time.sleep(5)
del s2 # GC will not call __del__(self) bcoz still s3 to same memory space
print("Now we are No Longer interested in maintaining S3 object memory space:")
time.sleep(5)
del s3 # GC will not call __del__(self) bcoz still s3 to same memory space
print("\nProgram Execution Ended")
time.sleep(5)
#Here GC will not call __del__(self)


---


#gce1.py
import gc
print("Line-3-->Is GC Running:",gc.isenabled())
print("\nThis Python class")
print("Destructor Topic is Going on")
gc.disable()
print("Line-5-->Is GC Running:",gc.isenabled())
print("Python is an OOP alng")
gc.enable()
print("Line-10-->Is GC Running:",gc.isenabled())
print("Python is also fun Prog")


---


#gce2.py
import time,gc
class Student:
    def __init__(self,sno,sname):
        print("I am from PC")
        self.sno=sno
        self.sname=sname
        print("\t{}\t{}".format(self.sno,self.sname))

    def __del__(self):

```

```
print("GC calls __del__")

#main program
print("\nProgram Execution Started")
print("Line-15-->Is GC is RUNNING:",gc.isenabled())
s1=Student(10,"RS")# Object Creation
print("\nNow we are No Longer interested in maintaining S1 object memory space:")
time.sleep(5)
s1=None # Calling GC Forcefully and it inturns calls Destructor
time.sleep(5)
gc.disable()
print("\nLine-22-->Is GC is RUNNING:",gc.isenabled())
s2=Student(20,"TR")# Object Creation
print("\nNow we are No Longer interested in maintaining S2 object memory space:")
time.sleep(5)
s2=None
print("Program Execution Ended")
```

===== objects in Python =====

- =>When we define a class, memory space is not created for Data Members and Methods but whose memory is created when we create an object w.r.t class name.
- =>To Store the data and to do any Data Processing, It is mandatory to create an object.
- =>To create an object, there must exists a class Definition otherwise we get NameError.

Definition of object:

=>Instance of a class is called object (Instance is nothing but allocating sufficient memory space for the Data Members and Methods of a class).

Syntax for creating an object

varname=classname()
(or)
varname=classname(Val1,Val2...val-n)

Examples: create an object of Student

so=Student()

Example:- create an object Employee

eo=Employee(10,"Rossum")

Differences Between Classes and Objects

Class:

- 1) A class is a collection of Data Members and Methods

- 2) When we define a class, memory space is not created for Data Members and Methods and it can be treated as specification / model for real time application.
 - 3) Definition of a particular exists only once
 - 4) When we develop any Program with OOPs principles, Class Definition Loaded First in main memory only once.
-

Objects:

- 1) Instance of a class is called Object
 - 2) When we create an object, we get the memory space for Data members and Methods of Class.
 - 3) w.r.t One class Definition, we can create multiple objects.
 - 4) we can create an object after loading the class definition otherwise we get NameError
-

X

Inheritance

- =>Inheritance is one of distinct features of OOPs
- =>The purpose of Inheritance is that " To build Re-usable Applications in Python Object Oriented Programming".
-
- =>Definition of Inheritance:
-
- =>The Process obtaining Data members , Methods and Constructors (Features) of one class into another class is called Inheritance.
- =>The class which is giving Data members , Methods and Constructors (Features) is called Super or Base or Parent Class.
- =>The Class which is taking Data members , Methods and Constructors (Features) is called Sub or Derived or Child Class.
- =>The Inheritance concept always follows Logical (Virtual) Memory Management. This Memory Management says that " Neither we write Source Code nor Takes Physical Memory Space ".
-

Advantages of Inheritance:

- =>When we develop any inheritance based application, we get the following advantages.
1. Application Development Time is Less
 2. Application Memory Space is Less
 3. Application Execution time is Fast / Less
 4. Application Performance is enhanced (Improved)
 5. Redundancy (Duplication) of the code is minimized.
-

Types of Inheritance

- =>A Type of Inheritance is a model / Paradigm , which makes us to understand how the features are inherited from Base Class into Derived Class.
- =>In Python Programming, we have 5 types of Inheritances. They are

1. Single Inheritance
 2. Multi Level Inheritance
 3. Hierarchical Inheritance
 4. Multiple Inheritance
 5. Hybrid Inheritance
-

1) Single Inheritance

Definition:

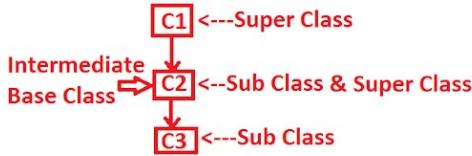
This Inheritance contains Single Base class and Single Derived Class

Diagram:



2. Multi Level Inheritance

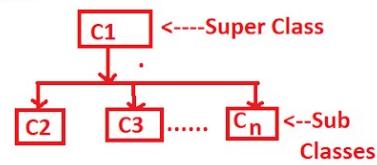
Definition: This Inheritance contains single base class , single derived class and Intermediate base class(es).



3. Hierarchical Inheritance

Definition: This Inheritance Contains Single Super Class and Multiple Sub Classes

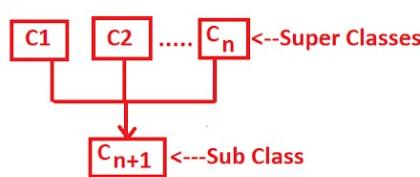
Diagram:



4. Multiple Inheritance

Definition: This Inheritance contains multiple Super Classes and single sub class.

Diagram:



5. Hybrid Inheritance:

Definition:

Hybrid Inheritance= Combination of any available Inheritance Types.

Diagram1:

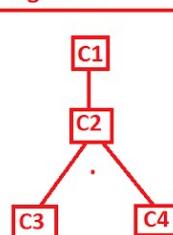
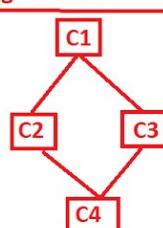


Diagram2:



Inheritaing the Features of Base Class Into Derived Class

Syntax:

```
-----  
class <classname-1>:  
-----  
-----  
  
class <classname-2>:  
-----  
-----  
-----  
  
class <classname-n>:  
-----  
-----  
  
class <classname-n+1>(<classname-1>,<classname-2>,...<classname-n>):  
-----  
-----  
-----
```

Explanation:

- =>Here <classname-1>,<classname-2>,...<classname-n> are called Base Classes / Super Classes
- =>here <classname-n+1> is called Derived Class.
- =>Here The features of <classname-1>,<classname-2>,...<classname-n> are Inherited into <classname-n+1> and these Features are available logically In <classname-n+1> and we can access them w.r.t Object Name (OR) Self (Instance) /Class Name (Class level and Static).
- =>When we develop any Inheritance Application, we are always recommeding to create an object of Bottom Most Derived Class bcoz It inherits the features of Intermediate Base Class(es) and Base class.
- =>For Every Class in Python, There exist a pre-defined implicit super class called "object" bcoz It provides Garbage Collection Facility to its sub classes.

#Program for demonstarting Inheritance
#InhProg1.py

```
class C1:  
    def setA(self):  
        self.a=10  
  
class C2(C1): # Single Inheritance C-- is called Base Class and C2 is called Derived Class  
    def setB(self):  
        self.b=20  
    def disp(self):  
        print("Val of a(C1-BC):{}".format(self.a))
```

```

print("Val of b(C2-DC):{}".format(self.b))

#main program
o2=C2()
print("content of o2:",o2.__dict__)
o2.setB()
print("content of o2:",o2.__dict__)
o2.setA()
print("content of o2:",o2.__dict__)
o2.disp()


---


#Program for demonstarting Inheritance
#InhProg2.py
class C1:
    def setA(self):
        self.a=10

class C2(C1): # Single Inheritance C-- is called Base Class and C2 is called Derived Class
    def setB(self):
        self.b=20
        self.setA() # calling Base Class Method from derived class method
        self.disp() # calling Current Class Method from other method of current
class
    def disp(self):
        print("Val of a(C1-BC):{}".format(self.a))
        print("Val of b(C2-DC):{}".format(self.b))

#main program
o2=C2()
print("content of o2:",o2.__dict__)
o2.setB()


---


#Program for demonstarting Inheritance
#InhProg3.py
class GrandParent:
    def getgpprop(self):
        self.gpp=3.4
class Parent(GrandParent):
    def getpprop(self):
        self.pp=13.4
class Child(Parent):
    def childprop(self):
        self.cprop=4.5
    def totalproperty(self):
        prop=self.gpp+self.pp+self.cprop
        print("-"*50)
        print("\tProperties of Child:")
        print("-"*50)
        print("\tGrand Parent Properties:{}".format(self.gpp))
        print("\tParent Properties:{}".format(self.pp))
        print("\tChild Properties:{}".format(self.cprop))
        print("-"*50)

```

```

        print("\tTotal Property:{}\n".format(prop))
        print("-"*50)

#main program
co=Child()
co.childprop()
co.getpprop()
co.getgpprop()
co.totalproperty()


---


#Program for demonstarting Inheritance
#InhProg4.py
class GrandParent:
    def getgpprop(self):
        self.gpp=3.4
class Parent(GrandParent):
    def getpprop(self):
        self.pp=13.4
class Child(Parent):
    def childprop(self):
        self.cprop=4.5
    def totalproperty(self):
        self.getgpprop()
        self.getpprop()
        self.childprop()
        prop=self.gpp+self.pp+self.cprop
        print("-"*50)
        print("\tProperties of Child:")
        print("-"*50)
        print("\tGrand Parent Properties:{}\n".format(self.gpp))
        print("\tParent Properties:{}\n".format(self.pp))
        print("\tChild Properties:{}\n".format(self.cprop))
        print("-"*50)
        print("\tTotal Property:{}\n".format(prop))
        print("-"*50)

```

```

#main program
co=Child()
co.totalproperty()


---


#Program for demonstarting Inheritance
#InhProg5.py
class GrandParent:
    def getgpprop(self):
        self.gpp=float(input("Enter Grand Parent Properties:"))
class Parent(GrandParent):
    def getpprop(self):
        self.pp=float(input("Enter Parent Properties:"))
class Child(Parent):
    def childprop(self):

```

```

        self.cprop=float(input("Enter Child Properties:"))
def totalproperty(self):
    self.getgpprop()
    self.getpprop()
    self.childprop()
    prop=self.gpp+self.pp+self.cprop
    print("-"*50)
    print("\tProperties of Child:")
    print("-"*50)
    print("\tGrand Parent Properties:{}".format(self.gpp))
    print("\tParent Properties:{}".format(self.pp))
    print("\tChild Properties:{}".format(self.cprop))
    print("-"*50)
    print("\tTotal Property:{}".format(prop))
    print("-"*50)

#main program
co=Child()
co.totalproperty()


---


#Program for demonstarting Inheritance
#InhProg6.py
class Father:
    def getfatherprop(self):
        self.fp=float(input("Enter Father Properties:"))
        return self.fp

class Mother:
    def getmotherprop(self):
        self.mp=float(input("Enter Mother Properties:"))
        return self.mp

class Child(Mother,Father):
    def totalprop(self):
        fp=self.getfatherprop()
        mp=self.getmotherprop()
        totprop=fp+mp
        print("\tProperties of Child:")
        print("-"*50)
        print("\tFather Properties:{}".format(fp))
        print("\tMother Properties:{}".format(mp))
        print("\tChild Total Properties:{}".format(totprop))
        print("-"*50)

#main program
c=Child()
c.totalprop()

```

Data Encapsulation and Data Abstraction

Data Encapsulation:

- =>The Process of Hiding the confidential Information / Data / Methods from external Programmers / end users is called Data Encapsulation.
- =>The Purpose of Encapsulation concept is that "To Hide Confidential Information / Features of Class (Data Members and Methods)".
- =>Data Encapsulation can be applied in two levels. They are
 - a) At Data Members Level
 - b) At Methods Level

=>To implement Data Encapsulation in python programming, The Data Members , Methods must be preceded with double under score (__)

Syntax1:- (Data member Lavel)

```
class <ClassName>:  
    def methodname(self):  
        self.__Data MemberName1=Value1  
        self.__Data MemberName2=Value2  
-----  
-----  
        self.__Data MemberName-  
n=Value-n  
(OR)
```

Syntax1:- (Data member Lavel)

```
class <ClassName>:  
    def __init__(self):  
        self.__Data MemberName1=Value1  
        self.__Data MemberName2=Value2  
-----  
-----  
        self.__Data MemberName-  
n=Value-n
```

Syntax2:- (Method Level)

```
class <ClassName>:  
    def __methodname(self):  
        self.Data MemberName1=Value1  
        self.Data MemberName2=Value2  
-----  
-----  
        self.Data MemberName-n=Value-n
```

Examples: Refer Acc1.py, Acc2.py, Acc3.py and Acc4.py Programs

Data Abstraction:

=>The Process of retrieving / extracting Essential Details without considering Hidden Details is called Data Abstraction.

Examples: Others1.py Others2.py
 Others3.py Others4.py Programs

Note:- We can't apply Data Encapsulation on Constructors in Python but whose Initialized Data Members can be encapsulated.

#Acc1.py----File Name and Module Name

class Account:

```
def __init__(self):
    self.__acno=10
    self.cname="Rossum"
    self.__bal=34
    self.__pin=1234
    self.bname="SBI"
```

#Acc2.py----File Name and Module Name

class Account:

```
def getaccdet(self):
    self.__acno=10
    self.cname="Rossum"
    self.__bal=34
    self.__pin=1234
    self.bname="SBI"
```

#Acc3.py----File Name and Module Name

class Account:

```
def __getaccdet(self):
    self.acno=10
    self.cname="Rossum"
    self.bal=34
    self.pin=1234
    self.bname="SBI"
```

#Acc4.py----File Name and Module Name

class Account:

```
def _____init__(self):
    self.acno=10
    self.cname="Rossum"
    self.bal=34
    self.pin=1234
    self.bname="SBI"
```

```

#Others1.py
from Acc1 import Account
ac=Account() # Object Creation
#print("\nAccount Number:{} ".format(ac.acno))
print("Account Name:{} ".format(ac.cname))
#print("Account Balance:{} ".format(ac.bal))
#print("Account Pin:{} ".format(ac.pin))
print("Account Branch Name:{} ".format(ac.bname))


---


#Others2.py
from Acc2 import Account
ac=Account() # Object Creation
ac.getaccdet()
#print("\nAccount Number:{} ".format(ac.acno))
print("Account Name:{} ".format(ac.cname))
#print("Account Balance:{} ".format(ac.bal))
#print("Account Pin:{} ".format(ac.pin))
print("Account Branch Name:{} ".format(ac.bname))


---


#Others3.py
from Acc3 import Account
ac=Account() # Object Creation
print(ac.__dict__)
ac.getaccdet()


---


#Others4.py
from Acc4 import Account
ac=Account() # Object Creation
ac.__init__()
print("\nAccount Number:{} ".format(ac.acno))
print("Account Name:{} ".format(ac.cname))
print("Account Balance:{} ".format(ac.bal))
print("Account Pin:{} ".format(ac.pin))
print("Account Branch Name:{} ".format(ac.bname))

```

===== Polymorphism in Python =====

- =>Polymorphism is one of the distinct features of OOPs
- =>The purpose of Polymorphism is that "Efficient Utilization Memory Space (OR) Less Memory space is achieved".
-
- =>Def. of Polymorphism:
-
- =>The Process of Representing "One Form in multiple Forms " is called Polymorphism.
-
- =>The Polymorphism Principle is implemented(Bring into action) by Using "Method Overriding" feature of all OO Programming Languages.
- =>In The definition of polymorphism, "One Form" represents "Original Method" and multiple forms represents Overridden Methods.

=>A "Form" is nothing but existence of a Method. if the method is existing in base class then it is called "Original Method(one form)" and if the method existing in derived class(es) then it is called "Overridden Method(multiple Forms)".

Method Overriding in Python

=>Method Overriding=Method Heading is same + Method Body is Different
(OR)

=>The process of re-defining the original method of base class into various derived classes for performing different operations is called Method Overriding.

=>To use Method Overriding in python program we must apply Inheritance Principle.

=>Method Overriding used for implementing Polymorphism Principle.

(PLOYMORPHISM<----METHOD OVERRIDING<----INHERITANCE<----CLASS AND OBJECTS)

Examples:

#methodoverex1.py

```
class Circle:  
    def draw(self): # original Method  
        print("Drawing Circle")  
  
class Rect(Circle):  
    def draw(self): # overridden Method  
        print("Drawing Rect:")  
        super().draw()  
  
class Square(Rect):  
    def draw(self): # overridden Method  
        print("Drawing Square:")  
        super().draw()
```

#main program

```
so=Square()  
so.draw()
```

#teacher.py

```
class Teacher:  
    def readsub(self):  
        print("Teacher advises to read 2 hours")
```

class LazyStudent(Teacher):

```
    def readsub(self):  
        print("LazyStudent never read at all")
```

class PerfectStudent(Teacher):

```
    def readsub(self):  
        print(" Perfect Student 2hrs reading and practicing")
```

ls=LazyStudent()

```
ls.readsub()  
ps=PerfectStudent()  
ps.readsub()
```

Number of approaches to call original methods / constructors from
Overridden methods / Constructors

=>We have two approaches to call original method / constructors of base class from
overridden method / constructors of derived class. They are

- 1) By using super()
 - 2) By using Class Name
-

1) By using super():

=>super() is one of the pre-defined function, which is used for calling super class original
method / constructor from overridden method / constructors of derived class.

Syntax1:- super().methodname(list of values if any)

Syntax2:- super().__init__(list of values if any)

=>with super() we are able to call only immediate base class method but unable to call
Specified method of base Class . To do this we must use class name approach.

2) By using Class Name:

=>By using ClassName approach, we can call any base class method / constructor name from
the context of derived class method / constructor names.

Syntax1:- ClassName.methodname(self, list of values if any)

Syntax2:- ClassName.__init__(self, list of values if any)

```
#PolyEx1.py  
class Circle:  
    def draw(self): # Original Method  
        print("Drawing Circle")
```

```
class Rect(Circle):  
    def draw(self): # Overridden Method  
        print("Drawing Rect")  
        super().draw()
```

```
class Square(Rect):  
    def draw(self): # Overridden method  
        print("Drawing Square")  
        super().draw()
```

```
#main program  
s=Square()  
s.draw()
```

```

#PolyEx2.py
class Circle:
    def draw(self): # Original Method
        print("Drawing Circle")

class Rect(Circle):
    def draw(self): # Overridden Method
        print("Drawing Rect")
class Square(Rect):
    def draw(self): # Overridden method
        print("Drawing Square")
        Circle.draw(self)
        Rect.draw(self)

#main program
s=Square()
s.draw()


---


#PolyEx3.py
class Circle:
    def __init__(self): # Original default Constructor
        print("Drawing Circle--DC")

class Rect(Circle):
    def __init__(self): # Overridtent default constructor
        print("Drawing Rect--DC")
        super().__init__()

#main program
ro=Rect() # Object Creation and calls Default Constructor


---


#PolyEx4.py
class Circle:
    def __init__(self): # Original default Constructor
        print("Drawing Circle--DC")

class Rect(Circle):
    def __init__(self): # Overridtent default constructor
        print("Drawing Rect--DC")
        Circle.__init__(self)

#main program
ro=Rect() # Object Creation and calls Default Constructor


---


#PolyEx5.py
class Circle:
    def area(self): # Orioginal Method (One Form)
        self.r=float(input("Enter Radious:"))
        self.ac=3.14*self.r**2
        print("Area of Circle:{} ".format(self.ac))
class Square(Circle):

```

```

def area(self): # Overridden Methods (Multiple Forms)
    self.s=float(input("Enter Side:"))
    self.sa=self.s**2
    print("Area of Square:{} ".format(self.sa))
    print("-"*50)
    super().area()

class Rect(Square):
    def area(self): # Overridden Methods (Multiple Forms)
        self.l=float(input("Enter Length:"))
        self.b=float(input("Enter Breadth:"))
        self.ar=self.l*self.b
        print("Area of Rect:{} ".format(self.ar))
        print("-"*50)
        super().area()

#main program
r=Rect()
r.area()


---


#PolyEx6.py
class Circle:
    def area(self): # Orioginal Method (One Form)
        self.r=float(input("Enter Radious:"))
        self.ac=3.14*self.r**2
        print("Area of Circle:{} ".format(self.ac))

class Square(Circle):
    def area(self): # Overridden Methods (Multiple Forms)
        self.s=float(input("Enter Side:"))
        self.sa=self.s**2
        print("Area of Square:{} ".format(self.sa))

class Rect(Square):
    def area(self): # Overridden Methods (Multiple Forms)
        self.l=float(input("Enter Length:"))
        self.b=float(input("Enter Breadth:"))
        self.ar=self.l*self.b
        print("Area of Rect:{} ".format(self.ar))
        print("-"*50)
        Circle.area(self)
        print("-"*50)
        super().area()

#main program
r=Rect()
r.area()


---


#PolyEx7.py
class Circle:
    def area(self): # Orioginal Method (One Form)
        self.r=float(input("Enter Radious:"))
        self.ac=3.14*self.r**2
        print("Area of Circle:{} ".format(self.ac))

class Square:
    def area(self): # Orioginal Method (One Form)
        self.s=float(input("Enter Side:"))

```

```

        self.sa=self.s**2
        print("Area of Square:{} ".format(self.sa))
class Rect(Square,Circle):
    def area(self): # Overridtent Method
        self.l=float(input("Enter Length:"))
        self.b=float(input("Enter Breadth:"))
        self.ar=self.l*self.b
        print("Area of Rect:{} ".format(self.ar))
        print("-----")
        Square.area(self)# OR super().area()
        Circle.area(self)

#main program
r=Rect()
r.area()


---


#PolyEx8.py
class Circle:
    def __init__(self): # Original Constructor (One Form)
        self.r=float(input("Enter Radious:"))
        self.ac=3.14*self.r**2
        print("Area of Circle:{} ".format(self.ac))

class Square:
    def __init__(self): # Original Constructor (One Form)
        self.s=float(input("Enter Side:"))
        self.sa=self.s**2
        print("Area of Square:{} ".format(self.sa))

class Rect(Square,Circle):
    def __init__(self): # Overridtent Constructor
        self.l=float(input("Enter Length:"))
        self.b=float(input("Enter Breadth:"))
        self.ar=self.l*self.b
        print("Area of Rect:{} ".format(self.ar))
        print("-----")
        Square.__init__(self)
        Circle.__init__(self)

#main program
r=Rect()


---


#PolyEx9.py
class Circle:
    def __init__(self,r): # Original Constructor (One Form)
        self.ac=3.14*r**2
        print("Area of Circle:{} ".format(self.ac))

class Square:
    def __init__(self,s): # Original Constructor (One Form)
        self.sa=s**2
        print("Area of Square:{} ".format(self.sa))

class Rect(Square,Circle):
    def __init__(self,l,b): # Overridtent Constructor
        self.ar=l*b
        print("Area of Rect:{} ".format(self.ar))

```

```

        print("-----")
        super().__init__(float(input("Enter Side:")))
        print("-----")
        Circle.__init__(self,float(input("Enter Radious:")))

#main program
l=float(input("Enter Length:"))
b=float(input("Enter Breadth:"))
r=Rect(l,b)


---


#PolyEx10.py
class Circle:
    def area(self,r): # Original Constructor (One Form)
        self.ac=3.14*r**2
        print("Area of Circle:{} ".format(self.ac))

class Square:
    def area(self,s): # Original Constructor (One Form)
        self.sa=s**2
        print("Area of Square:{} ".format(self.sa))

class Rect(Square,Circle):
    def area(self,l,b): # Overridtent Constructor
        self.ar=l*b
        print("Area of Rect:{} ".format(self.ar))
        print("-----")
        super().__init__(float(input("Enter Side:")))
        print("-----")
        Circle.area(self,float(input("Enter Radious:")))

#main program
l=float(input("Enter Length:"))
b=float(input("Enter Breadth:"))
r=Rect()
r.area(l,b)


---


#PolyEx11.py
class Circle:
    def area(self,r): # Original Constructor (One Form)
        self.ac=3.14*r**2
        print("Area of Circle:{} ".format(self.ac))

class Square:
    def __init__(self,s): # Original Constructor (One Form)
        self.sa=s**2
        print("Area of Square:{} ".format(self.sa))

class Rect(Square,Circle):
    def __init__(self,l,b): # Overridtent Constructor
        self.ar=l*b
        print("Area of Rect:{} ".format(self.ar))
        print("-----")
        super().__init__(float(input("Enter Side:")))
        print("-----")
        Circle.area(self,float(input("Enter Radious:")))

#main program
l=float(input("Enter Length:"))
b=float(input("Enter Breadth:"))

```

```

r=Rect(l,b)


---


#UnivCollStud.py
class Univ:
    def getunivdet(self):
        self.uname=input("Enter University Name:")
        self.uloc=input("Enter University Location:")
    def dispunivdet(self):
        print("-"*40)
        print("University details:")
        print("-"*40)
        print("University Name:",self.uname)
        print("University Location:",self.uloc)
        print("-"*40)

class College(Univ):
    def getcolldet(self):
        self.cname=input("Enter College Name:")
        self.cloc=input("Enter College Location:")
    def dispcolldet(self):
        print("-"*40)
        print("College details:")
        print("-"*40)
        print("College Name:",self.cname)
        print("College Location:",self.cloc)
        print("-"*40)

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("-"*40)
        print("Student details:")
        print("-"*40)
        print("Student Number:",self.sno)
        print("Student Name:",self.sname)
        print("Student Course:",self.crs)
        print("-"*40)

    """def savestuddata(self):
        #PDBC CODE
        con=cx_Oracle.connect("scott/tiger@localhost/orcl")
        cur=con.cursor()
        cur.execute("insert into StudentDet
values(%d,'%s','%s','%s','%s','%s','%s')")
        %(self.sno,self.sname,self.crs,self.cname,self.cloc,self.uname,self.uloc))
        con.commit()
        print("Student Data Saved in Table successfully")"""

#main program
s=Student()

```

```

s.getstuddet()
s.getcolldet()
s.getunivdet()
s.dispunivdet()
s.dispcolldet()
s.dispstuddet()

*****
create table StudentDet(sno number(2),sname varchar2(10),crs varchar2(10), cname
varchar2(10),cloc varchar2(10),uname varchar2(10),uloc varchar2(10))
****

#UnivCollStudDb.py
import cx_Oracle
class Univ:
    def getunivdet(self):
        self.uname=input("Enter University Name:")
        self.uloc=input("Enter University Location:")
    def dispunivdet(self):
        print("-"*40)
        print("University details:")
        print("-"*40)
        print("University Name:",self.uname)
        print("University Location:",self.uloc)
        print("-"*40)

class College(Univ):
    def getcolldet(self):
        self.cname=input("Enter College Name:")
        self.cloc=input("Enter College Location:")
    def dispcolldet(self):
        print("-"*40)
        print("College details:")
        print("-"*40)
        print("College Name:",self.cname)
        print("College Location:",self.cloc)
        print("-"*40)

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("-"*40)
        print("Student details:")
        print("-"*40)
        print("Student Number:",self.sno)
        print("Student Name:",self.sname)
        print("Student Course:",self.crs)
        print("-"*40)
    def savestuddata(self):

```

```

#PDBC CODE
con=cx_Oracle.connect("scott/tiger@localhost/orcl")
cur=con.cursor()
cur.execute("insert into StudentDet
values(%d,'%s','%s','%s','%s','%s','%s')"
%(self.sno,self.sname,self.crs,self cname,self.cloc,self.uname,self.uloc))
con.commit()
print("Student Data Saved in Table successfully")

#main program
s=Student()
s.getstuddet()
s.getcolldet()
s.getunivdet()
s.dispunivdet()
s.dispcolldet()
s.dispstuddet()
s.savestuddata()

#####
create table StudentDet(sno number(2),sname varchar2(10),crs varchar2(10), cname
varchar2(10),cloc varchar2(10),uname varchar2(10),uloc varchar2(10))
#####

```

Numpy Module---Data Science

Index

- =>Purpose of Numpy
 - =>History of Numpy
 - =>Advantages of Numpy
 - =>Similarities and Differences Between List and Numpy
 - =>Importance of "ndarray"
 - =>Number of Approaches to create an object of nd array
 - =>Attributes of ndarray
 - =>Types of Ndarrays (1-D, 2-D and n-Dimesional Arrays)
 - =>Basic Indexing and Slicing Operations on ndarray objects
 - =>Advanced Indexing and Slicing Operations on ndarray objects
 - =>Selecting or Filtering the values of ndarray objects
 - =>deleting, updating and adding the values of ndarray
 - =>Arithmetic or Matrix Operations on ndarray
 - =>Statistical Operations on ndarray
 - =>copy and view of ndarray
 - =>Sorting the values of ndarray
-

x

Numpy

Introduction to Numpy:

- =>Numpy stands for Numerical Python.
- =>Numpy is one of the pre-defined third party module / Library and numpy module is not a pre-defined module in Python Language.
- =>Syntax for installing any module:

```
pip install module-name
```

=>Example: Install numpy module

```
pip install numpy
```

- =>To use numpy as part of our program, we must import numpy module.
 - =>A Numpy module is a collection of Variables, Functions and Classes.
-

History of Numpy:

- =>Numpy was developed by studying existing module called "Numeric Library"(origin for development of numpy module)
- =>The Numeric Library was developed by JIM HUNGUNIAN
- =>The Numeric Library was not able to solve complex maths calculations.
- =>Numpy module developed by TRAVIS OLIPHANT for solving complex maths

calculations and array organization.
=>Numpy Module developed in the year 2005
=>Numpy Module developed in C and PYTHON languages.

Advantages of using NumPy

Need of NumPy:

=>With the revolution of data science, data analysis libraries like NumPy, SciPy, Scikit, Pandas, etc. have seen a lot of growth. With a much easier syntax than other programming languages, python is the first choice language for the data scientist.
=>NumPy provides a convenient and efficient way to handle the vast amount of data. NumPy is also very convenient with Matrix Operations and data reshaping. NumPy is fast which makes it reasonable to work with a large set of data.

The advantages of Numpy Programming are:

- 1) With Numpy Programming, we can deal with Arrays such 1-D, 2-D and Multi Dimensional Arrays.
 - 2) NumPy maintains minimal memory for large sets of data:
 - 3) Numpy provides Fast in Performing Operations bcoz internally its data is available at same address.
 - 4) NumPy performs array-oriented computing.
 - 5) It efficiently implements the multidimensional arrays.
 - 6) It performs scientific computations.
 - 7) It is capable of performing reshaping the data stored in multidimensional arrays.
 - 8) NumPy provides Many in-built functions for Various Complex Mathematical Operations such as statistical , financial, trigonometric Operations etc.
-

Python Traditional List VS Numpy Module

Similarities of python Traditional List VS Numpy Module:

=>An object of list used to store multiple values of same type or different type and both types (unique +duplicates) in single object.
=>In Numpy Programming, the data is organized in the object of "ndarray", which is one of the pre-defined class in numpy module. Hence an object of ndarray can store same type or different type and both types (unique +duplicates) in single object.
=>The objects of ndarray and list are mutable (changes can takes place)

Differences between Python Traditional List and ndarray object of Numpy Module:

=>An object of list contains both homogeneous and heterogeneous values whereas an object of ndarray of numpy can store only similar type of values(even we store different values, internally they are treated as similar type by treating all values of type "object").

=>On the object of list, we can't perform Vector Based Operations. whereas on the object of ndarray, we can perform Vector based operations.

=>In large sampling of data, List based applications takes more memory space whereas ndarray object takes less memory space.

=>List based applications are not efficient bcoz list object values takes more time to extract or retrieve (they are available at different Address) whereas numpy based applications are efficient bcoz of ndarray object values takes less time to extract or retrieve(they are available at same Address / clustered).

=>List object can't perform complex mathematical operations whereas an object of ndarray can perform complex mathematical operations.

===== X =====

Number of approaches to create an object of ndarray

=>"ndarray" is one of the pre-defined class of numpy module and whose object is used for storing the data in numpy programming in the form of 1-D, 2-D and n-Dimensional Arrays.

=>In numpy programming, we have the following essential approaches to create an object of ndarray.

1. array()
2. arange()
3. zeros()
4. ones()
5. full()
6. identity()
7. hstack()
8. vstack()

1) array():

=>This Function is used for converting Traditional Python Objects into ndarray object.

=>Syntax:- varname=numpy.array(Object,dtype)
=>Here var name is an object of <class,ndarray>
=>here array() is pre-defined function of numpy module used for
converting Traditional Python Objects into ndarray object.
=>object represents any Traditional Python Objects
=>dtype represents any numpy data type such as int8,int16,int32,float16,
float 32, float64,...etc (Internal dfata types of C lang)

Examples:

```
>>> import numpy as np
>>> l1=[10,20,30,40,50,60]
>>> print(l1,type(l1))-----[10, 20, 30, 40, 50, 60] <class 'list'>
>>> a=np.array(l1)
>>> print(a,type(a))-----[10 20 30 40 50 60] <class 'numpy.ndarray'>
>>> t=(10,20,30,40,50,60,70)
```

```

>>> print(t,type(t))-----(10, 20, 30, 40, 50, 60, 70) <class 'tuple'>
>>> a=np.array(t)
>>> print(a,type(a))-----[10 20 30 40 50 60 70] <class 'numpy.ndarray'>
>>> d1={10:1.2,20:4.5,30:6.7}
>>> a=np.array(d1)
>>> a----array({10: 1.2, 20: 4.5, 30: 6.7}, dtype=object)
-----
>>> t=(10,20,30,40,50,60)
>>> a=np.array(t)
>>> a-----array([10, 20, 30, 40, 50, 60])
>>> a.ndim-----1
>>> a.dtype-----dtype('int32')
>>> a.shape-----(6,)
>>> b=a.reshape(3,2)
>>> c=a.reshape(2,3)
>>> b-----
        array([[10, 20],
               [30, 40],
               [50, 60]])
>>> c
        array([[10, 20, 30],
               [40, 50, 60]])
>>> print(b,type(b))
        [[10 20]
         [30 40]
         [50 60]] <class 'numpy.ndarray'>
>>> print(c,type(c))
        [[10 20 30]
         [40 50 60]] <class 'numpy.ndarray'>
>>> b.ndim-----2
>>> c.ndim-----2
>>> b.shape-----(3, 2)
>>> c.shape-----(2, 3)
>>> d=a.reshape(3,3)----ValueError: cannot reshape array of size 6 into shape (3,3)
-----
-----
>>> t1=((10,20),(30,40))
>>> print(t1,type(t1))-----((10, 20), (30, 40)) <class 'tuple'>
>>> a=np.array(t1)
>>> a
        array([[10, 20],
               [30, 40]])
>>> a.ndim-----2
>>> a.shape-----(2, 2)
-----
>>> t1=(( (10,20,15),(30,40,25)),( (50,60,18),(70,80,35) ))
>>> print(t1,type(t1))
(((10, 20, 15), (30, 40, 25)), ((50, 60, 18), (70, 80, 35))) <class 'tuple'>
>>> a=np.array(t1)
>>> a

```

```
array([[[10, 20, 15],  
       [30, 40, 25]],  
  
      [[50, 60, 18],  
       [70, 80, 35]]])  
>>> print(a)  
[[[10 20 15]  
 [30 40 25]]  
  
 [[50 60 18]  
 [70 80 35]]]  
>>> a.ndim  
3  
>>> a.shape  
(2, 2, 3)  
>>> b=a.reshape(4,3)  
>>> b  
array([[10, 20, 15],  
       [30, 40, 25],  
       [50, 60, 18],  
       [70, 80, 35]])  
>>> c=a.reshape(3,4)  
>>> c  
array([[10, 20, 15, 30],  
       [40, 25, 50, 60],  
       [18, 70, 80, 35]])  
>>> d=a.reshape(3,2,2)  
>>> d  
array([[[10, 20],  
       [15, 30]],  
  
      [[40, 25],  
       [50, 60]],  
  
      [[18, 70],  
       [80, 35]]])  
>>> d[0]  
array([[10, 20],  
       [15, 30]])  
>>> d[1]  
array([[40, 25],  
       [50, 60]])  
>>> d[2]  
array([[18, 70],  
       [80, 35]])  
>>>
```

2. arange():

Syntax1:- varname=numpy.arange(Value)
Syntax2:- varname=numpy.arange(Start,Stop)
Syntax3:- varname=numpy.arange(Start,Stop,Step)
=>Here var name is an object of <class,ndarray>

=>Syntax-1 creates an object of ndarray with the values from 0 to value-1
=>Syntax-2 creates an object of ndarray with the values from Start to Stop-1
=>Syntax-3 creates an object of ndarray with the values from Start to Stop-1 with equal Interval of Value----step
=>arange() always create an object of ndarray in 1-D array only but not Possible to create directly 2-D and Multi Dimesional Arrays.
=>To create 2-D and Multi Dimesional Arrays, we must use reshape() or shape attribute

Examples:

```
>>> import numpy as np
>>> a=np.arange(10)
>>> a-----array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
>>> a.ndim-----1
>>> a=np.arange(50,62)
>>> print(a,type(a))---[50 51 52 53 54 55 56 57 58 59 60 61] <class 'numpy.ndarray'>
>>> a.ndim-----1
>>> a=np.arange(10,23,2)
>>> a-----array([10, 12, 14, 16, 18, 20, 22])
>>> a=np.arange(10,22,2)
>>> a-----array([10, 12, 14, 16, 18, 20])
>>> b=a.reshape(2,3)
>>> c=a.reshape(3,2)
>>> b-----
        array([[10, 12, 14],
               [16, 18, 20]])
>>> c
        array([[10, 12],
               [14, 16],
               [18, 20]])
>>> b.ndim----- 2
>>> c.ndim----- 2
>>> b.shape-----(2, 3)
>>> c.shape-----(3, 2)
>>> l1=[ [[10,20],[30,40]], [[15,25],[35,45]] ]
>>> l1-----[[[10, 20], [30, 40]], [[15, 25], [35, 45]]]
>>> a=np.arange(l1)-----TypeError: unsupported operand type(s) for -: 'list' and 'int'
```

3. zeros():

=>This Function is used for building ZERO matrix either with 1-D or 2-D or n-D
=>Syntax: varname=numpy.zeros(shape,dtype)

=>Here Shape can be 1-D(number of Zeros) or 2-D(Rows,Cols) or n-D(Number of Matrices,Number of Rows, Number of Columns)

Examples:

```
>>> import numpy as np
>>> a=np.zeros(12)
>>> a-----array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.])
>>> a=np.zeros(12,dtype=int)
>>> a-----array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
>>> a.reshape(3,4)
      array([[0, 0, 0, 0],
             [0, 0, 0, 0],
             [0, 0, 0, 0]])
>>> a.reshape(4,3)
      array([[0, 0, 0],
             [0, 0, 0],
             [0, 0, 0],
             [0, 0, 0]])
>>> a.reshape(6,2)
      array([[0, 0],
             [0, 0],
             [0, 0],
             [0, 0],
             [0, 0],
             [0, 0]])
>>> a.reshape(2,6)
      array([[0, 0, 0, 0, 0, 0],
             [0, 0, 0, 0, 0, 0]])
>>> a.reshape(2,3,2)
      array([[[0, 0],
               [0, 0],
               [0, 0]],
             [[0, 0],
               [0, 0],
               [0, 0]]])
>>> a.reshape(2,2,2)-----ValueError: cannot reshape array of size 12 into shape (2,2,2,2)
>>> a.reshape(3,2,2)
      array([[[0, 0],
               [0, 0]],
             [[0, 0],
               [0, 0]],
             [[0, 0],
               [0, 0]]])
>>> a.reshape(2,3,2)
      array([[[0, 0],
```

```
[0, 0],  
[0, 0]],  
  
[[0, 0],  
 [0, 0],  
 [0, 0]]])  
>>> a.reshape(2,2,3)  
array([[[0, 0, 0],  
        [0, 0, 0]],  
  
       [[0, 0, 0],  
        [0, 0, 0]]])  
-----  
>>> import numpy as np  
>>> a=np.zeros((3,3),dtype=int)  
>>> a  
array([[0, 0, 0],  
       [0, 0, 0],  
       [0, 0, 0]])  
>>> a=np.zeros((2,3))  
>>> a  
array([[0., 0., 0.],  
       [0., 0., 0.]])  
>>> a=np.zeros((2,3),int)  
>>> a  
array([[0, 0, 0],  
       [0, 0, 0]])  
>>> a=np.zeros((3,2,3),dtype=int)  
>>> a  
array([[[0, 0, 0],  
        [0, 0, 0]],  
  
       [[0, 0, 0],  
        [0, 0, 0]],  
  
       [[0, 0, 0],  
        [0, 0, 0]]])  
>>> print(a,type(a))  
[[[0 0 0]  
 [0 0 0]]  
  
 [[0 0 0]  
 [0 0 0]]  
  
 [[0 0 0]  
 [0 0 0]]] <class 'numpy.ndarray'>  
-----  
-----
```

4. ones()

=>This Function is used for building ONEs matrix either with 1-D or 2-D or n-D
=>Syntax: varname=numpy.ones(shape,dtype)

=>Here Shape can be 1-D(number of ones) or 2-D(Rows,Cols) or n-D(Number of Matrices,Number of Rows, Number of Columns)

Examples:

```
>>> import numpy as np
>>> a=np.ones(10)
>>> print(a,type(a))-----[1. 1. 1. 1. 1. 1. 1. 1. 1.] <class 'numpy.ndarray'>
>>> a=np.ones(10,dtype=int)
>>> print(a,type(a))-----[1 1 1 1 1 1 1 1 1] <class 'numpy.ndarray'>
>>> a.shape-----(10,)
>>> a.shape=(5,2)
>>> a
array([[1, 1],
       [1, 1],
       [1, 1],
       [1, 1],
       [1, 1]])
>>> a.ndim-----      2
>>> a.shape----- (5, 2)
>>> a.shape=(2,5)
>>> a
array([[1, 1, 1, 1, 1],
       [1, 1, 1, 1, 1]])
>>> a.shape-----(2, 5)
>>>
>>> a=np.ones((3,4),dtype=int)
>>> a
array([[1, 1, 1, 1],
       [1, 1, 1, 1],
       [1, 1, 1, 1]])
>>> a=np.ones((4,3),dtype=int)
>>> print(a,type(a))
[[1 1 1]
 [1 1 1]
 [1 1 1]
 [1 1 1]] <class 'numpy.ndarray'>
>>> a.shape-----(4, 3)
>>> a.shape=(3,2,2)
>>> a
array([[[1, 1],
         [1, 1]],
        [[1, 1],
         [1, 1]]],
```

```

[[1, 1],
 [1, 1]]])
>>> a=np.ones((4,3,3),dtype=int)
>>> a
array([[[1, 1, 1],
         [1, 1, 1],
         [1, 1, 1]],

        [[1, 1, 1],
         [1, 1, 1],
         [1, 1, 1]],

        [[1, 1, 1],
         [1, 1, 1],
         [1, 1, 1]],

        [[1, 1, 1],
         [1, 1, 1],
         [1, 1, 1]])
```

5) full()

=>This is function is used for building a matrix by specifying fill value either 1-D or 2-D or n-D

=>Syntax:-

```
varname=numpy.full(shape,fill_value,dtype)
=>varname is an obejct of <class, numpy.ndarray>
=>Here Shape can be 1-D(number of Fill_Value) or 2-D(Rows,Cols) or n-D( Number of Matrices,Number of Rows, Number of Columns)
=>fill_value can be any number of programmer choice
```

Examples:

```

>>> a=np.full(3,1)
>>> a-----array([1, 1, 1])
>>>print(type(a))-----<class,numpy.ndarray>
>>> a=np.full(3,9)
>>> a-----array([9, 9, 9])
>>> a=np.full(6,8)
>>> a-----array([8, 8, 8, 8, 8, 8])
>>> a.shape=(3,2)
>>> a
array([[8, 8],
       [8, 8],
       [8, 8]])
>>> a=np.full(6,9)
>>> a-----array([9, 9, 9, 9, 9, 9])
```

```

>>> a.reshape(2,3)
      array([[9, 9, 9],
             [9, 9, 9]])
>>> a=np.full((3,3),9)
>>> a
      array([[9, 9, 9],
             [9, 9, 9],
             [9, 9, 9]])
>>> a=np.full((2,3),6)
>>> a
      array([[6, 6, 6],
             [6, 6, 6]])
>>> a.reshape(3,2)
      array([[6, 6],
             [6, 6],
             [6, 6]])
>>> a=np.full((3,3,3),7)
>>> a
      array([[[7, 7, 7],
              [7, 7, 7],
              [7, 7, 7]],
              [[7, 7, 7],
              [7, 7, 7],
              [7, 7, 7]],
              [[7, 7, 7],
              [7, 7, 7],
              [7, 7, 7]]])
=====
```

6) identity():

=>This function always build Identity or unit matrix
=>Syntax:- varname=numpy.identity(N,dtype)
=>Here N represents Either we can take Rows or Columns and PVM takes as NXN Matrix
(Square Matrix--Unit or Identity)

Examples:

```

>>> import numpy as np
>>> a=np.identity(3,dtype=int)
>>> print(a,type(a))
[[1 0 0]
 [0 1 0]
 [0 0 1]] <class 'numpy.ndarray'>
>>> a=np.identity(5,dtype=int)
>>> print(a,type(a))
[[1 0 0 0 0]
 [0 1 0 0 0]
 [0 0 1 0 0]
```

```
[0 0 0 1 0]
[0 0 0 0 1]] <class 'numpy.ndarray'>
```

X

Numpy--Basic Indexing

==>If we want to access Single element of 1D,2D and N-D arrays we must use the concept of Basic Indexing.

=>Accessing Single Element 1D-Array :

=>Syntax:- ndarrayname [Index]

=>Here 'index' can be either either +ve or -ve indexing

Examples:

```
>>> a=np.array([10,20,30,40,50,60])
>>> a
array([10, 20, 30, 40, 50, 60])
>>> a[0]
10
>>> a[3]
40
```

=>Accessing single Element of 2D :

=>Syntax:- ndarrayobj[row index , column index]

Examples:-

```
>>>import numpy as np
>>> a=np.array([10,20,30,40,50,60])
>>> b=a.reshape(2,3)
>>> b
array([[10, 20, 30],
       [40, 50, 60]])
>>> b[0,0]
10
>>> b[0,1]
20
>>> b[1,2]
60
```

=>Accessing single Element of 3D :

Syntax:- ndarrayobj[Index of matrix , row index , column index]

Examples:

```
>>> a=np.array([10,20,30,40,50,60,70,80])
```

```
>>> b=a.reshape(2,2,2)
```

```
>>> b
```

```
array([[[10, 20],  
       [30, 40]],
```

```
       [[50, 60],  
        [70, 80]]])
```

```
>>> b[0,0,0]-----10
```

```
>>> b[-1,0,0]-----50
```

```
>>> b[-2,1,1]-----40
```

Numpy--Indexing and Slicing Operations of 1D,2D and 3D array

1D Arrays Slicing:

Syntax:- 1ndarrayobj[begin:end:step]

Examples:

```
>>> a=np.array([10,20,30,40,50,60,70])
```

```
>>> a-----array([10, 20, 30, 40, 50, 60, 70])
```

```
>>> a[::-1]-----array([70, 60, 50, 40, 30, 20, 10])
```

```
>>> a[:]-----array([10, 20, 30, 40, 50, 60, 70])
```

2D Arrays Slicing:

Syntax:- ndrrayobj[i , j]

here 'i' represents Row Index

here 'j' represents Column Index

(OR)

Syntax:- 2ndarrayobj[Row Index, Column Index]

Syntax:- 2ndarrayobj[begin:end:step, begin:end:step]

Examples:

```
>>> a=np.array([[10,20,30],[40,50,60]])
```

```
>>> a
```

```

array([[10, 20, 30],
       [40, 50, 60]])
>>> a[0,0]
10
>>> a[0,:,0:1]
array([[10],
       [40]])
>>> a[0:,1:2]
array([[20],
       [50]])
>>> a[1,:,:]
array([[40, 50, 60]])
=====
```

3D Arrays Slicing

Syntax:- 3dndrrayobj[i,j,k]

here 'i' represents Which 2D matrix (Matrix Number-->0 1 2 3 4 5.....)

here 'j' represents which Rows in that 2D matrix

here 'k' represents which Columns in that 2D matrix

(OR)

Syntax:- 3dndrrayobj[Matrix Index, Row Index, Column Index]

(OR)

Syntax:- 3dndrrayobj[begin:end:step, begin:end:step, begin:end:step]

Examples:

```

>>> lst=[[ [1,2,3],[4,5,6],[7,8,9] ],[ [13,14,15],[16,17,18],[19,20,21] ] ]
>>> print(lst)
[[[1, 2, 3], [4, 5, 6], [7, 8, 9]], [[13, 14, 15], [16, 17, 18], [19, 20, 21]]]
>>> arr2=np.array(lst)
>>> print(arr2)
[[[ 1  2  3]
 [ 4  5  6]
 [ 7  8  9]]
 [[13 14 15]
 [16 17 18]
 [19 20 21]]]
>>> arr2.ndim
3
>>> arr2.shape
(2, 3, 3)
>>> arr2[:, :, 0:1]
array([[[ 1],
       [ 4],
       [ 7]],
      [[13],
       [16],
       [19]]])
```

```

[19]]])
>>> arr2[:, :, 1]
array([[[ 1],
       [ 4],
       [ 7]],

      [[13],
       [16],
       [19]]])
>>> arr2[:, 0:2, 1:3]
array([[[ 2,  3],
       [ 5,  6]],

      [[14, 15],
       [17, 18]]])
>>> arr2[:, :2, 1:]
array([[[ 2,  3],
       [ 5,  6]],

      [[14, 15],
       [17, 18]]])

```

===== Numpy---Advanced Indexing =====

==>If we want to access multiple elements, which are not in order (arbitrary elements) of 1D,2D and N-D arrays we must use the concept of Advanced Indexing.
=>If we want access the elements based on some condition then we can't use basic indexing and Basic Slicing Operations. To fullfill such type of requirements we must use advanced Indexing.

=>Accessing Multiple Arbitrary Elements ---1D :

=>Syntax:- ndarrayname [x]

=>Here 'x' can be either ndarray or list which represents required indexes of arbitrary elements.

Examples:

```

>>> lst=[10,20,30,40,50,60,70,80,90]
>>> a=np.array(lst)
>>> print(a)-----[10 20 30 40 50 60 70 80 90]
#access 10 30 and 80 elements
# here indexes of 10 30 and 80 are 0 2 7
>>>lst=[0,2,7] here [0,2,7] are indexes of 10 30 and 80
>>> indexes=np.array(lst) # here lst converted into ndarray object
>>> print(indexes)-----[0 2 7]
>>> print(a[indexes])-----[10 30 80]
(OR)

```

```
>>> ind=[0,2,7] # prepare the list of indexes of arbitray elements(10,30,80) of ndarray and  
pass to ndarray  
>>> print(a[ind]) -----[10 30 80]  
OR  
>>> print(a[[0,2,7] ]) -----[10 30 80]  
Examples:
```

```
-----  
Q1-->Access 20 30 80 10 10 30  
>>> lst=[10,20,30,40,50,60,70,80,90]  
>>> a=np.array(lst)  
>>> print(a)-----[10 20 30 40 50 60 70 80 90]  
>>> ind=[1,2,7,0,0,2] # [1,2,7,0,0,2] are the indexes of 20 30 80 10 10 30  
>>> print(a[ind])-----[20 30 80 10 10 30]
```

=>Accessing Multiple Arbitrary Elements ---2D :

=>Syntax:- ndarrayobj[[row indexes],[column indexes]]

Examples:-

```
-----  
>>>import numpy as np  
>>>mat=np.array([ [1,2,3,4],[5,6,7,8],[9,10,11,12],[13,14,15,16] ] )  
>>> print(mat)  
[[ 1  2  3  4]  
 [ 5  6  7  8]  
 [ 9 10 11 12]  
 [13 14 15 16]]
```

Q1) Access the principle diagonal elements 1 6 11 16

Ans:- mat[[0,1,2,3],[0,1,2,3]]
=>When the above statement is executed, The PVM takes internally as
mat[(0,0), (1,1), (2,2),(3,3)]----- 1 6 11 16

```
>>> mat[ [0,1,2,3],[0,1,2,3] ]-----array([ 1, 6, 11, 16])
```

Q2) Access the elements 6 14

Ans: mat[[1,3] , [1,1]]
=>When the above statement is executed, The PVM takes internally as
mat[(1,1),(3,1)]

```
>>> mat[[1,3],[1,1]]-----array([ 6, 14])
```

=>Accessing Multiple Arbitrary Elements ---3D :

Syntax:- ndarray[[Indexes of 2Dmatrix],[row indexes],[column indexes]]

Examples:

```
>>>import numpy as np
>>>l1=[ [ [1,2,3,4],[5,6,7,8],[9,10,11,12] ],[ [13,14,15,16],[17,18,19,20],[21,22,23,24] ] ]
>>>mat3d=np.array(l1)
>>>print(mat3d)
>>> print(mat3d)
[[[ 1  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]]
 [[13 14 15 16]
 [17 18 19 20]
 [21 22 23 24]]]
>>> mat3d.ndim
3
>>> mat3d.shape
(2, 3, 4)
```

Q1) Access the elements 1 14 24

Ans:- mat3d[[0,1,1], [0,0,2], [0,1,3]]

When the above statement is executed, Internally PVM takes as follows.

=>mat3d[(0,0,0),(1,0,1),(1,2,3)]-Gives-->1 14 24

Q1) Access the elements 10 16

```
>>> mat3d[[-2,-1],[-1,-3],[-3,-1]]-----array([10, 16])
```

=====

OR

```
>>>l1=[ [ [1,2,3,4],[5,6,7,8],[9,10,11,12] ],[ [13,14,15,16],[17,18,19,20],[21,22,23,24] ] ]
>>>a=np.array(l1)
>>>a
array([[ [ 1,  2,  3,  4],
 [ 5,  6,  7,  8],
 [ 9, 10, 11, 12]],
 [[13, 14, 15, 16],
 [17, 18, 19, 20],
 [21, 22, 23, 24]]])
>>> #ndarrayobj[ [MatrixIndex],[Row Index],[Col Index] ]---Syntax
>>> #ndarrayobj[ [MatrixIndex],[Row Index],[Col Index] ]
>>> #access 1,8,13,20
>>> matind=(0,0,1,1)
>>> rowind=(0,1,0,1)
>>> colind=(0,3,0,3)
>>>a[matind,rowind,colind]
```

```

array([ 1,  8, 13, 20])
>>> a[ [0,0,0,1,1,1],[0,1,2,0,1,2],[0,1,2,0,1,2] ]
array([ 1,  6, 11, 13, 18, 23])
=====X=====
=====
a=np.array([10,20,30,40,50,60,70,80,15,25,35,45,55,65,75,85])
print(a)

a.shape=(2,2,2,2)
print(a)
[[[[10 20]
   [30 40]]

  [[50 60]
   [70 80]]]

  [[[15 25]
   [35 45]]]

  [[55 65]
   [75 85]]]]]

#access 10 from a---4-D
a[0][0][0][0]-----10
# access 10 and 40 from a---4D
a[[0,0],[0,0],[0,1],[0,1]]----array([10, 40])
# access 60,55 and 15 from a---4D
a[ [0,1,1],[1,1,0],[0,0,0],[1,0,0] ]----array([60, 55, 15])
=====X=====
=====
```

Numpy--Arithmetic Operations (OR) Matrix Operations

=>On the objects of ndarray, we can apply all types of Arithmetic Operators.
=>To perform Arithmetic Operations on the objects of ndarray in numpy programming, we use the following functions.

- a) add()
- b) subtract()
- c) multiply()
- d) dot() or matmul()
- e) divide()
- f) floor_divide()
- g) mod()
- h) power()

=>All the arithmetic Functions can also be performed w.r.t Arithmetic Operators.
=>All these Arithmetic Operations are called Matrix Operations.

a) add():

Syntax:- varname=numpy.add(ndarrayobj1, ndarrayobj2)
=>This function is used for adding elements of ndarrayobj1, ndarrayobj2 and result can be displayed
Examples:

```
>>> l1=[[10,20],[30,40]]  
>>> l2=[[1,2],[3,4]]  
>>> a=np.array(l1)  
>>> b=np.array(l2)  
>>> a  
array([[10, 20],  
       [30, 40]])  
>>> b  
array([[1, 2],  
       [3, 4]])  
>>> c=np.add(a,b)  
>>> c  
array([[11, 22],  
       [33, 44]])
```

```
>>> l1=[[10,20],[30,40]]  
>>> l2=[[1,2],[3,4]]  
>>> a=np.array(l1)  
>>> b=np.array(l2)  
>>> a  
array([[10, 20],  
       [30, 40]])  
>>> b  
array([[1, 2],  
       [3, 4]])  
>>> c=a+b # we used operator + instead of add()  
>>> c  
array([[11, 22],  
       [33, 44]])
```

b) subtract()

Syntax:- varname=numpy.subtract(ndarrayobj1, ndarrayobj2)
=>This function is used for subtracting elements of ndarrayobj1, ndarrayobj2 and result can be displayed

Examples:

```
>>> l1=[[10,20],[30,40]]  
>>> l2=[[1,2],[3,4]]  
>>> a=np.array(l1)  
>>> b=np.array(l2)  
>>> a  
array([[10, 20],
```

```
[30, 40]])
>>> b
array([[1, 2],
       [3, 4]])
>>> c=np.subtract(a,b)
>>> c
array([[ 9, 18],
       [27, 36]])
-----
```

```
>>> d=a-b # we used operator - instead of subtract()
>>> d
array([[ 9, 18],
       [27, 36]])
=====
```

c) multiply():

Syntax:- varname=numpy.multiply(ndarrayobj1, ndarrayobj2)
=>This function is used for performing element-wise multiplication of ndarrayobj1, ndarrayobj2 and result can be displayed

Examples:

```
>>> l1=[[1,2],[3,4]]
>>> l2=[[5,6],[4,3]]
>>> a=np.array(l1)
>>> b=np.array(l2)
>>> a
array([[1, 2],
       [3, 4]])
>>> b
array([[5, 6],
       [4, 3]])
>>> c=np.multiply(a,b)
>>> c
array([[ 5, 12],
       [12, 12]])
-----
```

```
>>> e=a*b # we used operator * instead of multiply()
>>> e
array([[ 5, 12],
       [12, 12]])
=====
```

d) dot() (or) matmul()

=>To perform Matrix Multiplication, we use dot(), matmul()

Syntax:- varname=numpy.dot(ndarrayobj1, ndarrayobj2)
Syntax:- varname=numpy.matmul(ndarrayobj1, ndarrayobj2)

=>These functions is used for performing actual matrix multiplication of ndarrayobj1, ndarrayobj2 and result can be displayed

Examples:

Examples:

```
>>> l1=[[1,2],[3,4]]
>>> l2=[[5,6],[4,3]]
>>> a=np.array(l1)
>>> b=np.array(l2)
>>> a
    array([[1, 2],
           [3, 4]])
>>> b
    array([[5, 6],
           [4, 3]])
>>> d=np.dot(a,b)
>>> d
    array([[13, 12],
           [31, 30]])
>>> e=np.matmul(a,b)
>>> e
    array([[13, 12],
           [31, 30]])
```

e) divide()

Syntax:- varname=numpy.divide(ndarray1,ndarry2)

=>This function is used for performing element-wise division of ndarrayobj1, ndarrayobj2 and result can be displayed

```
>>> l1=[[10,20],[30,40]]
>>> l2=[[1,2],[3,4]]
>>> a=np.array(l1)
>>> b=np.array(l2)
>>> a
    array([[10, 20],
           [30, 40]])
>>> b
    array([[1, 2],
           [3, 4]])
>>> c=np.divide(a,b)
>>> c
    array([[10., 10.],
           [10., 10.]])
```

```
>>> d=a/b # we used operator / instead of divide()
>>> d
```

```
    array([[10., 10.],
           [10., 10.]])
```

f) floor_divide()

Syntax:- varname=numpy.floor_divide(ndarray1,ndarry2)
=>This function is used for performing element-wise floor division of ndarrayobj1, ndarrayobj2 and result can be displayed

```
>>> l1=[[10,20],[30,40]]
>>> l2=[[1,2],[3,4]]
>>> a=np.array(l1)
>>> b=np.array(l2)
>>> a
array([[10, 20],
       [30, 40]])
>>> b
array([[1, 2],
       [3, 4]])
>>> c=np.floor_divide(a,b)
>>> c
array([[10, 10],
       [10, 10]])
-----
>>> d=a//b # we used operator // instead of floor_divide()
>>> d
array([[10, 10],
       [10, 10]])
```

g) mod()

Syntax:- varname=numpy.mod(ndarray1,ndarry2)
=>This function is used for performing element-wise modulo division of ndarrayobj1, ndarrayobj2 and result can be displayed

Examples:

```
>>> l1=[[10,20],[30,40]]
>>> l2=[[1,2],[3,4]]
>>> a=np.array(l1)
>>> b=np.array(l2)
>>> a
array([[10, 20],
       [30, 40]])
>>> b
array([[1, 2],
       [3, 4]])
>>> c=np.mod(a,b)
>>> c
array([[0., 0.],
       [0., 0.]])
```

=>We can also do with operator %

```
>>> e=a%b
>>> e
```

```
array([[0, 0],
       [0, 0]],  dtype=int32)
```

h) power():

Syntax:- varname=numpy.power(ndarray1,ndarry2)
=>This function is used for performing element-wise exponential of ndarrayobj1, ndarrayobj2 and result can be displayed

```
>>> l1=[[10,20],[30,40]]
>>> l2=[[1,2],[3,4]]
>>> a=np.array(l1)
>>> b=np.array(l2)
>>> a
      array([[10, 20],
             [30, 40]])
>>> b
      array([[1, 2],
             [3, 4]])
>>>c=np.power(a,b)
>>>print(c)
      array([[ 10,  400],
             [27000, 2560000]],

>>> f=a**b # Instead of using power() we can use ** operator
>>> f
      array([[ 10,  400],
             [27000, 2560000]],  dtype=int32)
=====X=====
```

===== Numpy--Statistical Operations =====

=>On the object of ndarray, we can the following Statistical Operations .

- a) amax()
- b) amin()
- c) mean()
- d) median()
- e) var()
- f) std()

=>These operation we can perform on the entire matrix and we can also peform on columnwise (axis=0) and Rowwise (axis=1)

a) amax():

=>This functions obtains maximum element of the entire matrix.

=>Syntax1:- varname=numpy.amax(ndarrayobject)

=>Syntax2:- varname=numpy.amax(ndarrayobject,axis=0)--->obtains max

elements on the basis of columns.

=>Syntax3:- varname=numpy.amax(ndarrayobject, axis=1)--->obtains max elements on the basis of Rows.

Examples:

```
-----  
>>> l1=[[1,2,3],[4,2,1],[3,4,2]]  
>>> A=np.array(l1)  
>>> print(A)  
[[1 2 3]  
 [4 2 1]  
 [3 4 2]]  
>>> max=npamax(A)  
>>> cmax=npamax(A, axis=0)  
>>> rmax=npamax(A, axis=1)  
>>> print("Max element=",max)-----Max eleemnt= 4  
>>> print("Column Max eleemnts=",cmax)---Column Max eleemnts= [4 4 3]  
>>> print("Row Max eleemnts=",rmax)---Row Max eleemnts= [3 4 4]
```

b) amin():

=>This functions obtains minimum element of the entire matrix.

=>Syntax1:- varname=numpy.amin(ndarrayobject)

=>Syntax2:- varname=numpy.amin(ndarrayobject, axis=0)--->obtains min

=>Syntax3:- varname=numpy.amin(ndarrayobject, axis=1)--->obtains min

Examples:

```
-----  
>>> l1=[[1,2,3],[4,2,1],[3,4,2]]  
>>> A=np.array(l1)  
>>> print(A)  
[[1 2 3]  
 [4 2 1]  
 [3 4 2]]  
>>> min=np.amin(A)  
>>> cmin=np.amin(A, axis=0)  
>>> rmin=np.amin(A, axis=1)  
>>> print("Min eleemnt=",min)---Min eleemnt= 1  
>>> print("Column Min eleemnts=",cmin)---Column Min eleemnts= [1 2 1]  
>>> print("Row Min eleemnts=",rmin)---Row Min eleemnts= [1 1 2]
```

c) mean():

=>This is used for cal mean of the total matrix elements.

=>The formula for mean=(sum of all elements of matrix) / total number of elements.

Syntax1:- varname=numpy.mean(ndarrayobject)

Syntax2:- varname=numpy.mean(ndarrayobject, axis=0)--->Columnwise Mean

Syntax3:- varname=numpy.mean(ndarrayobject, axis=1)--->Rowwise Mean

Examples:

```
>>> l1=[[1,2,3],[4,2,1],[3,4,2]]
>>> A=np.array(l1)
>>> print(A)
[[1 2 3]
 [4 2 1]
 [3 4 2]]
>>> m=np.mean(A)
>>> cm=np.mean(A, axis=0)
>>> rm=np.mean(A, axis=1)
>>> print("Mean=",m)-----Mean= 2.4444444444444446
>>> print("Column Mean=",cm)--Column Mean= [2.66666667 2.66666667 2. ]
>>> print("Row Mean=",rm)--Row Mean= [ 2. 2.33333333 3. ]
```

d) median()

=>This is used for calculating / obtaining median of entire matrix elements.
=>Median is nothing but sorting the given data in ascending order and select middle element.
=>If the number of sorted elements are odd then center or middle element becomes median.
=>If the number sorted elements are even then select center or middle of two elements, add them and divided by 2 and that result becomes median.

Syntax1:- varname=numpy.median(ndarrayobject)

Syntax2:- varname=numpy.median(ndarrayobject, axis=0)

Syntax3:- varname=numpy.median(ndarrayobject, axis=1)

Examples:

```
>>> l1=[[1,2,3],[4,2,1],[3,4,2]]
>>> A=np.array(l1)
>>> print(A)
[[1 2 3]
 [4 2 1]
 [3 4 2]]
>>> md=np.median(A)
>>> cmd=np.median(A, axis=0)
>>> rmd=np.median(A, axis=1)
>>> print("Median=",md)----Median= 2.0
>>> print("Column Median=",cmd)---Column Median= [3. 2. 2.]
>>> print("Row Median=",rmd)-----Row Median= [2. 2. 3.]
>>> l1=[[2,3],[4,1]]
>>> A=np.array(l1)
>>> print(A)
[[2 3]
 [4 1]]
```

```
>>> md=np.median(A)
>>> cmd=np.median(A,axis=0)
>>> rmd=np.median(A,axis=1)
>>> print("Median=",md)--Median= 2.5
>>> print("Column Median=",cmd)--Column Median= [3. 2.]
>>> print("Row Median=",rmd)--Row Median= [2.5 2.5]
```

e) var():

Variance= $\text{sqr}(\text{mean}-\text{xi}) / \text{total number of elements}$
here 'xi' represents each element of matrix.

Syntax1:- varname=numpy.var(ndarrayobject)

Syntax2:- varname=numpy.var(ndarrayobject, axis=0)

Syntax3:- varname=numpy.var(ndarrayobject, axis=1)

Examples:

```
>>> l1=[[1,2,3],[4,2,1],[3,4,2]]
>>> A=np.array(l1)
>>> print(A)
[[1 2 3]
 [4 2 1]
 [3 4 2]]
>>> vr=np.var(A)
>>> cvr=np.var(A, axis=0)
>>> rvr=np.var(A, axis=1)
>>> print("Variance=",vr)--Variance= 1.1358024691358024
>>> print("Column Variance=",cvr)--Column Variance= [1.555555556 0.888888889
>>> print("Row Variance=",rvr)--Row Variance=[0.66666667 1.555555556 0.66666667]
```

f) std()

standard deviation=sqrt(var)

Syntax1:- varname=numpy.std(ndarrayobject)

Syntax2:- varname=numpy.std(ndarrayobject, axis=0)

Syntax3:- varname=numpy.std(ndarrayobject, axis=1)

Examples:

```
>>> l1=[[1,2,3],[4,2,1],[3,4,2]]
>>> A=np.array(l1)
>>> print(A)
[[1 2 3]
 [4 2 1]]
```

```

[3 4 2]
>>> vr=np.var(A)
>>> cvr=np.var(A, axis=0)
>>> rvr=np.var(A, axis=1)
>>> print("Variance=",vr)-- Variance= 1.1358024691358024
>>> print("Column Variance=",cvr)--Column Variance= [1.55555556 0.88888889
>>> print("Row Variance=",rvr)--Row Variance= [0.66666667 1.55555556 0.66666667]

-----
>>> sd=np.std(A)
>>> csd=np.std(A, axis=0)
>>> rsd=np.std(A, axis=1)
>>> print("std=",sd)--std= 1.0657403385139377
>>> print(" column std=",csd)-- column std= [1.24721913 0.94280904 0.81649658]
>>> print("Row std=",rsd)--Row std= [0.81649658 1.24721913 0.81649658]
=====X=====

```

Note: numpy module does not contain mode().
mode() present in statistics module of Python

mode() gives Highest Frequent Element in given object
Examples:

```

>>> import statistics as s
>>> l1=[10,20,30,10,20,40,10]
>>> s.mode(l1)-----10
>>> l1=[10,20,30,10,20,40,10,20]
>>> s.mode(l1)-----10
>>> l1=[20,10,30,10,20,40,10,20]
>>> s.mode(l1)-----20
>>> s.multimode(l1)-----[20, 10]

-----
>>> a=np.array(l1)
>>> s.mode(a)-----20
>>> s.multimode(a)-----[20, 10]
=====
```

NumPy Sorting Arrays

=>Sorting is nothing arranging the elements in an ordered sequence.
=>Ordered sequence is any sequence that has an order corresponding to elements, like numeric or alphabetical, ascending or descending.
=>The NumPy ndarray object has a function called sort(), that will sort a specified array.

Examples:

```

import numpy as np
arr = np.array([3, 2, 0, 1])
print(np.sort(arr)) # [0 1 2 3]
=====
```

```

import numpy as np
arr = np.array(['banana', 'cherry', 'apple'])
=====
```

```
print(np.sort(arr)) # ['apple' 'banana' 'cherry']
```

```
-----  
import numpy as np  
arr = np.array([True, False, True])  
print(np.sort(arr)) # [False True True]
```

Sorting a 2-D Array

If you use the sort() method on a 2-D array, both columns and Rows of nd array will be sorted.

Examples:

```
-----  
import numpy as np  
arr = np.array([[3, 2, 4], [5, 0, 1]])  
print(np.sort(arr))  
#output  
[[2 3 4]  
 [0 1 5]]
```

```
a=np.array([110, 20, -30, 40, 50, 160, 7, 8, 90])  
print(a)  
  
np.sort(a)-----array([-30,  7,  8, 20, 40, 50, 90, 110, 160])  
np.sort(a)[::-1]----array([160, 110, 90, 50, 40, 20, 8, 7, -30])  
a.shape=(3,3)  
a-----array([[110, 20, -30],  
             [ 40, 50, 160],  
             [ 7, 8, 90]])
```

```
np.sort(a,axis=0) # ColumnWise  
array([[ 7,  8, -30],  
       [ 40, 20, 90],  
       [110, 50, 160]])
```

```
-----  
print(a)  
array([[110, 20, -30],  
       [ 40, 50, 160],  
       [ 7, 8, 90]])
```

```
np.sort(a,axis=1) # Row Wise  
array([[-30, 20, 110],  
       [ 40, 50, 160],  
       [ 7, 8, 90]])
```

```
np.delete()
```

Python's Numpy library provides a method to delete elements from a numpy array based on index position i.e.

```
numpy.delete(arr, obj, axis=None)
```

arr : Numpy ndarray from which elements needs to be deleted.

obj : Index position or list of index positions of items to be deleted from numpy ndarray arr.

axis : Axis along which we want to delete.

If 1 then delete columns.

If 0 then delete rows.

Examples:

```
# Create a Numpy array from list of numbers
```

```
arr = np.array([4,5,6,7,8,9,10,11])
```

Now let's delete an element at index position 2 in the above created

numpy array,

```
# Delete element at index position 2
```

```
arr = np.delete(arr, 2)
```

```
print('Modified Numpy Array by deleting element at index position 2')
```

```
print(arr)
```

Output:----Modified Numpy Array by deleting element at index position 2

```
[ 4  5  7  8  9 10 11]
```

To delete multiple elements from a numpy array by index positions, pass the numpy array and list of index positions to be deleted to np.delete() i.e.

```
# Create a Numpy array from list of numbers
```

```
arr = np.array([4, 5, 6, 7, 8, 9, 10, 11])
```

```
# Delete element at index positions 1,2 and 3
```

```
arr = np.delete(arr, [1,2,3])
```

```
print('Modified Numpy Array by deleting element at index position 1, 2 & 3')
```

```
print(arr)
```

Output:-----Modified Numpy Array by deleting element at index position 1, 2 & 3

```
[ 4  8  9 10 11]
```

Delete rows & columns from a 2D Numpy Array

Suppose we have a 2D numpy array i.e.

```
# Create a 2D numpy array from list of list
```

```
arr2D = np.array([[11 ,12, 13, 11],
```

```
                 [21, 22, 23, 24],
```

```
                 [31, 32, 33, 34]])
```

```
print(arr2D)
```

Output:

```
[[11 12 13 11]
```

```
[21 22 23 24]
```

```
[31 32 33 34]]
```

=>Now let's see how to delete rows and columns from it based on index positions.

=>Delete a column in 2D Numpy Array by column number

=>To delete a column from a 2D numpy array using np.delete() we need to pass the axis=1 along with numpy array and index of column i.e.

```
# Delete column at index 1
```

```
arr2D = np.delete(arr2D, 1, axis=1)
```

```
print('Modified 2D Numpy Array by removing columns at index 1')
```

```
print(arr2D)
```

```
Output:
```

Modified 2D Numpy Array by removing columns at index 1

```
[[11 13 11]
```

```
[21 23 24]
```

```
[31 33 34]]
```

=>It will delete the column at index position 1 from the above created 2D numpy array.

=>Delete multiple columns in 2D Numpy Array by column number

=>Pass axis=1 and list of column numbers to be deleted along with numpy array to np.delete() i.e.

```
# Create a 2D numpy array from list of list
```

```
arr2D = np.array([[11 ,12, 13, 11],
```

```
                 [21, 22, 23, 24],
```

```
                 [31, 32, 33, 34]])
```

```
# Delete column at index 2 & 3
```

```
arr2D = np.delete(arr2D, [2,3], axis=1)
```

```
print('Modified 2D Numpy Array by removing columns at index 2 & 3')
```

```
print(arr2D)
```

```
Output:
```

Modified 2D Numpy Array by removing columns at index 2 & 3

```
[[11 12]
```

```
[21 22]
```

```
[31 32]]
```

It deleted the columns at index positions 2 and 3 from the above created 2D numpy array.

Delete a row in 2D Numpy Array by row number

Our original 2D numpy array arr2D is,

```
[[11 12 13 11]
```

```
[21 22 23 24]
```

```
[31 32 33 34]]
```

To delete a row from a 2D numpy array using np.delete() we need to pass the axis=0 along with numpy array and index of row i.e. row number,

```
# Delete row at index 0 i.e. first row
```

```
arr2D = np.delete(arr2D, 0, axis=0)
```

```
print('Modified 2D Numpy Array by removing rows at index 0')
print(arr2D)
Output:
```

```
[[21 22 23 24]
 [31 32 33 34]]
```

It will delete the row at index position 0 from the above created 2D numpy array.

Delete multiple rows in 2D Numpy Array by row number
Our original 2D numpy array arr2D is,

```
[[11 12 13 11]
 [21 22 23 24]
 [31 32 33 34]]
```

Pass axis=0 and list of row numbers to be deleted along with numpy array to np.delete() i.e.

```
# Delete rows at ro1 1 & 2
arr2D = np.delete(arr2D, [1, 2], axis=0)
print('Modified 2D Numpy Array by removing rows at index 1 & 2')
print(arr2D)
Output:
```

Modified 2D Numpy Array by removing rows at index 1 & 2

```
[[11 12 13 11]]
```

It deleted the row at index position 1 and 2 from the above created 2D numpy array.

Delete specific elements in 2D Numpy Array by index position

Our original 2D numpy array arr2D is,

```
[[11 12 13 11]
 [21 22 23 24]
 [31 32 33 34]]
```

When we don't pass axis argument to np.delete() then it's default value is None, which means 2D numpy array will be flattened for deleting elements at given index position. Let's use np.delete() to delete element at row number 0 and column 2 from our 2D numpy array,

```
# Delete element in row 0 and column 2 from 2D numpy array
modArr = np.delete(arr2D, 2)
print('Modified 2D Numpy Array by removing element at row 0 & column 2')
print(modArr)
Output:
Modified 2D Numpy Array by removing element at row 0 & column 2
[11 12 11 21 22 23 24 31 32 33 34]
```

Pandas

Introduction to Pandas:

=>Pandas is an open source Python Library / Module providing high performance and data manipulation and Analysis Tool.
=>The word PANDAs derived from PANel DATA
=>The pandas concept developed by WES MCKinney in the year 2008.
=>The Traditional Python Programming does not contain any Module for Data Analysis and Now Python Programming uses Pandas as an analysis tool.
=>Python Pandas can be used in wide range of fields like Finance Services, Statistics , retail marketing sectors..etc
=>pandas module developed in C and Python Languages.

Instalation of Pandas:

=>The standard python software / Distribution(CPYTHON) does not contain any module for data analysis and now we are using third party module called PANDAS and whose module name is pandas
=>Programmatically to use pandas as part of our python program, we must install pandas module by using pip tool.

Syntax:- pip install module name

Example:- pip install pandas

Key Features of Pandas:----> Series DataFrame

- 1) Fast and Efficient Data Frame with default customized indexing
 - 2) Tools for loading the data in in-memory data objects(objects of Series, DataFrame as Panels)
 - 3) We can access the data from pandas by using Labeled Based Slicing and indexing.
 - 4) Columns from in-memory data objects(objects of Series, DataFrame as Panel) can be deleted and inserted and updated
-

===== Data Structures used in Pandas =====

=>In Pandas programming, we can store the data in 2 types of Data structures. They are.
a) Series
b) DataFrame

=>The best of way of thinking of these data structures is that The higher dimensional Data Structure is a container of its lower dimensional data structure.

Examples:

=>Series is part of DataFrame.

===== Series =====

- =>It is a One-Dimensional Labelled Array Capable of Storing / Holding Homogeneous data of any type (Integer, String, float,.....Python objects etc).
 - =>The Axis Labels are collectively called Index.
 - =>Pandas Series is nothing but a column value in excel sheet.
 - =>Pandas Series Values are Mutable.
 - =>Pandas Series contains Homogeneous Data (Internally even we store different types values , They are treated as object type)
-

Creating a Series

- >A Series object can be created by using the folowing Syntax:

Syntax:-

```
varname=pandas.Series(object, index, dtype)
```

Explanation:-

- =>Here varname is an object of <class, pandas.core.series.Series >
 - =>pandas is one of the pre-defined third party module name
 - =>Series() is pre-defined Function in pandas module and it is used for creating an object of Series class.
 - =>'object' can either int, float, complex, bool, str, bytes, bytearray,range, list, ndarray, dictetc (But not set type bcoz they are un-ordered)
 - =>'index' represents the position of values present Series object. The default value of Index starts from 0 to n-1, Here n represents number of values in Series object. Programatically we can give our own Index Values.
 - =>'dtype' represents data type (Ex:- int32, ,int64, float32, float64...etc)
-

Examples:- Create a series for 10 20 30 40 50 60

```
>>> import pandas as pd
>>> import numpy as np
>>> lst=[10,20,30,40,50,60]
>>> s=pd.Series(lst)
>>> print(s,type(s))
0    10
1    20
2    30
3    40
4    50
5    60
dtype: int64          <class 'pandas.core.series.Series'>
-----
```

```
>>> lst=[10,20,30,40,50,60]
>>> s=pd.Series(lst,dtype=float)
>>> print(s,type(s))
0    10.0
1    20.0
2    30.0
```

```
3 40.0
4 50.0
5 60.0
dtype: float64 <class 'pandas.core.series.Series'>
-----
>>> lst=["Rossum","Gosling","Travis","MCKinney"]
>>> a=np.array(lst)
>>> a -----array(['Rossum', 'Gosling', 'Travis', 'MCKinney'], dtype='|<U8')
>>> print(a, type(a))--['Rossum' 'Gosling' 'Travis' 'MCKinney'] <class 'numpy.ndarray'>
>>> s=pd.Series(a)
>>> print(s,type(s))
0    Rossum
1    Gosling
2    Travis
3   MCKinney
dtype: object    <class 'pandas.core.series.Series'>
-----
>>>lst=[10,"Rossum",34.56,"Author"]
>>> s=pd.Series(lst)
>>> print(s,type(s))
0      10
1    Rossum
2    34.56
3    Author
dtype: object    <class 'pandas.core.series.Series'>
-----
Creating an Series object with Programmer-defined Index
-----
>>> lst=[10,"Rossum",34.56,"Author"]
>>> print(lst)-----[10, 'Rossum', 34.56, 'Author']
>>> s=pd.Series(lst,index=["Stno","Name","Marks","Desg"])
>>> print(s)
  Stno      10
  Name    Rossum
  Marks    34.56
  Desg    Author
  dtype: object
>>> print(s["Stno"])-----10
-----
>>> lst=["Rossum","Gosling","Travis","MCKinney"]
>>> s=pd.Series(lst,index=[100,200,300,400])
>>> print(s,type(s))
  100    Rossum
  200    Gosling
  300    Travis
  400   MCKinney
dtype: object <class 'pandas.core.series.Series'>
-----
Creating a Series object from dict
-----
```

=>A dict object can be used for creating a series object
=>If we use dict object in Series() then keys can be taken as Indices (Or Indexes) automatically and corresponding values of dict can be taken as Series data.

Examples:

```
>>> import pandas as pd
>>> d1={"sub1":"Python","sub2":"Java","sub3":"Data Science","sub4":"ML"}
>>> print(d1)--{'sub1': 'Python', 'sub2': 'Java', 'sub3': 'Data Science', 'sub4': 'ML'}
>>> s=pd.Series(d1)
>>> print(s)
      sub1      Python
      sub2      Java
      sub3  Data Science
      sub4        ML
      dtype: object
>>> d2={"RS":2.3,"JG":1.2,"MCK":4.5,"TOLI":2.4}
>>> print(d2)--{'RS': 2.3, 'JG': 1.2, 'MCK': 4.5, 'TOLI': 2.4}
>>> s=pd.Series(d2)
>>> print(s)
      RS    2.3
      JG    1.2
      MCK   4.5
      TOLI  2.4
      dtype: float64
```

=====X=====

=====Counter=====

Index

=>What is Python Counter?
=>Why use Python Counter?
=>Counter with String
=>Counter with List
=>Counter with Dictionary
=>Counter with Tuple
=>Counter with Set
=>Accessing, Initializing and Updating Counters
=>Deleting an Element from Counter
=>Methods Available on Python Counter
=>Reassigning Counts in Python
=>Get and set the count of Elements using Counter

=>Python Counter is a container that will hold the count of each of the elements present in the container. The counter is a sub-class available inside the dictionary class.

Uses Python Counter

=>The Counter holds the data in an unordered collection.

- =>The elements here represent the keys and the count as values.
 - =>It allows you to count the items in an iterable list.
 - =>Arithmetic operations like addition, subtraction, intersection, and union can be easily performed on a Counter.
 - =>A Counter can also count elements from another counter
-

Syntax:

```
Varname=collections.Counter(Iterable object)
=>Varname is an object of <class 'collections.Counter'>
=>Iterable object can be list, tuple, dictionary, string
```

Example1:

```
from collections import Counter
list1 = ['x','y','z','x','x','x','y', 'z']
print(Counter(list1)) # Counter({'x': 4, 'y': 2, 'z': 2})
```

Example2:

```
from collections import Counter
dict1 = {'x': 4, 'y': 2, 'z': 2, 'z': 2}
print(Counter(dict1)) # Counter({'x': 4, 'y': 2, 'z': 2})
```

Example3

```
from collections import Counter
c=Counter("MIssisipi")
print(c) # Counter({'s': 3, 'i': 3, 'M': 1, 'T': 1, 'p': 1})
```

Example4:

```
from collections import Counter
tuple1 = ('x','y','z','x','x','y','z')
print(Counter(tuple1)) # Counter({'x': 4, 'y': 2, 'z': 2})
```

=>Methods Available on Python Counter

The methods available with Counter are:

- 1) elements() : This method will return you all the elements with count >0. Elements with 0 or -1 count will not be returned.

Examples:

```
d = Counter( a = 2, b = 3, c = 6, d = 1, e = 5)
for i in d.elements():
    print ( i, end = " ") # a a b b b c c c c c d e e e e
```

- 2) most_common(value): This method will return you the most common elements from Counter list.

```
c=Counter({'x': 5, 'y': 12, 'z': -2, 'x1':0})
ce = c.most_common(2)
print(ce) # [(y, 12), (x, 5)]
print(common_element)
```

3) subtract(): This method is used to deduct the elements from another Counter.

Examples:

```
from collections import Counter
c1 = Counter(A=4, B=3, C=10)
c2 = Counter(A=10, B=3, C=4)
c1.subtract(c2)
print(c1) # Counter({'c': 6, 'B': 0, 'A': -6})
```

4) update(): This method is used to update the elements from another Counter.

```
from collections import Counter
coun = Counter()
coun.update([1, 2, 3, 1, 2, 1, 1, 2])
print(coun) # Counter({1: 4, 2: 3, 3: 1})
coun.update([1, 2, 4])
print(coun) # Counter({1: 5, 2: 4, 3: 1, 4: 1})
```

5) total()--> This Function is used for adding all the occurrences of Counter Values.

Examples:

```
s="MISSISSIPPI"
c=Counter(s)
print("content of c=",c) # content of c= Counter({'T': 4, 'S': 4, 'P': 2, 'M': 1})
t=c.total()
print("Total=",t) # Total=11
```

#Program for finding number of occurrences of a String

#s="MISSISSIPPI" Output: M:1 I : 4 S : 4 P:2

#non-counter.py

s="MISSISSIPPI"

d={} # create an empty dict

for ch in s:

 if ch not in d:

 d[ch]=1

 else:

 d[ch]=d[ch]+1

else:

 print("content of dict:",d)

 print("-----")

 for k,v in d.items():

 print("\t{}--->{} ".format(k,v))

#Program for finding number of occurrences of a String

#s="MISSISSIPPI"

#counterex1.py

from collections import Counter

s="MISSISSIPPI" # here s is an object of str and it is considered as Iterable object

c=Counter(s) # here c is an object of <class, collections.Counter>

print("content of c=",c) # content of c= Counter({'T': 4, 'S': 4, 'P': 2, 'M': 1})

print("-----")

for on,ov in c.items():

 print("\t{}--->{} ".format(on,ov))

print("-----")

```

#counterex2.py
from collections import Counter as c
lst=[10,20,10,20,10,30,40,10,20,50,34,10] # Here lst is called list object
c1=c(lst)
print("content of c1=",c1)
print("-----")
for on in c1:
    print("\t{}-->{}".format(on,c1.get(on)))


---


#counterex3.py
from collections import Counter
c1=Counter((10,20,"Python","Python",20,"Python",10))
print("content of c1=",c1)
print("-----")
for on in c1.keys():
    print("\t{}-->{}".format(on,c1[on]))


---


#counterex4.py
from collections import Counter
c1=Counter({10:2,20:4,40:5,10:6})
print("content of c1=",c1)
print("-----")
for on in c1.keys():
    print("\t{}-->{}".format(on,c1[on]))


---


#counterex5.py
from collections import Counter
c1=Counter(a=10,b=20,c=3,d=4)
print("content of c1=",c1)
print("-----")
for on in c1.keys():
    print("\t{}-->{}".format(on,c1[on]))


---


#counterex6.py----update()
from collections import Counter
s="MISSISSIPPI"
c=Counter(s)
print("content of c=",c) # content of c= Counter({'I': 4, 'S': 4, 'P': 2, 'M': 1})
print("-----")
s1="bitter"
c.update(s1)
print("content of c after update=",c) # content of c= Counter({'I': 4, 'S': 4, 'P': 2, 'M': 1})
for k,v in c.items():
    print("\t{}-->{}".format(k,v))

c.popitem()
print("-----")
for k,v in c.items():
    print("\t{}-->{}".format(k,v))


---


#counterex7.py----most_common(val) and total()
from collections import Counter
s="MISSISSIPPI"
c=Counter(s)

```

```
print("content of c=",c) # content of c= Counter({'I': 4, 'S': 4, 'P': 2, 'M': 1})
print("-----")
print("content of c after update=",c) # content of c= Counter({'I': 4, 'S': 4, 'P': 2, 'M': 1})
for k,v in c.items():
    print("\t{}-->{}".format(k,v))
print("-----")
cm=c.most_common(2)
print(cm) # [(I, 4), (S, 4)]
print("-----")
t=c.total()
print("Total=",t) # Total=11
```

```
#counterex8.py
```

```
from collections import Counter
c1=Counter({10:2,20:4,40:5,10:3,25:-2,35:0})
print("content of c1=",c1)
print("-----")
for on in c1.keys():
    print("\t{}-->{}".format(on,c1[on]))
print("-----")
for e in c1.elements():
    print(e)
```

```
#counterex9.py
```

```
from collections import Counter
c=Counter() # Empty counter
print("content of c=",c)
c.update("AYAAN")
print("content of c after update=",c) # Counter({'A': 3, 'Y': 1, 'N': 1})
print('-----')
for v in c.elements():
    print("\t{}".format(v))
c['A']=c.get('A')+3
print("content of c after update=",c) # Counter({'A': 3, 'Y': 1, 'N': 1})
```

```
#counterex10.py
```

```
from collections import Counter
c1=Counter("abaabaabc")
c2=Counter("ababcdef")
c3=c1+c2
print("counter1=",c1)
print("counter2=",c2)
print("Add of c1 and c2=",c3)
c4=c2-c1
print("Sub of c1 and c2=",c4)
c5=c1|c2 # Union Operation
print("content of c5 after Union=",c5)
c6=c1&c2
print("content of c6 after Intersection=",c6)
```

=====

DataFrame in Pandas

=====

- =>A DataFrame is 2-Dimensional Data Structure to organize the data .
 - =>In Otherwords a DataFrame Organizes the data in the Tabular Format, which is nothing but Collection of Rows and Columns.
 - =>The Columns of DataFrame can be Different Data Types or Same Type
 - =>The Size of DataFrame can be mutable.
-

Number of approaches to create DataFrame

- =>To create an object of DataFrame, we use pre-defined DataFrame() which is present in pandas Module and returns an object of DataFrame class.
- =>We have 5 Ways to create an object of DataFrame. They are
 - a) By using list / tuple
 - b) By using dict
 - c) By using set type
 - d) By using Series
 - e) By using ndarray of numpy
 - f) By using CSV File (Comma Separated Values)

- =>Syntax for creating an object of DataFrame in pandas:

```
varname=pandas.DataFrame(object,index,columns,dtype)
```

Explanation:

- =>'varname' is an object of <class,'pandas.core.dataframe.DataFrame'>
- =>'pandas.DataFrame()' is a pre-defined function present in pandas module and it is used to create an object of DataFrame for storing Data sets.
- =>'object' represents list (or) tuple (or) dict (or) Series (or) ndarray (or) CSV file
- =>'index' represents Row index and whose default indexing starts from 0,1,...n-1 where 'n' represents number of values in DataFrame object.
- =>'columns' represents Column index whose default indexing starts from 0,1..n-1 where n number of columns.
- =>'dtype' represents data type of values of Column Value.

Creating an object DataFrame by Using list / tuple

```
>>>import pandas as pd  
>>>lst=[10,20,30,40]  
>>>df=pd.DataFrame(lst)  
>>>print(df)
```

```
0  
0 10  
1 20  
2 30  
3 40
```

```
lst=[[10,20,30,40],["RS","JS","MCK","TRV"]]  
df=pd.DataFrame(lst)  
print(df)
```

```

      0  1  2  3
0 10 20 30 40
1 RS JS MCK TRV
-----
lst=[[10,'RS'],[20,'JG'],[30,'MCK'],[40,'TRA']]
df=pd.DataFrame(lst)
print(df)
      0  1
0 10  RS
1 20  JG
2 30  MCK
3 40  TRA
-----
lst=[[10,'RS'],[20,'JG'],[30,'MCK'],[40,'TRA']]
df=pd.DataFrame(lst, index=[1,2,3,4],columns=['Rno','Name'])
print(df)

      Rno Name
1   10  RS
2   20  JG
3   30  MCK
4   40  TRA
-----
tpl=( ("Rossum",75), ("Gosling",85), ("Travis",65), ("Ritche",95),("McKinney",60) )
df=pd.DataFrame(tpl, index=[1,2,3,4,5],columns=['Name','Age'])
print(df)
      Name  Age
1  Rossum  75
2  Gosling  85
3  Travis   65
4  Ritche   95
5  McKinney 60
-----
```

Creating an object DataFrame by Using dict object

=>When we create an object of DataFrame by using Dict , all the keys are taken as Column Names and Values of Value are taken as Data.

Examples:

```

>>> import pandas as pd
>>>
dictdata={"Names":["Rossum","Gosling","Ritche","McKinney"],"Subjects":["Python","Java",
,"C","Pandas"],"Ages":[65,80,85,55] }
>>> df=pd.DataFrame(dictdata)
>>> print(df)
      Names Subjects Ages
0  Rossum    Python    65
1  Gosling     Java    80
2  Ritche       C    85
-----
```

```
      3 McKinney Pandas      55
>>> df=pd.DataFrame(dictdata,index=[1,2,3,4])
>>> print(df)
   Names Subjects    Ages
1  Rossum    Python     65
2  Gosling     Java      80
3  Ritche        C      85
4 McKinney Pandas      55
```

Creating an object DataFrame by Using Series object

```
>>> import pandas as pd
>>> sdata=pd.Series([10,20,30,40])
>>> df=pd.DataFrame(sdata)
>>> print(df)
   0
0 10
1 20
2 30
3 40
>>> sdata=pd.Series({"IntMarks":[10,20,30,40],"ExtMarks":[80,75,65,50]})
>>> print(sdata)
IntMarks [10, 20, 30, 40]
ExtMarks [80, 75, 65, 50]
dtype: object

>>> df=pd.DataFrame(sdata)
>>> print(df)
   0
   IntMarks [10, 20, 30, 40]
   ExtMarks [80, 75, 65, 50]
>>> ddata={"IntMarks":[10,20,30,40],"ExtMarks":[80,75,65,50]}
>>> df=pd.DataFrame(ddata)
>>> print(df)
   IntMarks  ExtMarks
0         10      80
1         20      75
2         30      65
3         40      50
```

Creating an object DataFrame by Using ndarray object

```
>>> import numpy as np
>>> l1=[[10,60],[20,70],[40,50]]
>>> a=np.array(l1)
>>> df=pd.DataFrame(a)
>>> print(df)
   0  1
0 10 60
1 20 70
```

```

2 40 50
>>> df=pd.DataFrame(a,columns=["IntMarks","ExtMarks"])
>>> print(df)
      IntMarks  ExtMarks
0            10         60
1            20         70
2            40         50

```

e) By using CSV File(Comma Separated Values)

```

import pandas as pd1
df=pd1.read_csv("D:\KVR-JAVA\stud.csv")
print("type of df=",type(df)) #type of df= <class 'pandas.core.frame.DataFrame'>
print(df)

```

----- OUTPUT -----

	stno	name	marks
0	10	Rossum	45.67
1	20	Gosling	55.55
2	30	Ritche	66.66
3	40	Travis	77.77
4	50	KVR	11.11

Accesssing the Data of DataFrame

- 1) DataFrameobj.head(no.of rows)
 - 2) DataFrameobj.tail(no.of rows)
 - 3) DataFrameobj.describe()
 - 4) DataFrameobj.shape
 - 5) DataFrameobj [start:stop:step]
 - 6) DataFrameobj["Col Name"]
 - 7) DataFrameobj[["Col Name1","Col Name-2"...."Col Name-n"]]
 - 8) DataFrameobj[["Col Name1","Col Name-2"...."Col Name-n"]] [start:stop:step]
 - 9) DataFrameobj.iterrows()
-

Understabding loc() ---- here start and stop index Included and
Col Names can be used(but not column numbers)]

- 1) DataFrameobj.loc[row_number]
 - 2) DataFrameobj.loc[row_number,[Col Name,.....]]
 - 3) DataFrameobj.loc[start:stop:step]
 - 4) DataFrameobj.loc[start:stop:step,["Col Name"]]
 - 5) DataFrameobj.loc[start:stop:step,["Col Name1", Col Name-2....."]]
 - 6) DataFrameobj.loc[start:stop:step,"Col Name1" : Col Name-n"]
-

Understabding iloc() ---- here start index included and stop index excluded and
Col Numbers must be used(but not column names)]

- 1) DataFrameobj.iloc[row_number]
 - 2) DataFrameobj.iloc[row_number, Col Number.....]
 - 3) DataFrameobj.iloc[row_number,[Col Number1, Col Number2.....]]
 - 3) DataFrameobj.iloc[row start:row stop, Col Start: Col stop]
 - 4) DataFrameobj.iloc[row start:row stop:step, Col Start: Col stop:step]
 - 5) DataFrameobj.iloc[row start:row stop, Col Number]
 - 6) DataFrameobj.iloc[[row number1, row number-2.....]]
 - 7) DataFrameobj.iloc[row start: row stop , [Col Number1, Col Number2.....]]
 - 8) DataFrameobj.iloc[:, [Col Number1, Col Number2.....]]
-
-
-

Adding new Column Name to Data Frame

- 1) dataframeobj['new col name']=default value
 - 2) dataframeobj['new col name']=expression
-
-
-

Removing Column Name from Data Frame

- 1) data.drop(columns="col name")
 - 2) data.drop(columns="col name", inplace=True)
-
-
-

sorting the dataframe data

- 1) data.sort_values(["colname"])
 - 2) data.sort_values(["colname"], ascending=False)
-
-
-

knowing duplicates in dataframe data

- 1) data.duplicated()-----gives boolean result
-
-
-

Removing duplicates from dataframe data

- 1) data.drop_duplicates()
 - 2) data.drop_duplicates(inplace=True)
-
-
-

Data Filtering and Conditional Change / updatons

- 1) data.loc[simple condition]

Ex: df.loc[df["maths"]>75]
df.loc[df["maths"]>90 ,["name","maths"]]

2) dataframeobj.loc[compound condition]

Ex: df.loc[(df["maths"]>60) & (df["maths"]<85)]
Ex: df.loc[(df["maths"]>95) &

(df["maths"]<=99),["name","maths"]]

MOST IMP

3) dataframeobj.loc[(compund condition), ["Col Name"]]=Expression

Ex: df.loc[(df["percent"]>=60) & (df["percent"]<=80),["grade"]]="First" # cond updattion.

To Export the DataFrame object data to the csv file

df.to_csv("E:\KVR-PYTHON-7AM\PANDAS\studfinaldata.csv")

To Export the DataFrame object data to the txt file

df.to_csv("E:\KVR-PYTHON-7AM\PANDAS\class_10.txt")

(or)

df.to_csv("E:\KVR-PYTHON-7AM\PANDAS\class_10.txt",index=False)

(OR)

df.to_csv("E:\KVR-PYTHON-7AM\PANDAS\class_10.txt",index=False,sep="\t")

To read the data from EXCEL into dataframe object

dataframeobj=pandas.read_excel("Absolute path of excel file")

Examples:

df=pd.read_excel("D:\KVR\kvr.xlsx")
print(df)

csv file

htno	name	telugu	english	hindi	maths	science	social
100	Ramesh	50	60	66	98	66	55
101	Rajesh	45	67	34	67	66	78
102	Rossum	56	88	56	99	44	77
103	Raji	56	78	34	56	88	55
104	Kalyan	51	63	62	93	67	51
105	Karthik	48	62	39	68	65	88
106	Kambli	53	81	59	92	48	73
107	Praveen	46	88	74	86	78	45
108	Ganesh	53	62	76	88	76	35
109	Nags	55	77	44	77	86	58
110	Biswa	66	48	86	95	48	47
111	Ritchi	66	68	64	76	98	75
104	Kalyan	51	63	62	93	67	51
112	shareef	50	63	99	90	76	67

DataFrame--GroupBy

=>The Group By mechanism in the Pandas provides a way to break a DataFrame into different groups or chunks based on the values of single or multiple columns.

=>Let's understand with some examples.

=>Assume we have a DataFrame,

ID	Name	Age	City	Experience
11	Jack	44	Sydney	19
12	Riti	41	Delhi	17
13	Aadi	46	Mumbai	11
14	Mohit	45	Delhi	15
15	Veena	43	Delhi	14
16	Shaunak	42	Mumbai	17
17	Manik	42	Sydney	14
18	Vikas	42	Delhi	11
19	Samir	42	Mumbai	15
20	Shobhit	40	Sydney	12

=>This DataFrame has a column ‘City’ which has three unique values like, “Delhi”, “Mumbai” and “Sydney”. We want to create different groups out of this DataFrame based on the column “City” values.

=>As this column has only three unique values, so there will be three different groups.

=>Group 1 will contain all the rows for which column “City” has the value “Delhi” i.e.

ID	Name	Age	City	Experience
12	Riti	41	Delhi	17
14	Mohit	45	Delhi	15
15	Veena	43	Delhi	14
18	Vikas	42	Delhi	11

Group 2 will contain all the rows for which column “City” has the value “Mumbai” i.e.

ID	Name	Age	City	Experience
13	Aadi	46	Mumbai	11
16	Shaunak	42	Mumbai	17
19	Samir	42	Mumbai	15

Group 3 will contain all the rows for which column “City” has the value “Sydney” i.e.

ID	Name	Age	City	Experience
11	Jack	44	Sydney	19
17	Manik	42	Sydney	14
20	Shobhit	40	Sydney	12

- DataFrame.groupby() method

DataFrame's groupby() method accepts column names as arguments. Based on the column values, it creates several groups and returns a DataFrameGroupBy object that contains information about these groups.

For example, let's create groups based on the column "City",

```
# Create Groups based on values in column 'city'  
groupObj = df.groupby('City')  
print(groupObj)
```

Output

```
<pandas.core.groupby.generic.DataFrameGroupBy object at 0x000002895CA14048>
```

The groupby() function created three groups because column 'City' has three unique values. It returned a DataFrameGroupBy object with information regarding all three groups.

Iterate over all the DataFrame Groups

DataFrame's groupby() function returns a DataFrameGroupBy object, which contains the information of all the groups. The DataFrameGroupBy is an iterable object. It means using a for loop, we can iterate over all the created Groups,

```
# Iterate over all the groups  
for grpName, rows in df.groupby('City'):
```

```
    print("Group Name: ", grpName)  
    print('Group Content: ')  
    print(rows)
```

Output:

```
Group Name: Delhi  
Group Content:  
    Name Age City Experience  
ID  
12  Riti  41  Delhi     17  
14  Mohit 45  Delhi     15  
15  Veena 43  Delhi     14  
18  Vikas 42  Delhi     11
```

Group Name: Mumbai

Group Content:

```
Name Age City Experience  
ID  
13 Aadi 46 Mumbai 11  
16 Shaunak 42 Mumbai 17  
19 Samir 42 Mumbai 15  
Group Name: Sydney  
Group Content:  
Name Age City Experience  
ID  
11 Jack 44 Sydney 19  
17 Manik 42 Sydney 14  
20 Shobhit 40 Sydney 12
```

--
Get first row of each Group

-->DataFrame's groupby() function returns a DataFrameGroupBy object, which contains the information of all the groups. The DataFrameGroupBy object also provides a function first(), and it returns a DataFrame containing the first row of each of the Group.

--
Get nth row of each Group

-->The pandas.groupby().nth() function is used to get the value corresponding the nth row for each group. To get the first value in a group, pass 0 as an argument to the nth() function.

For example

```
# Get first row of each group  
firstRowDf = df.groupby('City').first()  
print(firstRowDf)
```

Output:

```
Name Age Experience  
City  
Delhi Riti 41 17  
Mumbai Aadi 46 11  
Sydney Jack 44 19
```

There were three unique values in the column "City", therefore 3 groups were created. The first() function fetched the first row of each of the Group and returned a DataFrame populated with that. The returned DataFrame has a row for each of the city and it is the first row from each of the city groups.

--
Get the count of number of DataFrame Groups

--
The DataFrameGroupBy object also provides a function size(), and it returns the count of rows in each of the groups created by the groupby() function. For example,

```
# Get the size of DataFrame groups  
print(df.groupby('City').size())
```

Output:

```
Delhi      4  
Mumbai    3  
Sydney    3  
dtype: int64
```

As there were three unique values in the column “City”, therefore 3 groups were created by groupby() function. The size() function returned a Series containing the count of number of rows for each of the group.

```
===  
#GroupByEx1.py  
import pandas as pd  
# List of Tuples  
employees = [  
    (11, 'Jack', 44, 'Sydney', 19),  
    (12, 'Riti', 41, 'Delhi', 17),  
    (13, 'Aadi', 46, 'Mumbai', 11),  
    (14, 'Mohit', 45, 'Delhi', 15),  
    (15, 'Veena', 43, 'Delhi', 14),  
    (16, 'Shaunak', 42, 'Mumbai', 17),  
    (17, 'Manik', 42, 'Sydney', 14),  
    (18, 'Vikas', 42, 'Delhi', 11),  
    (19, 'Samir', 42, 'Mumbai', 15),  
    (20, 'Shobhit', 40, 'Sydney', 12)]  
# Create a DataFrame object  
df = pd.DataFrame(employees, columns=['ID', 'Name', 'Age', 'City', 'Experience'])  
df = df.set_index('ID')
```

```
# Display the DataFrame  
print("-"*40)  
print(df)  
print("-"*40)
```

```
#GroupByEx2.py  
import pandas as pd  
# List of Tuples  
employees = [  
    (11, 'Jack', 44, 'Sydney', 19),  
    (12, 'Riti', 41, 'Delhi', 17),  
    (13, 'Aadi', 46, 'Mumbai', 11),  
    (14, 'Mohit', 45, 'Delhi', 15),  
    (15, 'Veena', 43, 'Delhi', 14),  
    (16, 'Shaunak', 42, 'Mumbai', 17),
```

```

(17, 'Manik', 42, 'Sydney', 14 ),
(18, 'Vikas', 42, 'Delhi', 11 ),
(19, 'Samir', 42, 'Mumbai', 15 ),
(20, 'Shobhit', 40, 'Sydney', 12 )]

# Create a DataFrame object
df = pd.DataFrame(employees,
                  columns=['ID', 'Name', 'Age', 'City', 'Experience'])
df = df.set_index('ID')

# Display the DataFrame
print(df)
print("-----")
groupObj = df.groupby('City')
for grpName, rows in groupObj:
    print("Group Name: ", grpName)
    print('Group Content: ')
    print(rows)
print("-----")

#GroupByEx3.py
import pandas as pd
# List of Tuples
employees = [
    (11, 'Jack', 44, 'Sydney', 19),
    (12, 'Riti', 41, 'Delhi', 17),
    (13, 'Aadi', 46, 'Mumbai', 11),
    (14, 'Mohit', 45, 'Delhi', 15),
    (15, 'Veena', 43, 'Delhi', 14),
    (16, 'Shaunak', 42, 'Mumbai', 17),
    (17, 'Manik', 42, 'Sydney', 14),
    (18, 'Vikas', 42, 'Delhi', 11),
    (19, 'Samir', 42, 'Mumbai', 15),
    (20, 'Shobhit', 40, 'Sydney', 12 )]

# Create a DataFrame object
df = pd.DataFrame(employees,
                  columns=['ID', 'Name', 'Age', 'City', 'Experience'])
df = df.set_index('ID')

# Display the DataFrame
print(df)
print("*"*40)
print("-----")
groupObj = df.groupby('City')
for grpName, rows in groupObj:
    print("Group Name: ", grpName)
    print('Group Content: ')
    print(rows)
print("\n-----")

firstRowDf = df.groupby('City').first()
print(firstRowDf)
print("\n-----")
firstRowDf = df.groupby('City').nth(1)
print(firstRowDf)

```

```
print("\n-----")
print(df.groupby('City').size())
print("\n-----")
specificGroup = df.groupby('City').get_group('Mumbai')
print(specificGroup)
print("-----")


---


#GroupByEx11.py
import pandas as pd
# List of Tuples
employees = [
    (11, 'Jack', 44, 'Sydney', 19),
    (12, 'Riti', 41, 'Delhi', 17),
    (13, 'Aadi', 46, 'Mumbai', 11),
    (14, 'Mohit', 45, 'Delhi', 15),
    (15, 'Veena', 43, 'Delhi', 14),
    (16, 'Shaunak', 42, 'Mumbai', 17),
    (17, 'Manik', 42, 'Sydney', 14),
    (18, 'Vikas', 42, 'Delhi', 11),
    (19, 'Samir', 42, 'Mumbai', 15),
    (20, 'Shobhit', 40, 'Sydney', 12)]
# Create a DataFrame object
df = pd.DataFrame(employees, columns=['ID', 'Name', 'Age', 'City', 'Experience'])
df = df.set_index('ID')
# Display the DataFrame
print("-"*40)
print(df)
print("-"*40)
groupObj = df.groupby('City')
for grpname, row in groupObj:
    print("Group Name: {}".format(grpname))
    print(row)


---


#GroupByEx111.py
import pandas as pd
# List of Tuples
employees = [
    (11, 'Jack', 44, 'Sydney', 19),
    (12, 'Riti', 41, 'Delhi', 17),
    (13, 'Aadi', 46, 'Mumbai', 11),
    (14, 'Mohit', 45, 'Delhi', 15),
    (15, 'Veena', 43, 'Delhi', 14),
    (16, 'Shaunak', 42, 'Mumbai', 17),
    (17, 'Manik', 42, 'Sydney', 14),
    (18, 'Vikas', 42, 'Delhi', 11),
    (19, 'Samir', 42, 'Mumbai', 15),
    (20, 'Shobhit', 40, 'Sydney', 12)]
# Create a DataFrame object
df = pd.DataFrame(employees, columns=['ID', 'Name', 'Age', 'City', 'Experience'])
df = df.set_index('ID')
# Display the DataFrame
print("-"*40)
```

```
print(df)
print("-"*40)
groupObj = df.groupby('City')
for grpname,row in groupObj:
    print("Group Name:{} ".format(grpname))
    print(row)
print("*"*50)
# Get first row of each group
firstRowDf = df.groupby('City').first()
print(firstRowDf)
print("*"*50)
firstRowDf = df.groupby('City').nth(1)
print(firstRowDf)
print("*"*50)
# Get the size of DataFrame groups
print(df.groupby('City').size())
print("*"*50)
```

Regular Expressions in Python--3 Days

Index

- =>Purpose of Regular Expressions
 - =>Applications of Regular Expressions
 - =>Definition of Regular Expressions
 - =>Module Name (re) for dealing with Regular Expressions
 - =>Functions in re module
 - a) search()
 - b).findall()
 - c) finditer()
 - d) start()
 - e) end()
 - f) group()
 - =>Programming Examples

 - =>Programmer-Defined Classes in Regular Expressions
 - =>Programming Examples
 - =>Pre-Defined-Defined Classes in Regular Expressions
 - =>Programming Examples
 - =>Quantifiers in Regular Expressions
 - =>Programming Examples:

 - =>Combined Programming Examples on Programmer , Pre-Defined-Defined Classes and Quantifiers
-

Introduction to Regular Expressions in Python

- =>Regular Expressions is one of the Indepedent Concept from Proramming Languages.
 - =>The purpose of Regular Expressions in Python is that " To Build Robust Application by Validation of Data".
-

Applications of Regular Expressions

- =>Regular Expressions are used in development of Pattern Matching / text matching
 - =>Regular Expressions are used in development of Language Compilers and Interpreters.
 - =>Regular Expressions are used in development of Universal Protocols.
 - =>Regular Expressions are used in development of Electronic Cuircutary deisgnning
 - =>Regular Expressions are used in development of Customized data Validation / Extractions from
given data.....many more.
-

Definition of Regular Expressions:

=>A Regular Expression is one of the Search Pattern which is a combination of Alphabets,Digits and special symbols and it used to Search or Match or Find in given data and obtains Desired Result.

=>To deal with Regular Expressions programming / applications, we have a pre-defined module called "re".

===== Pre-defined Functions in re module =====

=>The 're' module contains the following essential Functions.

1) finditer():

Syntax:- varname=re.finditer("search-pattern","Given data")

=>here varname is an object of type <class,'Callable_Iterator'>

=>This function is used for searching the "search pattern" in given data iteratively and it returns table of entries which contains start index , end index and matched value based on the search pattern.

2) findall():

Syntax:- varname=re.findall("search-pattern","Given data")

=>here varname is an object of <class,'list'>

=>This function is used for searching the search pattern in entire given data and find all occurrences / matches and it returns all the matched values in the form an object <class,'list'> but not returning Start and End Index Values.

3) search():

Syntax:- varname=re.search("search-pattern","Given data")

=>here varname is an object of <class,'re.Match'> or <class,'NoneType'>

=>This function is used for searching the search pattern in given data for first occurrence / match only but not for other occurrences / matches.

=>if the search pattern found in given data then it returns an object of match class which contains matched value and start and end index values and it indicates search is successful.

=>if the search pattern not found in given data then it returns None which is type <class, "NoneType"> and it indicates search is un-successful

4) group():

=>This function is used obtaining matched value by the object of match class
=>This function present in match class of re module

Syntax:- varname=matchtabobj.group()

5) start():

=>This function is used obtaining starting index of matched value
=>This function present in match class of re module

Syntax: varname=matchobj.start()

6) end():

=>This function is used obtaining end index+1 of matched value
=>This function present in match class of re module

Syntax: varname=matchobj.end()

7) sub() Function

=> This function replaces the matches with the text of your choice:

```
import re
txt = "The rain in Spain"
x = re.sub("\s", "9", txt)
print(x)----- The9rain9in9Spain
```

#RegExpr1.py

```
import re
gd="Python is an oop lang. Python is also Func Prog lang"
sp="Python"
words=re.findall(sp,gd) # words= ["Python","Python"]
print(words, type(words)) # ['Python', 'Python'] <class 'list'>
print("\n{} time(s) '{}' available".format(len(words),sp))
```

#RegExpr2.py

```
import re
gd="Python is an oop lang. Python is also Func Prog lang"
sp="Python"
res=re.search(sp,gd) #
print("Type of res=",type(res)) # Success: <re.Match> Un-Success: NoneType
if(res!=None):
    print("Search is Sucessful")
    print("Satrtng Index={}".format(res.start()))
    print("End Index={}".format(res.end()))
    print("Matched Value={}".format(res.group()))
else:
    print("Search is Un-Sucessful")
    print("{} is not Found:".format(sp))
```

#RegExpr3.py

```
import re
gd="Python is an oop lang. Python is also Func Prog lang"
sp="python"
```

```

mattable=re.finditer(sp,gd)
print(type(mattable)) # <class 'callable_iterator'>
print("-"*60)
for mat in mattable: # here mat is in an object of <class, re.match>
    print("Starting Index:{} End Index:{}")
    Value:{}".format(mat.start(),mat.end(),mat.group()))


---


#RegExpr4.py
import re
gd="Python is an oop lang. Python is also Func Prog lang"
sp="Python"
mattable=re.finditer(sp,gd)
print(type(mattable)) # <class 'callable_iterator'>
print("-"*60)
cnt=0
for mat in mattable: # here mat is in an object of <class, re.match>
    print("Starting Index:{} End Index:{}")
    Value:{}".format(mat.start(),mat.end(),mat.group()))
    cnt=cnt+1
print("-"*60)
print("{} found {} time(s)".format(sp,cnt))
print("-"*60)


---


#program for search either 'a' or 'b' or 'c' only
#RegExpr5.py
import re
mat=re.finditer("[abc]","cAh#6K*3aMR9!bQT6%")
print("-"*50)
for m in mat:
    print("Start Index:{} End Index:{} Value:{}".format(m.start(),m.end(), m.group()))
print("-"*50)

```

E:\KVR-PYTHON-11AM\REG EXPR>py RegExpr5.py

```

Start Index:0 End Index:1 Value:c
Start Index:8 End Index:9 Value:a
Start Index:13 End Index:14 Value:b

```

```

#program for search for all except 'a' or 'b' or 'c'
#RegExpr6.py
import re
mat=re.finditer("[^abc]","cAh#6K*3aMR9!bQT6%")
print("-"*50)
for m in mat:
    print("Start Index:{} End Index:{} Value:{}".format(m.start(),m.end(), m.group()))
print("-"*50)

```

```
E:\KVR-PYTHON-11AM\REG EXPR>py RegExpr6.py
```

```
Start Index:1 End Index:2 Value:A
Start Index:2 End Index:3 Value:h
Start Index:3 End Index:4 Value:#
Start Index:4 End Index:5 Value:6
Start Index:5 End Index:6 Value:K
Start Index:6 End Index:7 Value:*
Start Index:7 End Index:8 Value:3
Start Index:9 End Index:10 Value:M
Start Index:10 End Index:11 Value:R
Start Index:11 End Index:12 Value:9
Start Index:12 End Index:13 Value:!
Start Index:14 End Index:15 Value:Q
Start Index:15 End Index:16 Value:T
Start Index:16 End Index:17 Value:6
Start Index:17 End Index:18 Value:%
```

```
#program for search for all lower case alphabets
#RegExpr7.py
import re
mat=re.finditer("[a-z]","cAh#6K*3aMR9!bQT6%")
noc=0
print("-"*50)
for m in mat:
    print("Start Index:{} End Index:{} Value:{}".format(m.start(),m.end(), m.group()))
    noc=noc+1
print("-"*50)
print("No. of small alphabets=",noc)
```

```
E:\KVR-PYTHON-11AM\REG EXPR>py RegExpr7.py
```

```
Start Index:0 End Index:1 Value:c
Start Index:2 End Index:3 Value:h
Start Index:8 End Index:9 Value:a
Start Index:13 End Index:14 Value:b
```

```
No. of small alphabets= 4
```

```
#program for search for all except lower case alphabets
#RegExpr8.py
import re
mat=re.finditer("[^a-z]","cAh#6K*3aMR9!bQT6%")
print("-"*50)
for m in mat:
    print("Start Index:{} End Index:{} Value:{}".format(m.start(),m.end(), m.group()))
```

```
    noc=noc+1
print("-"*50)
```

"""
E:\KVR-PYTHON-11AM\REG EXPR>py RegExpr8.py

```
Start Index:1 End Index:2 Value:A
Start Index:3 End Index:4 Value:#
Start Index:4 End Index:5 Value:6
Start Index:5 End Index:6 Value:K
Start Index:6 End Index:7 Value:*
Start Index:7 End Index:8 Value:3
Start Index:9 End Index:10 Value:M
Start Index:10 End Index:11 Value:R
Start Index:11 End Index:12 Value:9
Start Index:12 End Index:13 Value:!
Start Index:14 End Index:15 Value:Q
Start Index:15 End Index:16 Value:T
Start Index:16 End Index:17 Value:6
Start Index:17 End Index:18 Value:%
```

"""
#program for search for all Upper case alphabets only
#RegExpr9.py
import re
mat=re.finditer("[A-Z]","cAh#6K*3aMR9!bQT6%")
print("-"*50)
for m in mat:
 print("Start Index:{} End Index:{} Value:{}".format(m.start(),m.end(), m.group()))
print("-"*50)

"""
E:\KVR-PYTHON-11AM\REG EXPR>py RegExpr9.py

```
Start Index:1 End Index:2 Value:A
Start Index:5 End Index:6 Value:K
Start Index:9 End Index:10 Value:M
Start Index:10 End Index:11 Value:R
Start Index:14 End Index:15 Value:Q
Start Index:15 End Index:16 Value:T
```

"""
#program for search for all except Upper case alphabets
#RegExpr10.py
import re
mat=re.finditer("[^A-Z]","cAh#6K*3aMR9!bQT6%")
print("-"*50)
for m in mat:

```
    print("Start Index:{} End Index:{} Value:{}".format(m.start(),m.end(), m.group())))
print("-"*50)
```

:::::

E:\KVR-PYTHON-11AM\REG EXPR>py RegExpr10.py

```
Start Index:0 End Index:1 Value:c
Start Index:2 End Index:3 Value:h
Start Index:3 End Index:4 Value:#
Start Index:4 End Index:5 Value:6
Start Index:6 End Index:7 Value:*
Start Index:7 End Index:8 Value:3
Start Index:8 End Index:9 Value:a
Start Index:11 End Index:12 Value:9
Start Index:12 End Index:13 Value:!
Start Index:13 End Index:14 Value:b
Start Index:16 End Index:17 Value:6
Start Index:17 End Index:18 Value:%
```

:::::

```
#program for search for all Digits
#RegExpr11.py
import re
mat=re.finditer("[0-9]","cAh#6K*3aMR9!bQT6%")
print("-"*50)
for m in mat:
    print("Start Index:{} End Index:{} Value:{}".format(m.start(),m.end(), m.group()))
print("-"*50)
```

:::::

E:\KVR-PYTHON-11AM\REG EXPR>py RegExpr11.py

```
Start Index:4 End Index:5 Value:6
Start Index:7 End Index:8 Value:3
Start Index:11 End Index:12 Value:9
Start Index:16 End Index:17 Value:6
```

'''

```
#program for search for all except Digits
#RegExpr12.py
import re
mat=re.finditer("[^0-9]","cAh#6K*3aMR9!bQT6%")
print("-"*50)
for m in mat:
    print("Start Index:{} End Index:{} Value:{}".format(m.start(),m.end(), m.group()))
print("-"*50)
```

:::::

E:\KVR-PYTHON-11AM\REG EXPR>py RegExpr12.py

```
Start Index:0 End Index:1 Value:c
Start Index:1 End Index:2 Value:A
Start Index:2 End Index:3 Value:h
Start Index:3 End Index:4 Value:#
Start Index:5 End Index:6 Value:K
Start Index:6 End Index:7 Value:*
Start Index:8 End Index:9 Value:a
Start Index:9 End Index:10 Value:M
Start Index:10 End Index:11 Value:R
Start Index:12 End Index:13 Value:!
Start Index:13 End Index:14 Value:b
Start Index:14 End Index:15 Value:Q
Start Index:15 End Index:16 Value:T
Start Index:17 End Index:18 Value:%
```

```
#program for searches for all alphabets
#RegExpr13.py
import re
mat=re.finditer("[A-Za-z]","cAh#6K*3aMR9!bQT6%")
print("-"*50)
for m in mat:
    print("Start Index:{} End Index:{} Value:{}".format(m.start(),m.end(), m.group()))
print("-"*50)
```

```
E:\KVR-PYTHON-11AM\REG EXPR>py RegExpr13.py
```

```
Start Index:0 End Index:1 Value:c
Start Index:1 End Index:2 Value:A
Start Index:2 End Index:3 Value:h
Start Index:5 End Index:6 Value:K
Start Index:8 End Index:9 Value:a
Start Index:9 End Index:10 Value:M
Start Index:10 End Index:11 Value:R
Start Index:13 End Index:14 Value:b
Start Index:14 End Index:15 Value:Q
Start Index:15 End Index:16 Value:T
```

```
#program for searches for all except alphabets
#RegExpr14.py
import re
mat=re.finditer("[^A-Za-z]","cAh#6K*3aMR9!bQT6%")
print("-"*50)
for m in mat:
    print("Start Index:{} End Index:{} Value:{}".format(m.start(),m.end(), m.group()))
print("-"*50)
```

:::::

E:\KVR-PYTHON-11AM\REG EXPR>py RegExpr14.py

```
Start Index:3 End Index:4 Value:#  
Start Index:4 End Index:5 Value:6  
Start Index:6 End Index:7 Value:/*  
Start Index:7 End Index:8 Value:3  
Start Index:11 End Index:12 Value:9  
Start Index:12 End Index:13 Value:!  
Start Index:16 End Index:17 Value:6  
Start Index:17 End Index:18 Value:%
```

:::::

```
#program for searches for all Alphabets and Digits (Alpha nums)  
#RegExpr15.py  
import re  
mat=re.finditer("[A-Za-z0-9]","cAh#6K*3aMR9!bQT6%")  
print("-"*50)  
for m in mat:  
    print("Start Index:{} End Index:{} Value:{}".format(m.start(),m.end(), m.group()))  
print("-"*50)
```

:::::

E:\KVR-PYTHON-11AM\REG EXPR>py RegExpr15.py

```
Start Index:0 End Index:1 Value:c  
Start Index:1 End Index:2 Value:A  
Start Index:2 End Index:3 Value:h  
Start Index:4 End Index:5 Value:6  
Start Index:5 End Index:6 Value:K  
Start Index:7 End Index:8 Value:3  
Start Index:8 End Index:9 Value:a  
Start Index:9 End Index:10 Value:M  
Start Index:10 End Index:11 Value:R  
Start Index:11 End Index:12 Value:9  
Start Index:13 End Index:14 Value:b  
Start Index:14 End Index:15 Value:Q  
Start Index:15 End Index:16 Value:T  
Start Index:16 End Index:17 Value:6
```

:::::

```
#program for searching Special Symbols  
#RegExpr16.py  
import re  
mat=re.finditer("[^A-Za-z0-9]","cAh#6K*3aMR9!bQT6%")  
print("-"*50)  
for m in mat:  
    print("Start Index:{} End Index:{} Value:{}".format(m.start(),m.end(), m.group()))  
print("-"*50)
```

:::::

```
E:\KVR-PYTHON-11AM\REG EXPR>py RegExpr16.py
```

```
Start Index:3 End Index:4 Value:#  
Start Index:6 End Index:7 Value:*\nStart Index:12 End Index:13 Value:!  
Start Index:17 End Index:18 Value:%
```

===== Pre-Defined Character Classes in Reg Expr =====

=>Pre-Defined Character Classes are those which are already available in Python software and they are used for Preparing or Designing Search Pattern for searching in given data for obtaining Desired Result.

=>Syntax for Using Pre-Defined Character Classes
 " \Search Pattern "

=>The Following are Pre-Defined Character Classes.

- 1) \s----->Searches for Space Character only
 - 2) \S----->Searches for all except Space Character
 - 3) \d----->Searches for Digits only OR [0-9]
 - 4) \D----->Searches for all except Digits OR [^0-9]
 - 5) \w----->Searches for Word Character OR [A-Za-z0-9]
 - 6) \W----->Searches for Special Symbols OR [^A-Za-z0-9]
-

#program for searching Space Character only

```
#RegExpr17.py  
import re  
mat=re.finditer("\s"," cAh#6K *3aMR9!bQT6%")  
print("-"*50)  
for m in mat:  
    print("Start Index:{} End Index:{} Value:{}".format(m.start(),m.end(), m.group()))  
print("-"*50)
```

```
E:\KVR-PYTHON-11AM\REG EXPR>py RegExpr17.py
```

```
Start Index:0 End Index:1 Value:  
Start Index:7 End Index:8 Value:
```

#program for searching all except Space Character

```
#RegExpr18.py
import re
mat=re.finditer("\S"," cAh#6K *3aMR9!bQT6%")
print("-"*50)
for m in mat:
    print("Start Index:{} End Index:{} Value:{}".format(m.start(),m.end(), m.group()))
print("-"*50)
```

:::::

```
E:\KVR-PYTHON-11AM\REG EXPR>py RegExpr18.py
```

```
Start Index:1 End Index:2 Value:c
Start Index:2 End Index:3 Value:A
Start Index:3 End Index:4 Value:h
Start Index:4 End Index:5 Value:#
Start Index:5 End Index:6 Value:6
Start Index:6 End Index:7 Value:K
Start Index:8 End Index:9 Value:*
Start Index:9 End Index:10 Value:3
Start Index:10 End Index:11 Value:a
Start Index:11 End Index:12 Value:M
Start Index:12 End Index:13 Value:R
Start Index:13 End Index:14 Value:9
Start Index:14 End Index:15 Value:!
Start Index:15 End Index:16 Value:b
Start Index:16 End Index:17 Value:Q
Start Index:17 End Index:18 Value:T
Start Index:18 End Index:19 Value:6
Start Index:19 End Index:20 Value:%
```

:::::

```
#program for searching all digits
#RegExpr19.py
import re
mat=re.finditer("\d"," cAh#6K *3aMR9!bQT6%")
print("-"*50)
for m in mat:
    print("Start Index:{} End Index:{} Value:{}".format(m.start(),m.end(), m.group()))
print("-"*50)
```

:::::

```
E:\KVR-PYTHON-11AM\REG EXPR>py RegExpr19.py
```

```
Start Index:5 End Index:6 Value:6
Start Index:9 End Index:10 Value:3
Start Index:13 End Index:14 Value:9
Start Index:18 End Index:19 Value:6
```

:::::

```
#program for searching all except Digits
```

```
#RegExpr20.py
import re
mat=re.finditer("\D"," cAh#6K *3aMR9!bQT6%")
print("-"*50)
for m in mat:
    print("Start Index:{} End Index:{} Value:{}".format(m.start(),m.end(), m.group()))
print("-"*50)
```

```
E:\KVR-PYTHON-11AM\REG EXPR>py RegExpr20.py
```

```
Start Index:0 End Index:1 Value:
Start Index:1 End Index:2 Value:c
Start Index:2 End Index:3 Value:A
Start Index:3 End Index:4 Value:h
Start Index:4 End Index:5 Value:#
Start Index:6 End Index:7 Value:K
Start Index:7 End Index:8 Value:
Start Index:8 End Index:9 Value:*
Start Index:10 End Index:11 Value:a
Start Index:11 End Index:12 Value:M
Start Index:12 End Index:13 Value:R
Start Index:14 End Index:15 Value:!
Start Index:15 End Index:16 Value:b
Start Index:16 End Index:17 Value:Q
Start Index:17 End Index:18 Value:T
Start Index:19 End Index:20 Value:%
```

```
#program for searching all word Characters
#RegExpr21.py
import re
mat=re.finditer("\w"," cAh#6K *3aMR9!bQT6%")
print("-"*50)
for m in mat:
    print("Start Index:{} End Index:{} Value:{}".format(m.start(),m.end(), m.group()))
print("-"*50)
```

```
E:\KVR-PYTHON-11AM\REG EXPR>py RegExpr21.py
```

```
Start Index:1 End Index:2 Value:c
Start Index:2 End Index:3 Value:A
Start Index:3 End Index:4 Value:h
Start Index:5 End Index:6 Value:6
Start Index:6 End Index:7 Value:K
Start Index:9 End Index:10 Value:3
Start Index:10 End Index:11 Value:a
Start Index:11 End Index:12 Value:M
Start Index:12 End Index:13 Value:R
```

```
Start Index:13 End Index:14 Value:9
Start Index:15 End Index:16 Value:b
Start Index:16 End Index:17 Value:Q
Start Index:17 End Index:18 Value:T
Start Index:18 End Index:19 Value:6
```

```
#program for searching all Special Symbols Except word Characters
#RegExpr22.py
import re
mat=re.finditer("\W"," cAh#6K *3aMR9!bQT6%")
print("-"*50)
for m in mat:
    print("Start Index:{} End Index:{} Value:{}".format(m.start(),m.end(), m.group()))
print("-"*50)
```

```
E:\KVR-PYTHON-11AM\REG EXPR>py RegExpr22.py
```

```
Start Index:0 End Index:1 Value:
Start Index:4 End Index:5 Value:# 
Start Index:7 End Index:8 Value:
Start Index:8 End Index:9 Value:*
Start Index:14 End Index:15 Value:!
Start Index:19 End Index:20 Value:%
```

===== Quantifiers in Regular Expressions =====

=>The purpose of Quantifiers in Regular Expressions is that "To Search for zero or one more Values in given data".

=>Quantifiers are also used in Search Patterns to search in given data for obtaining desired result.

=>The essential Quantifiers in Regular Expressions are given bellow

1. k----->Searches for Exactly one 'k'
2. k+----->Searches for One k or More k's
3. k*----->Searches for either zero k or 1 k or more k's
4. k?----->Searches for either zero k or one k
5. . ----->Searches for all

Imp Special Points

-
- 1) \d+---->Searches one or more digits (OR) [0-9]+
 - 2) \ddd---->Searches for 3 Digit Number OR \d{3} \d{10} (OR) [0-9]{3}
 - 3) \d{n}---->Searches for N-Digit Number OR [0-9]{n}
 - 4) \d{n}.\d{m}--->Searches N-Digit Integer and M-Digit floating point value(Ex 34.56 56.78..)

5) \w{n}---->Searches n-length word OR [A-Za-z0-9]{n}
6) \w+---->Searches for either one word char or more word Characters.(OR) [A-Za-z0-9]+

7) \d{n,m }--->Searches for Min n-Digit Number and Maximum m-digit number
OR [0-9]{n,m}

```
#program for searching Exatctly One k
#RegExpr23.py
import re
mat=re.finditer("k","kkvkkvkkkvkv")
print("-"*50)
for m in mat:
    print("Start Index:{} End Index:{} Value:{}".format(m.start(),m.end(), m.group()))
print("-"*50)
```

"""
E:\KVR-PYTHON-11AM\REG EXPR>py RegExpr23.py

```
Start Index:0 End Index:1 Value:k
Start Index:1 End Index:2 Value:k
Start Index:3 End Index:4 Value:k
Start Index:4 End Index:5 Value:k
Start Index:6 End Index:7 Value:k
Start Index:7 End Index:8 Value:k
Start Index:8 End Index:9 Value:k
Start Index:10 End Index:11 Value:k
```

```
"""  
#program for searching one or more k's
#RegExpr24.py
import re
mat=re.finditer("k+","kkvkkvkkkvkv")
print("-"*50)
for m in mat:
    print("Start Index:{} End Index:{} Value:{}".format(m.start(),m.end(), m.group()))
print("-"*50)
```

"""
E:\KVR-PYTHON-11AM\REG EXPR>py RegExpr24.py

```
Start Index:0 End Index:1 Value:k
Start Index:2 End Index:4 Value:kk
Start Index:5 End Index:8 Value:kkk
Start Index:9 End Index:10 Value:k
```

```
"""  
#program for searching zero k or one k or more k's
#RegExpr25.py
import re
mat=re.finditer("k*","kkvkkvkkkvkv")
print("-"*50)
```

```
for m in mat:  
    print("Start Index:{} End Index:{} Value:{}".format(m.start(),m.end(), m.group()))  
print("-"*50)
```

"""

E:\KVR-PYTHON-11AM\REG EXPR>py RegExpr25.py

```
Start Index:0 End Index:1 Value:k  
Start Index:1 End Index:1 Value:  
Start Index:2 End Index:4 Value:kk  
Start Index:4 End Index:4 Value:  
Start Index:5 End Index:8 Value:kkk  
Start Index:8 End Index:8 Value:  
Start Index:9 End Index:10 Value:k  
Start Index:10 End Index:10 Value:  
Start Index:11 End Index:11 Value:
```

"""

```
#program for searching zero k or one k  
#RegExpr26.py  
import re  
mat=re.finditer("k?","kvkkvkkkvkv")  
print("-"*50)  
for m in mat:  
    print("Start Index:{} End Index:{} Value:{}".format(m.start(),m.end(), m.group()))  
print("-"*50)
```

"""

E:\KVR-PYTHON-11AM\REG EXPR>py RegExpr26.py

```
Start Index:0 End Index:1 Value:k  
Start Index:1 End Index:1 Value:  
Start Index:2 End Index:3 Value:k  
Start Index:3 End Index:4 Value:k  
Start Index:4 End Index:4 Value:  
Start Index:5 End Index:6 Value:k  
Start Index:6 End Index:7 Value:k  
Start Index:7 End Index:8 Value:k  
Start Index:8 End Index:8 Value:  
Start Index:9 End Index:10 Value:k  
Start Index:10 End Index:10 Value:  
Start Index:11 End Index:11 Value:
```

"""

```
#program for searching for all occurences  
#RegExpr27.py  
import re  
mat=re.finditer(".","kvkkvkkkvkv")  
print("-"*50)  
for m in mat:
```

```
    print("Start Index:{} End Index:{} Value:{}".format(m.start(),m.end(), m.group())))
print("-"*50)
```

"""

E:\KVR-PYTHON-11AM\REG EXPR>py RegExpr27.py

```
Start Index:0 End Index:1 Value:k
Start Index:1 End Index:2 Value:v
Start Index:2 End Index:3 Value:k
Start Index:3 End Index:4 Value:k
Start Index:4 End Index:5 Value:v
Start Index:5 End Index:6 Value:k
Start Index:6 End Index:7 Value:k
Start Index:7 End Index:8 Value:k
Start Index:8 End Index:9 Value:v
Start Index:9 End Index:10 Value:k
Start Index:10 End Index:11 Value:v
```

"

===== Programmer-Defined Character Classes in Reg Expr =====

=>Programmer-Defined Character Classes are those which defined by Programmers and they are for Preparing or Designing Search Pattern for searching in given data for obtaining Desired Result.

=>Syntax for Using Programmer-Defined Character Classes
" [Search Pattern] "

=>The Following are Programmer-Defined Character Classes.

1. [abc]---->Searches for either a or b or c
2. [^abc]---->Searches for all except a or b or c
3. [a-z]---->Searches for all Lower case alphabets only
4. [^a-z]---->Searches for all except Lower Case Alphabets
5. [A-Z]---->Searches for all Upper Case Alphabets
6. [^A-Z]---->Searches for all except Upper Case Alphabets
7. [0-9]---->Searches for all Digits only
8. [^0-9]---->Searches for all except Digits
9. [A-Za-z]---->Searches for all Alphabets (Lower and Upper) only
10. [^A-Za-z]---->Searches for all except Alphabets (Lower and Upper)

11. [A-Za-z0-9]-->Searches for all Alphabets and Digits(Alpha numberics--Word Character)
 12. [^A-Za-z0-9]-->Searches for all Special Symbols (ie Not Alphabets and Digits)
 13. [a-zA-Z0-9]---->Searches for lower alphabets and digits
 14. [^a-zA-Z0-9]----->Searches for all except lower alphabets and digits
 15. [A-Z0-9]---->Searches for all Upper case Alphabets and Digits
 16. [^A-Z0-9]---->Searches for all except Upper case Alphabets and Digits
-

```
#Program for validating and valid mailids from KBD
#MailldsValidation.py
import re
lst=[]
while(True):
    mail=input("Enter Mail Id:")
    res=re.search("\S+@\S+",mail)
    if(res!=None):
        lst.append(mail)
        ch=input("Do u want to insert another e-mail id(yes/no):")
        if(ch.lower()=="no"):
            break
    else:
        print("Invalid mail--try again")
print("Valid mails: {}".format(lst))


---


#Program for extracting mail-id from given string data
#MailsExamples1.py
import re
gd="Rossum mailid is rossum123@psf.com , Dennis mailid is dennis_c@bellland.co.in ,
Travis mailid is travis_numpy@numpy.org , Kinney mailid is kinney_pandas@pandas.net.in
and James mailid is james.java@sun.com"
sp="\S+@\S+"
mails=re.findall(sp,gd)
print("List of Mails:")
for mail in mails:
    print("\t{}".format(mail))


---


#Program for Extracting the Marks from given Data
#MarksListEx.py
import re
gd="Rossum got 22 marks , Dennis got 11 marks , Travis got 44 marks , Kinney got 55
marks and James got 999 marks"
sp="\d{2,3}"
markslist=re.findall(sp,gd)
for marks in markslist:
    print("\t{}".format(marks))


---


#Program for validating Mobile Number
```

```
#MobileNumberValidationEx1.py
import re
while(True):
    mno=input("Enter Mobile Number:")
    if(len(mno)==10):
        res=re.search("\d{10}",mno)
        if(res!=None):
            print("Ur Mobile Number is Valid:")
            break
        else:
            print("Invalid Mobile Number bcoz It contains non-numeric")
    else:
        print("Invalid Mobile Number bcoz length is >10--try again")
```

```
#Program for validating Mobile Number where it starts 98xxxxxxxx
```

```
#MobileNumberValidationEx2.py
import re
while(True):
    mno=input("Enter Mobile Number:")
    if(len(mno)==10):
        res=re.search("98\d{8}",mno)
        if(res!=None):
            print("Ur Mobile Number is Valid:")
            break
        else:
            print("Invalid Mobile Number bcoz It contains non-numeric")
    else:
        print("Invalid Mobile Number bcoz length is >10--try again")
```

```
#Program for Extracting the Names from given Data
```

```
#NamesListEx.py
import re
gd="Rossum got 22 marks , Dennis got 11 marks , Travis got 44 marks , Kinney got 55
marks and James got 99 marks"
sp="[A-Z][a-z]+"
print("-----By using findall()-----")
nameslist=re.findall(sp,gd)
print("Names of Student:")
for name in nameslist:
    print("\t{}".format(name))
print("-----")
print("-----By using finditer()-----")
matchnames=re.finditer(sp,gd)
print("Names of Student:")
for name in matchnames:
    print("Start Index:{} End Index:{}\nName:{}{}".format(name.start(),name.end(),name.group()))
print("-----")
```

```
#Program for extracting names and mail-ids from mailsdata.data file
```

```
#NamesMailsExamples2.py
import re
with open("mailsdata.data","r") as fp:
```

```

        filedata=fp.read()
        mails=re.findall("\S+@\S+",filedata)
        nameslist=re.findall("[A-Z][a-z]+",filedata)
        print("\tnames\tMails:")
        for names,mail in zip(nameslist,mails):
            print("\t{}\t{}".format(names,mail))


---


#Program for Extracting the Names and Marks from given Data
#NamesMarksListEx.py
import re
gd="Rossum got 22 marks , Dennis got 11 marks , Travis got 44 marks , Kinney got 55
marks and James got 999 marks"
sp1="\d{2,3}"
sp2="[A-Z][a-z]+"
markslist=re.findall(sp1,gd)
nameslist=re.findall(sp2,gd)
print("-"*50)
print("\tNames\tMarks")
print("-"*50)
for names,marks in zip(nameslist,markslist):
    print("\t{}\t{}".format(names,marks))
print("-"*50)


---


#Program extracting the names and marks from stud.info file
#studnamesmarksfiles.py
import re
with open("stud.info","r") as fp:
    filedata=fp.read()
    nameslist=re.findall("[A-Z][a-z]+",filedata)
    markslist=re.findall("\d{2,3}",filedata)
    print("-"*50)
    print("\tNames\tMarks")
    print("-"*50)
    for names,marks in zip(nameslist,markslist):
        print("\t{}\t{}".format(names,marks))
    print("-"*50)


---



```

=====

generator in python

=====

- =>generator is one of the function
- =>The generator function always contains yield keyword
- =>If the function contains return statement then it is called Normal Function
- =>If the function contains yield keyword then it is called generator
- =>Syntax:

```
def function_name(start,stop,step):
```

```
yield value
```

- =>The 'yield' key word is used for giving the value back to function call from function defintion and continue the function execution until condition becomes false.

=>The advantage of generators over functions concept is that it save lot of memory space in the case large sampling of data. In otherwords Functions gives all the result at once and it take more memory space where as generators gives one value at a time when programmer requested and takes minimized memory space.

```
#non-genex.py
r=range(1,2000)
print(r)
for v in r:
    print(v)
print("-----")
#GenEx1.py
def kvrrange(b,e):
    while(b<=e):
        yield b
        b=b+1

#main program
k=kvrrange(10,15) # here k is an object <class, generator>
while(True):
    try:
        print(next(k))
    except StopIteration:
        break
```

```
#GenEx2.py
def kvrrange(b,e,s):
    while(b<=e):
        yield b
        b=b+s

#main program
k=kvrrange(10,50,5) # here k is an object <class, generator>
while(True):
    try:
        print(next(k))
    except StopIteration:
        break
```

===== Decorators in Python =====

=>Decorator is one of the Function which will provides Additional Processing capability to the normal Function value and returns the modified value.

=>A Decorator Function is always takes Normal Function as parameter

Syntax:-

```
def functionname1( functionname ): # Decorator
    def innerfunctionname(): # Inner Function name
        val=functionname()
```

```

-----#
#do the operation on ' val '
-----
return resut # Inner Funtion must return modified value
return innerfunctionname # Decorator returns inner function name

```

=>here functionname1 is called Decorator function
=>here Functionname as a formal parameter . Every decorator function must take normal function as parameter.

```
#dece1.py
def getval(): # Normal Function defined by me and used by many programmers
    return float(input("Enter Any Numerical value:"))
```

```
def calsquare(gv): # Decorator
    def operation():
        n=gv()
        res=n**2
        return res
    return operation
```

```
def calsquareroot(hyd):# Decorator
    def op():
        val=hyd()
        res=val**0.5
        return res
    return op
```

```
#main program
opresult=calsquare( getval )
res=opresult()
print("Square=",res)
print("-----")
opres=calsquareroot(getval)
r=opres()
print("Square Root=",r)
```

```
#dece2.py
```

```
def calsquare(gv): # Decorator
    def operation():
        n=gv()
        res=n**2
        return res
    return operation
```

```
@calsquare
def getval(): # Normal Function defined by me and used by many programmers
    return float(input("Enter Any Numerical value:"))
```

```
#main program
```

```
res=getval()
print("Square =",res)
#decex3.py
```

```
def calsquareroot(gv): # Decorator
    def operation():
        n=gv()
        res=n**0.5
        return res
    return operation

@calsquareroot
def getval(): # Normal Function defined by me and used by many programmers
    return float(input("Enter Any Numerical value:"))
```

```
#main program
res=getval()
print("Square Root =",res)
#decex4.py
```

```
def calsquareroot(gv): # Decorator
    def operation():
        n=gv()
        res=n**0.5
        return res
    return operation
```

```
def calsquare(gv): # Decorator
    def operation():
        n=gv()
        res=n**2
        return res
    return operation
```

```
@calsquare
def getval1(): # Normal Function defined by me and used by many programmers
    return float(input("Enter Any Numerical value:"))
@calsquareroot
def getval2(): # Normal Function defined by me and used by many programmers
    return float(input("Enter Any Numerical value:"))
```

```
#main program
res=getval1()
print("Square =",res)
res=getval2()
print("Square Root =",res)
#non-decex1.py
```

```
def getval(): # Normal Function defined by me and used by many programmers
    return float(input("Enter Any Numerical value:"))
```

```
def calsquare(): # defined by Prog1
    n=getval()
    res=n**2
    print("Square=",res)

def calsquareroot(): # defined by Prog2
    n=getval()
    res=n**0.5
    print("Square Root=",res)

#main program
calsquare()
calsquareroot()
```

=====

Multi Threading in Python

=====

Index

=>Purpose of Multi Threading

=>Types of Applications

- a) Process Based Applications
- b) Thread Based Application

=>Module Name for Developing Thread Based Applications (threading)

=>Detailed Discussion of "threading module"

=>Number of approaches to develop Thread Based Applications.

- a) By using Functional Approach
- b) By Using Object Oriented Approach

=>Programming Examples

Dead Locks in Multi Threading.

=>Programming Examples.

=====

Introduction to Thread Based Applications

=====

=>The purpose of multi threading is that "To provide Concurrent / Simultaneous execution / Parallel Execution".

=>Concurrent Execution is nothing but executing the operations all at once.

=>The advantage of Concurrent execution is that to get less execution time.

=>If a Python Program contains multiple threads then it is called Multi Threading program.

=>Def. of thread:

=>A flow of Control is called thread.

=>The purpose of thread is that "To Perform certain operation whose logic developed in Functions / Methods concurrently."

=>By default Every Python contains Single Thread and whose name is "MainThread" and It provides Sequential Execution.

=>Programmatically, In a Python Program we can create multiple sub / Child threads and whose purpose is that "To execute operations whose logic is written in Functions / Methods Concurrently".

=>Hence Programmatically a Python Program contains two types of Threads. They are

- a) MainThread
- b) Sub / Child Threads

=>MainThread is created / Initiated by PVM ,when program execution starts and the role of mainThread is to execute main program statements and Monitor the execution status of Sub threads(if sub threads present).

=>The Sub / Child Threads always executes operations whose logic is written in Functions / Methods Concurrently".

=====X=====

=

Number of approaches to develop thread based Applications

=>In Python Programming, we have 2 approaches to develop thread based applications. They are

- 1) By Using Functional Approach
- 2) By using Object Oriented Approach

1) By Using Functional Approach

Steps:

1. import threading module
2. Define a Function, which is containing logic for Performing Operation by the sub thread.
3. Create a sub therad by using Thread class of threading module
4. Dispatch the sub thread for executing target function

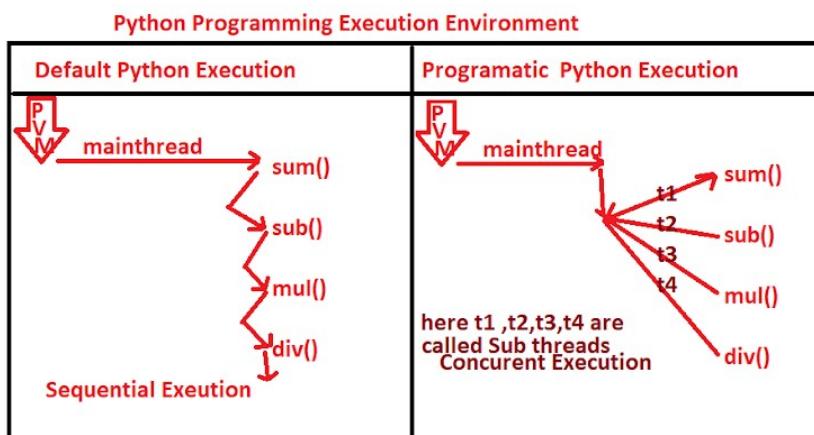
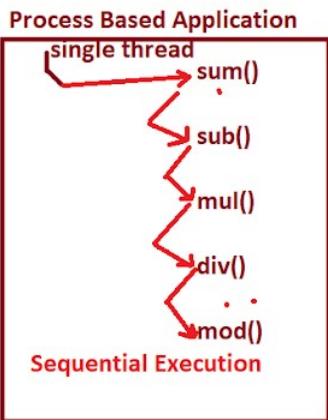
Example: Refer NumberGenEx1.py

2) By using Object Oriented Approach

Steps:

1. import threading module
2. Define a Programmer-Defined Class
3. Define Instance Method in a Class
4. Create a an object Programmer-Defined Class
5. Create a sub therad by using Thread class of threading module
6. Dispatch the sub thread for executing target function

Example: Refer NumberGenEx3.py



```

#non-threadingEx1.py
import threading,time
def findsquares(lst):
    for val in lst:
        print("5-->Therad Name:{}--"
>Square({})={}".format(threading.current_thread().name,val,val**2))
        time.sleep(1)

def findcubes(lst):
    for val in lst:
        print("10-->Therad Name:{}--"
>cubes({})={}".format(threading.current_thread().name,val,val**3))
        time.sleep(1)
    
```

```

#main program
bt=time.time()
print("Line-15->Default Name Thread in main program=",threading.current_thread().name)
lst=[12,5,6,10,23,-5,15]
findsquares(lst) # Function Call
print("\nMain Thread at Line: 18")
findcubes(lst)
et=time.time()
print("Total Time Taken by this Program={}".format(et-bt))


---


#WiththreadingEx1.py
import threading,time
def findsquares(lst):
    for val in lst:
        print("5-->Therad Name:{}--"
>Square({})={}".format(threading.current_thread().name,val,val**2))
        time.sleep(1)

def findcubes(lst):
    for val in lst:
        print("10-->Therad Name:{}--"
>cubes({})={}".format(threading.current_thread().name,val,val**3))
        time.sleep(1)

#main program
bt=time.time()
print("Line-15->Default Name Thread in main program=",threading.current_thread().name) #
Main Thread
print("Initial Number of Threads=",threading.active_count())
lst=[12,5,6,10,23,-5,15]
#create sub threads/ child threads
t1=threading.Thread(target=findsquares,args=(lst,) ) # Thread-1
t2=threading.Thread(target=findcubes,args=(lst,))# Thread-2
t1.name="ROssum"
t2.name="Travis"
t1.start()
t2.start()
print("Number of Threads=",threading.active_count())
t1.join()
t2.join()
print("Number of Threads=",threading.active_count())
et=time.time()
print("Total Time Taken by Threads Program={}".format(et-bt))


---


=====
```

==
Number of approaches for developing Thread Based Applications
=====

==
=>we can Develop thread based applications in 2 ways. They are
1. By using Functional Programming

2. By using Object Oriented Programming

1. By using Functional Programming

=>Steps:

1. Import threading Module and other modules if required
2. Define Programmer-Defined Function which contains logic to perform the task.
- 3.create sub thread(s)
4. dispatch the sub thread(s) by using start()

Examples:

Refer NumberGenEx1.py

1. By using Object Oriented Programming

=>Steps:

1. Import threading Module and other modules if required
2. Define Programmer-defined class name
3. Define Programmer-defined Method Name
4. Create Sub Thread(s)
5. Dispatch the sub threads by using start()

Examples: Refer NumberGenEx2.py

=====

Module Name used for Developing Thread Based Applications

=====

=>For Developing Thread Based Applications, we use a pre-defined module called "threading".

MODULE NAME : threading

Functions Names in "threading" module

1) current_thread():- This Function is used for obtaining Name of the thread.

Syntax: varname=threading.current_thread()

Examples:-

```
tname=threading.current_thread()  
print("Default Name of thread=",tname.name)
```

2) active_count(): This Function is used obtaining number of therads which are

Syntax: varname=threading.active_count()

Example

```
noat=threading.active_count()
print("Number of active threads=",noat) # 1
```

Class Name in "threading" module : "Thread"

Methods in Thread Class

1) Thread(target,args)

Syntax:-

varname=threading.Thread(target=FunctionName,args=(list of values if any)
=>here varname is an object of Thread class and treated as sub thread / child

thread.

=>Examples: t1=threading.Thread(target=findsquares,args=(lst,))
t2=threading.Thread(target=welcome)

Here t1 and t2 are called Sub Threads OR Child Threads.

2) is_alive()

=>This Function is used for checking whether the Sub thread is under execution or not.

=>If the sub thread is under running Process then it returns True otherwise it returns False.

=>Syntax: var name=threading.is_alive()

Examples: Refer ThreadEx2.py

3. start()

=>This Function is used for dispatching sub threads to the corresponding target function.

=>Syntax: subthreadobj.start()

=>Examples Refer ThreadEx2.py

4. setName(): It is used for setting the User-Friendly thread name for sub thread

syntax: subthreadobj.setName(" User-Friendly thread name")

Here setName() deprecated on the name of "name" attribute

Syntax: subthreadobj.name=User-Friendly thread name

Example: Refer ThreadEx3.py

5. getName()

=> It is used for getting the thread name

syntax: varname=threadobj.getName()

Here `getName()` deprecated on the name of "name" attribute

Syntax: `threadobj.name`

Example: Refer ThreadEx3.py

6) join():

=>This Function is used for making sub threads to join with main thread.

=>When subthreads object calls join() then main thread will wait until until sub threads complete their execution.

=> Syntax:- subthreadobj1.join()
 subthreadobj2.join()

subthreadobj-n.join()

Examples: Refer NumberGenEx1.py

```
#Program for finding default thread name
#ThreadEx1.py
import threading
tname=threading.current_thread()
print("Default Name of thread=",tname.name)

#OR

dfname=threading.current_thread().name
print("Default Name of thread=",dfname)
noat=threading.active_count()
print("Number of active threads=",noat) # 1

#Program for creating sub threads
#ThreadEx2.py
import threading
def welcome():
    print("\ni am from welcome() ")
    print("welcome() executed by:{} ".format(threading.current_thread().name))
```

```
# main program
print("Default Thread Name:{}".format(threading.current_thread().name))
t1=threading.Thread(target=welcome)#creating sub thread
es=t1.is_alive()
print("Is sub thread under execution before start():{}".format(es))#False
t1.start()
print("Is sub thread under execution after start():{}".format(t1.is_alive()))#True
#Program for creating sub threads and setting and getting the names
#ThreadEx3.py
import threading
def welcome():
    print("\ni am from welcome() ")
    print("welcome() executed by:{}".format(threading.current_thread().name))
```

```

# main program
print("Default Thread Name:{} ".format(threading.current_thread().name))
t1=threading.Thread(target=welcome)#creating sub thread
print("Default Name of Sub Thread=",t1.name) # Thread-1
#set programmer-defined name to sub thread by using setName()
t1.name="KVR"
print("Programmer-defined Name for Sub Thread=",t1.name) # KVR


---


#Write a python program which will generate 1 to n numbers after each and every second
#NumberGenEx1.py
import threading,time # Step-1
def numbergenerate(n): # Step-2
    print("Name of the thread in numbergenerate()=",threading.current_thread().name)
    if(n<=0):
        print("{} is invalid input:".format(n))
    else:
        print("-"*50)
        print("Numbers within:{} ".format(n))
        print("-"*50)
        for i in range(1,n+1):
            print("\t{}".format(i))
            time.sleep(1)
        print("-"*50)

#main program
n=int(input("Enter How Many Numbers u want to generate:"))
t1=threading.Thread(target=numbergenerate,args=(n,)) # Sub thread -- # Step-3
t1.start() # Step-4
print("Line-21-->Number of active threads=",threading.active_count())
t1.join()
print("Line-23-->Number of active threads after completion=",threading.active_count())


---


#Write a python program which will generate 1 to n numbers after each and every second
#NumberGenEx2.py
import threading,time # Step-1
def numbergenerate(): # Step-2
    n=int(input("Enter How Many Numbers u want to generate:"))
    print("Name of the thread in numbergenerate()=",threading.current_thread().name)
    if(n<=0):
        print("{} is invalid input:".format(n))
    else:
        print("-"*50)
        print("Numbers within:{} ".format(n))
        print("-"*50)
        for i in range(1,n+1):
            print("\t{}".format(i))
            time.sleep(1)
        print("-"*50)

#main program
t1=threading.Thread(target=numbergenerate) # Sub thread -- # Step-3

```

```
t1.start() # Step-4
print("Line-21-->Number of active threads=",threading.active_count())
t1.join()
print("Line-23-->Number of active threads after completion=",threading.active_count())


---


#Write a python program which will generate 1 to n numbers after each and every second
#NumberGenEx3.py
import threading,time # Step-1
class Numbers: # Step-2
    def generate(self,n): # Step-3
        print("Name of the thread in
generate()=",threading.current_thread().name)
        if(n<=0):
            print("{} is invalid input:".format(n))
        else:
            print("-"*50)
            print("Numbers within:{} ".format(n))
            print("-"*50)
            for i in range(n,0,-1):
                print("\t{}".format(i))
                time.sleep(1)
            print("-"*50)

#main program
no=Numbers()
t1=threading.Thread(target=no.generate,args=(int(input("Enter How Many Numbers u want
to generate:"))),) # Step-4
t1.name="KVR"
t1.start() # Step-5


---


#write a thread based application which will generate even number and odd number
separately by using multiple threads
#EvenOddThreadsEx1.py
import threading,time
def even(n):
    for val in range(2,n+1,2):
        print("{}-->{}".format(threading.current_thread().name,val))
        time.sleep(1)
def odd(n):
    for val in range(1,n+1,2):
        print("{}-->{}".format(threading.current_thread().name,val))
        time.sleep(1)

#main program
n=int(input("Enter How Many Even and Odd Number u want:"))
t1=threading.Thread(target=even, args=(n,))
t1.name="EvenThread"
t2=threading.Thread(target=odd, args=(n,))
t2.name="OddThread"
t1.start()
t2.start()
```

```

#write a thread based application which will generate even number and odd number
separately by using multiple threads
#EvenOddThreadsEx2.py
import threading,time
class EvenNumbers:
    def __init__(self,n):
        self.n=n
    def even(self):
        for val in range(2,self.n+1,2):
            print(" {}---> {}".format(threading.current_thread().name,val))
            time.sleep(1)
class OddNumbers:
    def __init__(self,n):
        self.n=n
    def odd(self):
        for val in range(1,self.n+1,2):
            print(" {}---> {}".format(threading.current_thread().name,val))
            time.sleep(1)

```

```

#main program
n=int(input("Enter How Many Even and Odd Number u want:"))
eo=EvenNumbers(n) # Object creation and calling Parameterized Const
od=OddNumbers(n) # Object creation and calling Parameterized Const
t1=threading.Thread(target=eo.even)
t1.name="EvenThread"
t2=threading.Thread(target=od.odd)
t2.name="OddThread"
t1.start()
t2.start()

```

#write a thread based application which will generate even number and odd number
separately by using multiple threads

```

#EvenOddThreadsEx3.py
import threading,time
class EvenNumbers:
    def __init__(self,n):
        self.n=n
    def even(self):
        for val in range(2,self.n+1,2):
            print(" {}---> {}".format(threading.current_thread().name,val))
            time.sleep(1)
class OddNumbers:
    def __init__(self,n):
        self.n=n
    def odd(self):
        for val in range(1,self.n+1,2):
            print(" {}---> {}".format(threading.current_thread().name,val))
            time.sleep(1)

```

```

#main program
n=int(input("Enter How Many Even and Odd Number u want:"))

```

```
t1=threading.Thread(target=EvenNumbers(n).even)
t1.name="EvenThread"
t2=threading.Thread(target=OddNumbers(n) .odd)
t2.name="OddThread"
t1.start()
t2.start()
```

Synchronization in Multi Threading
(OR)
Locking concept in Threading

=>When multiple threads are operating / working on the same resource(function / method) then by default we get dead lock result / race condition / wrong result / non-thread safety result.

=>To overcome this dead lock problems, we must apply the concept of Synchronization

=>The advantage of synchronization concept is that to avoid dead lock result and provides Thread Safety Result.

=>In Python Programming, we can obtain synchronization concept by using locking and un-locking concept.

=>Steps for implementing Synchronization Concept:

(OR)
Steps for avoiding dead lock

1) obtain / create an object of Lock class, which is present in threading module.

Syntax:-

lockobj=threading.Lock()

2) To obtain the lock on the sharable resource, we must use acquire()

Syntax:

lockobj.acquire()

Once current object acquire the lock, other thread objects are made wait until current thread object releases the lock.

3) To un-lock the sharable resource/current object, we must use release()

Syntax:

lockobj.release()

Once current object releases the lock, other objects are permitted into sharable resource. This process of aquiring and releasing the lock will be continued until all the objects completed their execution.

#MulTablesFunNonSyncEx1.py

import threading,time

```

def multable(n):
    if(n<=0):
        print("{} is invalid Input".format(n))
    else:
        print("-"*50)
        print("\tMul Table for:{} by thread")
        Name:{}".format(n,threading.current_thread().name))
        print("-"*50)
        for i in range(1,11):
            print("\t{} x {}={}".format(n,i,n*i))
            time.sleep(1)
        else:
            print("-"*50)

#main program
t1=threading.Thread(target=multable,args=(10,))
t2=threading.Thread(target=multable,args=(12,))
t3=threading.Thread(target=multable,args=(19,))
t4=threading.Thread(target=multable,args=(2,))
t1.start()
t2.start()
t3.start()
t4.start()


---


#MulTablesOopsNonSyncEx1.py
import threading,time
class Tables:
    def multable(self,n):
        if(n<=0):
            print("{} is invalid Input".format(n))
        else:
            print("-"*50)
            print("\tMul Table for:{} by thread")
            Name:{}".format(n,threading.current_thread().name))
            print("-"*50)
            for i in range(1,11):
                print("\t{} x {}={}".format(n,i,n*i))
                time.sleep(1)
            else:
                print("-"*50)

#main program
t1=threading.Thread(target=Tables().multable,args=(10,))
t2=threading.Thread(target=Tables().multable,args=(12,))
t3=threading.Thread(target=Tables().multable,args=(19,))
t4=threading.Thread(target=Tables().multable,args=(2,))
t1.start()
t2.start()
t3.start()
t4.start()


---


#MulTablesFunWithSyncEx1.py

```

```

import threading,time
def multable(n):
    #Obtain the lock
    L.acquire()
    if(n<=0):
        print("{} is invalid Input".format(n))
    else:
        print("-"*50)
        print("\tMul Table for:{} by thread"
Name:{}".format(n,threading.current_thread().name))
        print("-"*50)
        for i in range(1,11):
            print("\t{} x {}={}".format(n,i,n*i))
            time.sleep(1)
        else:
            print("-"*50)
    #release the Lock
    L.release()

#main program
L=threading.Lock() # Step-1--Creating an object of Lock class
#creating Multiple thrads with same target resource
t1=threading.Thread(target=multable,args=(10,))
t2=threading.Thread(target=multable,args=(-12,))
t3=threading.Thread(target=multable,args=(19,))
t4=threading.Thread(target=multable,args=(2,))
t1.start()
t2.start()
t3.start()
t4.start()


---


#MulTablesOopsWithSyncEx1.py
import threading,time
class Tables:
    def multable(self,n):
        L.acquire()
        if(n<=0):
            print("{} is invalid Input".format(n))
        else:
            print("-"*50)
            print("\tMul Table for:{} by thread"
Name:{}".format(n,threading.current_thread().name))
            print("-"*50)
            for i in range(1,11):
                print("\t{} x {}={}".format(n,i,n*i))
                time.sleep(1)
            else:
                print("-"*50)
        L.release()

#main program
L=threading.Lock()
t1=threading.Thread(target=Tables().multable,args=(10,))
```

```
t2=threading.Thread(target=Tables().multable,args=(12,))
t3=threading.Thread(target=Tables().multable,args=(19,))
t4=threading.Thread(target=Tables().multable,args=(2,))
t1.start()
t2.start()
t3.start()
t4.start()


---


#MulTablesOopsWithSyncEx2.py
import threading,time
class Tables:
    L=threading.Lock() # class level Data Member
    def multable(self,n):
        Tables.L.acquire()
        if(n<=0):
            print("{} is invalid Input".format(n))
        else:
            print("-"*50)
            print("\tMul Table for:{} by thread"
Name:{}".format(n,threading.current_thread().name))
            print("-"*50)
            for i in range(1,11):
                print("\t{} x {}={}".format(n,i,n*i))
                time.sleep(1)
            else:
                print("-"*50)
        Tables.L.release()


---


#main program
t1=threading.Thread(target=Tables().multable,args=(10,))
t2=threading.Thread(target=Tables().multable,args=(-12,))
t3=threading.Thread(target=Tables().multable,args=(19,))
t4=threading.Thread(target=Tables().multable,args=(-2,))
t1.start()
t2.start()
t3.start()
t4.start()


---


#MulTablesOopsWithSyncEx3.py
import threading,time
class Tables:
    L=threading.Lock() # class level Data Member
    def multable(self,n):
        self.L.acquire()
        if(n<=0):
            print("{} is invalid Input".format(n))
        else:
            print("-"*50)
            print("\tMul Table for:{} by thread"
Name:{}".format(n,threading.current_thread().name))
            print("-"*50)
            for i in range(1,11):
                print("\t{} x {}={}".format(n,i,n*i))
```

```

                time.sleep(1)
            else:
                print("-"*50)
        self.L.release()

#main program
t1=threading.Thread(target=Tables().multable,args=(10,))
t2=threading.Thread(target=Tables().multable,args=(-12,))
t3=threading.Thread(target=Tables().multable,args=(19,))
t4=threading.Thread(target=Tables().multable,args=(-2,))
t1.start()
t2.start()
t3.start()
t4.start()


---


#MulTablesOopsWithSyncEx4.py
import threading,time
class Tables:
    @classmethod
    def getLock(cls):
        cls.lck=threading.Lock() # class level Data Member

    def multable(self,n):
        Tables.lck.acquire()
        if(n<=0):
            print("{} is invalid Input".format(n))
        else:
            print("-"*50)
            print("\tMul Table for:{} by thread"
Name:{}".format(n,threading.current_thread().name))
            print("-"*50)
            for i in range(1,11):
                print("\t{} x {}={} ".format(n,i,n*i))
                time.sleep(1)
            else:
                print("-"*50)
        Tables.lck.release()

#main program
Tables.getLock()
t1=threading.Thread(target=Tables().multable,args=(10,))
t2=threading.Thread(target=Tables().multable,args=(-12,))
t3=threading.Thread(target=Tables().multable,args=(19,))
t4=threading.Thread(target=Tables().multable,args=(-2,))
t1.start()
t2.start()
t3.start()
t4.start()


---


#MalyaOopsWithSyncEx4.py
import threading
class Malya:
    def __init__(self):
        self.amount=10000

```

```

def giveMoney(self,custamt):
    L.acquire()
    print("\nCustomer Name:{} ".format(threading.current_thread().name))
    if(custamt<=self.amount):
        self.amount-=self.amount-custamt
        print("{} got Money from
Mayla".format(threading.current_thread().name))
        print("Remaining amount in Malya
Account:{} ".format(self.amount))
    else:
        print("{} ,Check Bounced and contact
Malya:".format(threading.current_thread().name))
    L.release()

#main program
L=threading.Lock()
m=Malya()
t1=threading.Thread(target=m.giveMoney,args=(10001,))
t2=threading.Thread(target=m.giveMoney,args=(9000,))
t3=threading.Thread(target=m.giveMoney,args=(10000,))
t4=threading.Thread(target=m.giveMoney,args=(1000,))
t1.start()
t2.start()
t3.start()
t4.start()


---


#Reservation.py
import threading,time
class Train:
    def __init__(self,seats):
        self.seats=seats
        self.L=threading.Lock()

    def reservation(self,pseats):
        self.L.acquire()
        if(pseats>self.seats):
            print("Dear Passenger: {}, {} are not
available:".format(threading.current_thread().name,pseats))
        else:
            self.seats=self.seats-pseats
            print("Dear Passenger: {}, {} are
Reserved:".format(threading.current_thread().name,pseats))
            print("Remaining Seats in Train: {}".format(self.seats))
            time.sleep(1)
        self.L.release()

#main program
t=Train(10)
p1=threading.Thread(target=t.reservation,args=(14,))
p1.name="Ramesh"
p2=threading.Thread(target=t.reservation,args=(14,))

```

```
p2.name="Rajesh"
p3=threading.Thread(target=t.reservation,args=(5,))
p3.name="Rossum"
p4=threading.Thread(target=t.reservation,args=(2,))
p4.name="Sheela"
p1.start()
p2.start()
p3.start()
p4.start()
```

===== random module =====

=>random one of pre-defined module present in python
=>The purpose of random is that "To generate random values in various contexts".
=>random module contains the following essential functions.

a) randrange()
b) randint()

c) random()
d) uniform()

e) choice()
f) shuffle()
g) sample()

===== a) randrange() =====

=>This function is used for generating random integer values between specified limits.
Syntax1:- random.randrange(Value)

 This syntax generates any random value between 0 to Value-1

Syntax-2: random.randrange(start,stop)
 This syntax generates any random value between start to stop-1

Examples:

```
>>> import random
>>> print(random.randrange(100,150))---133
>>> print(random.randrange(100,150))---121
>>> print(random.randrange(100,150))---139
>>> print(random.randrange(100,150))---143
>>> print(random.randrange(100,150))---106
>>> print(random.randrange(100,150))---133
>>> print(random.randrange(10))---5
>>> print(random.randrange(10))---9
```

```
=====  
#randrangeex.py
import random
```

```
for i in range(1,6):
```

```
    print(random.randrange(10))
```

```
print("-----")
```

```
for i in range(1,6):
```

```
    print(random.randrange(1000,1100))
```

```
print("-----")
```

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

b) randint():

=>Syntax:- random.randint(start,stop)

=>This syntax generates any random value between start to stop. Here start and stop are inclusive.

Examples:

```
>>> print(random.randint(10,15))----10
```

```
>>> print(random.randint(10,15))----13
```

```
>>> print(random.randint(10,15))----14
```

```
>>> print(random.randint(10,15))----11
```

```
>>> print(random.randint(10,15))----15
```

```
#randintex.py
```

```
import random
```

```
for i in range(1,6):
```

```
    print(random.randint(10,20))
```

```
print("-----")
```

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

c) random()

=>Syntax:- random.random()

=>This syntax generates floating point random values between 0.0 and 1.0 (Exclusive))

Examples:

```
>>> import random
```

```
>>> print(random.random())-----0.1623906138450063
```

```
>>> print(random.random())-----0.15382209709271966
```

```
>>> print(random.random())-----0.09542283007844476
```

```
>>> print(random.random())-----0.6134301633766425
```

```
#randomex.py
```

```
import random
```

```
lst=[]
```

```
for i in range(1,6):
```

```
    lst.append("%0.2f" %random.random())
```

```
print("-----")
```

```
print("Content of lst={}".format(lst))
```

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

d) uniform()

Syntax:- random.uniform(start,stop)

=>This generates random floating point values from start to stop-1 values

=>The values of start and stop can both Integer or floating point values.

Examples:

```
-----  
>>> import random  
>>> print(random.uniform(10,15))----- 14.416746067678286  
>>> print(random.uniform(10,15))----13.2420406264978  
>>> print(random.uniform(10,15))----11.716110933506432  
>>> print(random.uniform(10,15))-----10.703499588966528  
>>> print(random.uniform(10,15))----11.306226559323017  
>>> print(random.uniform(10.75,15.75))-----13.939787347170148  
>>> print(random.uniform(10.75,15.75))---10.760428232717597  
-----
```

```
#uniformex.py  
import random  
lst=[]  
for i in range(1,6):  
    lst.append(float("%0.2f" %random.uniform(10,15.5)))  
print("-----")  
print("Content of lst={}".format(lst))  
=====X=====  
e) choice():  
-----
```

Syntax:- `random.choice(Iterable_object)`

=>This function obtains random values from Iterable_object.

EXAMPLES:

```
-----  
>>>  
print(random.choice([10,20,30,40,50]),random.choice("PYTHON"),random.choice(range(10,  
15)))--40 T 11  
>>>  
print(random.choice([10,20,30,40,50]),random.choice("PYTHON"),random.choice(range(10,  
15)))-----30 P 12  
>>>  
print(random.choice([10,20,30,40,50]),random.choice("PYTHON"),random.choice(range(10,  
15)))-----40 N 12  
-----
```

```
#choiceex.py  
import random  
s="AaBRe#%^@8YuQLPau*&"  
for i in range(1,6):  
    print(random.choice(s),random.choice(s),random.choice(s),random.choice(s))  
=====X=====
```

f) shuffle():

=>This Function is used for re-organizing the elements of any mutable object but not on immutable object.

Syntax:- `random.shuffle(list)`

=>We can shuffle the data of list but not other objects of Data Types

Examples:

```
>>> d={10:"cadburry",20:"kitkat",30:"malkybar", 40:"dairymilk"}
>>> print(d)---{10: 'cadburry', 20: 'kitkat', 30: 'malkybar', 40: 'dairymilk'}
>>> for k,v in d.items():
...     print(k,"--",v)
...
    10 -- cadburry
    20 -- kitkat
    30 -- malkybar
    40 -- dairymilk
>>> import random
>>> print(random.shuffle(d))----Traceback (most recent call last):
                                                File "<stdin>", line 1, in
<module>
                                                File
"C:\Users\nareshit\AppData\Local\Programs\Python\Python310\lib\random.py", line 394, in
shuffle
                                                x[i], x[j] = x[j], x[i]
                                                KeyError: 3
>>> s={10,20,30,40,50}
>>> print(random.shuffle(s))
                                                Traceback (most recent call last):
                                                File "<stdin>", line 1, in <module>
                                                File
"C:\Users\nareshit\AppData\Local\Programs\Python\Python310\lib\random.py", line 394, in
shuffle
                                                x[i], x[j] = x[j], x[i]
                                                TypeError: 'set' object is not
subscriptable
>>> t=(10,20,30,40,50)
>>> print(random.shuffle(t))
                                                Traceback (most recent call last):
                                                File "<stdin>", line 1, in <module>
                                                File
"C:\Users\nareshit\AppData\Local\Programs\Python\Python310\lib\random.py", line 394, in
shuffle
                                                x[i], x[j] = x[j], x[i]
                                                TypeError: 'tuple' object does not
support item assignment
>>> l=[10,20,30,40,50]
>>> print(random.shuffle(l))----None
>>> print(l)-----[30, 40, 50, 10, 20]
>>> random.shuffle(l)
>>> print(l)-----[40, 30, 10, 20, 50]
>>> random.shuffle(l)
>>> print(l)-----[40, 10, 50, 20, 30]
>>> random.shuffle(l)
>>> print(l)-----[30, 50, 20, 40, 10]
```

```
#shuffleex.py
import random as r
l=[10,"Python","Rossum",34.56,True]
for i in range(1,6):
    r.shuffle(l)
    print("content of l=",l)
=====X=====
g) sample()
```

=>This Function is used for selecting random samples from any Iterable object based on number of samples(+ve)

Syntax:- random.sample(iterable_object, k)
=>Here 'k' can be number of samples.

Examples:

```
>>> import random
>>> s="ABCabcERTYUertyu$%^&*#@!%^&ghjkiyl"
>>> print(random.sample(s,5))-----['A', '*', '^', 'j', 't']
>>> print(random.sample(s,5))-----['%', 'l', 'b', 'C', 'y']
>>> print(random.sample(s,5))-----['%', 'e', 'Y', 'j', 'u']
>>> print(random.sample(s,5))-----['y', 'E', '&', '$', '#']
>>> print(random.sample(s,5))-----['j', '*', 't', '$', 'u']
```

```
#sampleex.py
import random
lst=[10,"Rossum","Python",34.56,True]
for i in range(1,6):
    print(random.sample(lst,2))
=====X=====
```

```
#program for demostrating choice()
#choiceex1.py
import random as r
s="python"
for i in range(1,6):
    print(r.choice(s))
```

```
#program for demostrating choice()
#choiceex2.py
import random as r
text="ABCDEFGHIJKLMNPQRSTUVWXYZ"
digits="0123456789"
ss="~!@#$%^&*()_+"
small="abcdefghijklmnoprstuvwxyz"
for i in range(1,6):
    print(r.choice(text),r.choice(digits),r.choice(ss),r.choice(small))
```

```
#program for demostrating choice()
#choiceex3.py
import random as r
text="ABCDEFGHIJKLMNPQRSTUVWXYZ"
```

```
digits="0123456789"
for i in range(1,11):
    print("TS09"+r.choice(text)+r.choice(text)+r.choice(digits)+r.choice(digits)+r.choice(digits)+r.choice(digits))


---


#program for demostrating randint()
#RandintEx.py
import random as r
for i in range(1,6):
    print(r.randint(3,10))


---


#program for demostrating randint()
#RandintEx1.py
import random as r
for i in range(1,6):
    print(r.randint(3,10))


---


#program for demostrating randint()
#RandintEx2.py
import random as r
for i in range(1,6):
    print(r.randint(1000,9999))


---


#program for demostrating random()
#randomex1.py
import random as r
for i in range(1,6):
    print(r.random())


---


#program for demostrating random()
#randomex2.py
import random as r
for i in range(1,6):
    print(round(r.random(),2))


---


#program for demostrating randrange()
#randrangeex1.py
import random as r
for i in range(1,6):
    print(r.randrange(3,10))


---


#program for demostrating randrange()
#randrangeex2.py
import random as r
for i in range(1,6):
    print(r.randrange(10000,100000))


---


#program for demostrating sample()
#sampleex1.py
import random as r
s="MISSISSIOPPI"
for i in range(1,6):
    print(r.sample(s,3))


---


#program for demostrating sample()
#sampleex2.py
import random as r
s="MISSISSIOPPI"
```

```
for i in range(1,6):
    l=r.sample(s,3)
    k=""
    k=k.join(l)
    print(k)


---


#program for demostrating choice()
#sampleex3.py
import random as r
text="ABCDEFGHIJKLMNPQRSTUVWXYZ"
digits="0123456789"
for i in range(1,11):
    print("TS08",r.sample(text,2),r.sample(digits,4))


---


#program for demostrating choice()
#sampleex4.py
import random as r
text="ABCDEFGHIJKLMNPQRSTUVWXYZ"
digits="0123456789"
for i in range(1,11):
    ap=r.sample(text,2)
    dg=r.sample(digits,4)
    ap=ap+dg
    k=""
    k=k.join(ap)
    print("TS08",k)


---


#program for demostrating choice()
#SBIACNO.py
import random as r
for i in range(1000,1020):
    print("SBI000",i)


---


#program for demostrating shuffle()
#shuffleex1.py
import random as r
lst=[10,"RS",23.45,True,2+3j]
for i in range(1,6):
    r.shuffle(lst)
    print(lst)


---


#program for demostrating shuffle()
#shuffleex2.py
import random as r
s="MISSISSIPPI"
lst=list(s)
for i in range(1,6):
    r.shuffle(lst)
    print(lst)


---


#program for demostrating shuffle()
#shuffleex3.py
import random as r
s="MISSISSIOPPI"
print("Given Data:",s)
```

```
print("Other Sufflings:")
lst=list(s)
for i in range(1,12):
    r.shuffle(lst)
    k=""
    k=k.join(lst)
    print(k)


---


#program for demoingraining uniform()
#uniformex1.py
import random as r
for i in range(1,6):
    print(r.uniform(100.5,101.5))


---


#program for demoingraining uniform()
#uniformex2.py
import random as r
for i in range(1,6):
    print(round(r.uniform(100,200),3))
```

=====

Introduction to Network Programming

=====

=>The purpose of Network Programming is that "To share the data between multiple remote Machines which are located across the world".

=>Def. of Network: A Network is a collection of Inter-connected Autonomous
----- Computers connected with Server

=>In Network Programming, we can develop two types of Programs. They are

1. Server Side Application / Program
 2. Client Side Application / Program
-

Server Side Application / Program

=>A Server Side Application / Program is one, which always receives request from client side program, Process the request and gives response to client side Program.

Client Side Application / Program

=>A Client Side Application / Program is one which always makes a request to Server Side Program and obtains Services / Response from Server Side Program.

-
Definition of DNS (Domain Naming Service):

=DNS is one of the Physical Machine where Server Side Program Resides.

=>The default DNS of every computer is "localhost"

Definition of IP Address (Internet Protocol Address):

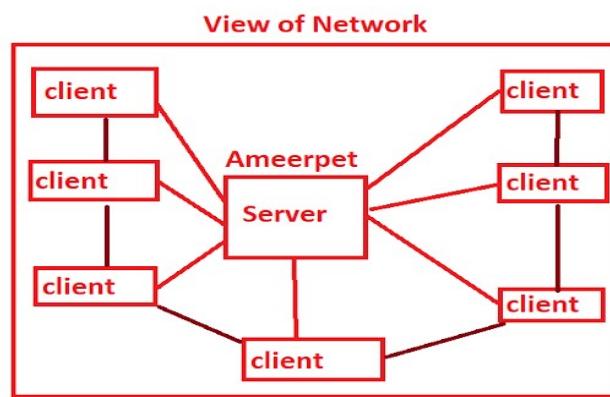
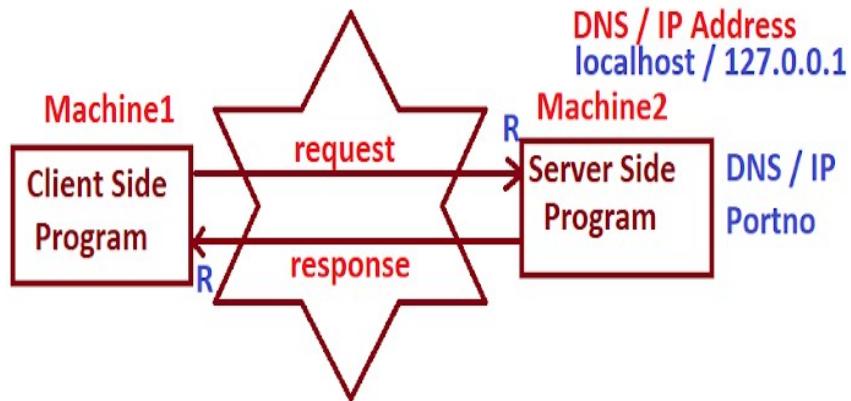
=>IP Address is one of the Numerical value of the Physical Machine where Server

Side Program Resides.

=>The default IP Address of every computer is "127.0.0.1"(loop back address)

Definition of Portno

=>A Port Number is one of the Logical Numerical Id where server side program is running.



Developing Server and Client Side Applications

Steps for Developing Server Side Applications:

1. import socket module
 2. Every Server Side Program must BIND with DNS/ IP Address and Portno.
 3. Every Server Side Program must be configured in such way that to how many client side programs it can provide services.
 4. Every Server Side Program must ACCEPT the request from Client Side Program.
 5. Server Side Program must READ the requested data of Client Side Program.
 6. Server Side Program must PROCESS the client side program request and gives RESPONSE to Client Side Program.
 7. Repeat step-(4)(5) and(6) until Client Side Propgram stops sending Requests.
-

Steps for Developing Client Side Applications

1. import socket module
 2. Every Client Side Program must CONNECT to Server Side Program by passing (DNS / IP Address, portno)
 3. Every Client Side Program must SEND Request to Server Side Program
 4. Every Client Side Program must RECEIVE the response from Server Side Program.
 5. Client Side Program can repeat Step-(3) and (4) until Client Side Program completed its number of requests.
-

Module Name for Developing Networking Applications

=>The Module Name for Developing Networking Applications is "socket".
=>In General, socket is one of the Bi-Directional Communication Entity Between multiple Devices

1) socket()

=>Syntax: varname=socket.socket()
=>Here varname is an object of <class 'socket'>
=>This Function is used for creating an object socket class at both Server and Client Side Programs.
=>Examples:- s=socket.socket()

2) bind()

=>Syntax: - serversocketobj.bind(("DNS/IPAddr",portno))
=>This Function is used for making the Server Side Program and binding at Certain machine (DNS / IP Addr) and at certain Port number.
=>Examples: s.bind(("localhost",8888))

```
(OR)  
s.bind( "127.0.0.1",8888 )
```

3) listen()

=>Syntax: serversocketobj.listen(number of clients)

=>This Function is used for configuring the server side program in such a way that to how many clients The server side porogram can provide services.

=>Examples: s.listen(2)

4) accept()

=> Syntax:- clientsocket, clientaddr=serversockobj.accept()

=>This Function is used accepting and connecting by server Side Program to the Client Side Program.

=>Example: - cs,ca=s.accept()

5) recv() with decode()

=>Syntax: strdata=clientsocketobj.recv(1024/2048/4096).decode()

=>The Function is used at Server Side for receiving Client Request and it can also be used at Client Side for receiving Server Response.

=>Examples: strdata=cs.recv(1024).decode() # Server Side and Client Side

6) send() with encode()

=>Syntax: clientsocketobj.send("strdata".encode())

=>The Function is used at Server Side for Sending Response to Client Side Program and used at Client Side for Sending Request to Server side program

=>Examples: strdata=cs.send("strdata".encode()) # # Server Side and Client Side

7) connect()

Syntax: clientsocketobj.connect(("DNS/IP Addr","Portno"))

=>This Function is used for obtaining connection Server Side Program by the client side Program

=>Examples: clientsocketobj.connect("localhost",8888)

(OR)

```
clientsocketobj.connect("127.0.0.1",8888)
```

#ClientSquare.py

```
import socket
```

```
s=socket.socket()
```

```
s.connect(("localhost",8888))
```

```
print("CSP Obtains Connection from Server Side Program")
```

```
n=input("Enter a value for getting square:")
```

```
s.send(n.encode())
```

```
res=s.recv(1024).decode()
```

```
print("Square({})={}".format(n,res))
```

#ServerSquare.py

ServerSquare.py

```

import socket
s=socket.socket()
s.bind(("localhost",8888))
s.listen(2)
print("SSP is Ready to accept any CSP request")
while(True):
    try:
        cs,ca=s.accept()
        strdata=cs.recv(1024).decode()
        n=int(strdata)
        print("Val of Client at Server Side:{} ".format(n))
        res=n*n
        cs.send(str(res).encode())
    except ValueError:
        cs.send("Don't enter alnums,strs and symbols".encode())


---


#This Program considered as Server Side Program, It receives the Messages from Client Side
Program and Gives Answer as Response to client side Program
#ChatServer.py-----Program-(A)
import socket
s=socket.socket()
s.bind( ("127.0.0.1",9999) )
s.listen(1)
print("SSP is ready to accept any CSP:")
print("-"*40)
while(True):
    cs,addr=s.accept()
    csdata=cs.recv(1024).decode()
    print("Student Msg-->{}".format(csdata))
    sdata=input("KVR-->")
    cs.send(sdata.encode())


---


#ServerSquare.py
import socket
s=socket.socket()
s.bind(("localhost",8888))
s.listen(2)
print("SSP is Ready to accept any CSP request")
while(True):
    try:
        cs,ca=s.accept()
        strdata=cs.recv(1024).decode()
        n=int(strdata)
        print("Val of Client at Server Side:{} ".format(n))
        res=n*n
        cs.send(str(res).encode())
    except ValueError:
        cs.send("Don't enter alnums,strs and symbols".encode())


---


#write a client side program which will accept employee no.from keyboard,send to the server
and get employee name,salary and designation from server side program.
#ClientEmpData.py
import socket

```

```

s=socket.socket()
s.connect(("127.0.0.1",3600))
print("CSP got Connection From SSP:")
empno=input("\nEnter Employee Number:")
s.send(empno.encode())
sdata=s.recv(1024).decode()
print("-----")
print("Result from Server about Employee:")
print("-----")
print(sdata)
print("-----")
# Write a server side program which will accept employee number from client, retrieve
empname, salary and designation from emp table.
#ServerEmpData.py-----Program-(A)
import socket
import cx_Oracle
s=socket.socket()
s.bind(("127.0.0.1",3600))
s.listen(2)
print("SSP is Ready to accept CSP request:")
while(True):
    try:
        cs,ca=s.accept()
        eno=int(cs.recv(1024).decode())
        #PDBC
        oracon=cx_Oracle.connect("scott/tiger@localhost/orcl")
        print("SSP connectd to Oracle DB")
        cur=oracon.cursor()
        cur.execute("select name,sal,cname from employee where eno=%d"
%eno)
        record=cur.fetchone()
        if(record==None):
            cs.send("Employee Record Does not Exist".encode())
        else:
            cs.send(str(record).encode())
    except ValueError:
        cs.send("Don't enter strs,Symbols and alph-numerics for empno".encode())
    )
    except cx_Oracle.DatabaseError as db:
        cs.send("Prob in DB"+str(db).encode())
    except :
        cs.send("OOOOPS Some went wrong".encode())

```

=====

os module

=====

=>In Python, "os" is one pre-defined module.

=>The purpose of os module is that "To perform some os related operations" much

- 1) Obtaining Current Working Folder(getcwd()).

- 2) Creating Folder / Directory. (mkdir())
 - 3) Creating Folders Hierarchy. (makedirs())
 - 4) Removing Folder / Directory. (rmdir())
 - 5) Removing Folders Hierarchy. (removedirs())
 - 6) Removing File Name from Folder(remove())
 - 7) Renaming a Folder/File Name. (rename())
 - 8) List the file names in folder (listdir())
-

1) Obtaining Current Working Folder.

=>For Obtaining Current Working Folder, we use a pre-defined function called getcwd()
present in os module,

=>Syntax: varname=os.getcwd()

```
#Program for Obtaining Current Working Folder  
# cwdex.py  
import os  
cwdname=os.getcwd()  
print("Current Working Folder=", cwdname)
```

2) Creating Folder / Directory

=>For Creating a Folder / Directory, we use mkdir().

=>Syntax: os.mkdir("Folder Name")

=>if the folder name already exist then we get FileNotFoundError

=>mkdir() can create only one folder at a time and if we try to create folder hierarchy then
we get FileNotFoundError.

=>in mkdir(), if we specify any folder name with escape sequence (\n \u \d\igits,\t..etc) then
we get OSError.

Examples:

```
#Program for Creating Folder / Directory  
# mkdirex.py  
import os  
try:  
    os.mkdir("D:\suraj\python\7am")  
    print("Folder Created Successfully-verify")  
except FileNotFoundError:  
    print("mkdir() can create only one folder at a time")  
except FileExistsError:  
    print("The specified folder already exist")  
except OSError:  
    print("Check ur path of folder names")
```

3) Creating Folders Hierarchy.

=>For Creating Folders Hierarchy, we use makedirs().

=>Syntax: os.makedirs("Folders Hierarchy")

=>Here Folders Hierarchy represent Root Folder\sub folder\sub-sub folder so on...

=>if the folder name already exist then we get FileExistsError
=> if we specify any folder name with escape sequence (\n \u \digits,\t..etc) then
we get OSError.

Examples:

```
#Program for Creating Folders Hierarchy
#makedirsex.py
import os
try:
    os.makedirs("D:\\India\\Hyd\\ampt\\python\\python")
    print("Folder Created Successfully-verify")
except FileExistsError:
    print("The specified folder already exist")
except OSError:
    print("Check ur path of folder names")
```

4) Removing Folder / Directory.

=>For Removing Folder / Directory, we use rmdir()
=>syntax: os.rmdir("folder name")
=>rmdir() can remove folder name provided folder name is empty.
=>if the folder name already exist then we get FileExistsError
=>if we specify any folder name with escape sequence (\n \u \digits,\t..etc) then
we get OSError.

```
#Program for Removing Folder / Directory
#rmdirsex.py
import os
try:
    os.rmdir("D:\\KVR")
    print("Folder removed Successfully-verify")
except FileNotFoundError:
    print("folder name does not exist")
except OSError:
    print("rmdir() can remove those foilder which are empty--check ur path")
```

5) Removing Folders Hierarchy. (removedirs())

=>For Removing Removing Folders Hierarchy, we use removedirs()
=>Syntax: os.removedirs("Folders Hierarchy")
=>Here Folders Hierarchy represent Root Folder\\sub folder\\sub-sub folder so on...
=>if the folder name not exist then we get FileNotFoundError
=> if we specify any folder name with escape sequence (\n \u \digits,\t..etc) then
we get OSError.

Examples

```
#Program for Removing Folders Hierarchy
#removedirsex.py
```

```
import os
try:
    os.removedirs("D:\\India\\Hyd\\ampt\\python\\python")
    print("Folders Hierarchy Removed Successfully-verify")
except FileNotFoundError:
    print("The specified folders hierarchy does exist")
except OSError:
    print("remove those folder which are empty-Check ur path of folder names")
```

6) Removing File Name from Folder.

=>To remove the file name from folder, we use remove()
=>Syntax: os.remove("Absolute Path of File Name")
=>If the file name does not exist then we get FileNotFoundError

Examples

```
#Program for removing the file name from folder
#RemoveFileEx.py
import os
try:
    os.remove("E:\\KVR-PYTHON-7AM\\MODULES\\SE3.py")
    print("File Name removed Sucessfully")
except FileNotFoundError:
    print("File does not exist")
```

7) Renaming a Folder/File Name.

=>To rename a folder, we rename()
=>Syntax: os.rename("Old Folder Name", "New Folde Name")
=>If the Old Folder Name does not exist then we get FileNotFoundError.

Examples

```
#Program for renaming a folder anme
#RenameFolderEx.py
import os
try:
    os.rename("D:\\KVR","D:\\PYTHON")
    print("Folder Name renamed")
except FileNotFoundError:
    print("File does not exist")
```

8) List the file names in folder.

=>To list the file names in folder, we use listdir()
=>Syntax: os.listdir("Absolute Path of Folder Name")
=>If the Old Folder Name does not exist then we get FileNotFoundError.

Examples:

```
#Program for Listing files ijn folder
#ListFileFolderEx.py
import os
try:
    FolderName=input("Enter Folder name to list files:")
    fileslist=os.listdir(FolderName)
    print("-"*50)
    print("List of Files:")
    print("-"*50)
    for filename in fileslist:
        print("\t{}".format(filename))
    print("-"*50)
except FileNotFoundError:
    print("Folder does not exist")
```

===== Iterators in Python =====

Why should WE use Iterators:

=>In modern days, we have a lot of data in our hands, and handling this huge amount of data creates problems for everyone who wants to do some sort of analysis with that data. So, If you've ever struggled with handling huge amounts of data, and your machine running out of memory, then WE use the concept of Iterators in Python.

=>Therefore, Rather than putting all the data in the memory in one step, it would be better if we could work with it in bits or some small chunks, dealing with only that data that is required at that moment. As a result, this would reduce the load on our computer memory tremendously. And this is what exactly the iterators do.

=>Therefore, you can use Iterators to save a ton of memory, as Iterators don't compute their items when they are generated, but only when they are called upon.

=>Iterator in python is an object that is used to iterate over iterable objects like lists, tuples, dicts, and sets.

=>The iterator object is initialized using the iter() method. It uses the next() method for iteration.

=>Here iter() is used for converting Iterable object into Iterator object.

=>next() is used for obtaining next element of iterator object and if no next element then we get an exception called StopIteration.

=>On the object of Iterator, we can't perform Indexing and Slicing Operations bcoz They supply the values on demand .

Examples:

```
s = 'Python'
itobj = iter(s)
while True:
```

```

try:
    item = next(s)      # Iterate by calling next
    print(item)
except StopIteration:    # exception will happen when iteration will over
    break
=====
mytuple = ("apple", "banana", "cherry")
myit = iter(mytuple)
print(next(myit))
print(next(myit))
print(next(myit))
print(next(myit))
=====
#IteratorEx1.py
lst=["Python","Java","C","C++","DSc"]
print(lst,type(lst))
print("-----")
lstitr=iter(lst)
print("Content of Iterators:")
print(next(lstitr))
print(next(lstitr))
print(next(lstitr))
print(next(lstitr))
print(next(lstitr))
print(next(lstitr))
print(next(lstitr))
=====
#IteratorEx2.py
tpl=("Python","Java","C","C++","DSc")
print(tpl,type(tpl))
print("-----")
tplitr=iter(tpl)
print("Content of Iterators:",type(tplitr))
while(True):
    try:
        print(next(tplitr))
    except StopIteration:
        break
=====
#IteratorEx3.py
tpl={"Python","Java","C","C++","DSc"}
print(tpl,type(tpl))
print("-----")
tplitr=iter(tpl)
print("Content of Iterators:",type(tplitr))
while(True):
    try:
        print(next(tplitr))
    except StopIteration:
        break
=====
```

JSON file

=>JSON (JavaScript Object Notation) is a popular data format used for representing structured data. It's common to transmit and receive data between a server and Client web application development in the form of JSON format.

=>In otherwords,JSON is a lightweight data format for data interchange which can be easily read and written by humans, easily parsed and generated by machines.

=>It is a complete language-independent text format. To work with JSON data, Python has a built-in module called json.

Parse JSON (Convert from JSON to Python)

=>`json.loads()` method can parse a json string and converted into Python dictionary.

Syntax:

```
dictobj=json.loads(json_string)
```

Examples:

```
# Python program to convert JSON to Python
```

```
import json
# JSON string
employee = ' {"id":"09", "name": "Rossum", "department":"IT"} '
# Convert JSON string to Python dict
employee_dict = json.loads(employee)
print(employee_dict)
```

Python--- read JSON file

=>`json.load()` method can read a file which contains a JSON object.

Consider a file named `employee.json` which contains a JSON object.

Syntax:

```
json.load(file_object)
```

Python--- write to JSON file

=>`json.dump()` method can write dict object data to a file.

Syntax:

```
json.dump(dict object, file_pointer)
```

```
# Python program to convert JSON to Python
```

```
#JSONLOADS.py
```

```
import json
```

```
# JSON string
```

```
employee = ' {"id":"09", "name": "Rossum", "department":"IT"} '
```

```
# Convert JSON string to Python dict
```

```
print("Type employee=",type(employee))
```

```
ed = json.loads(employee)
```

```
print(ed,type(ed))
for k,v in ed.items():
    print("{}-->{}".format(k,v))


---


# Python program to read json file
#JsonRead.py
import json
# Opening JSON file
try:
    with open("sample.json","r") as fp:
        # returns JSON object as a dictionary
        dictdata = json.load(fp)
        # Iterating through the json list
        for k,v in dictdata.items():
            print("{}--->{}".format(k,v))
except FileNotFoundError:
    print("Json File does not exist")


---


#JsontoDict.py
import json
# JSON string
employee = ' {"id":"09", "name": "Rossum", "department":"IT"} '
print(type(employee))
"""print("Json String data=",employee)
# Convert string to Python dict
employee_dict = json.loads(employee)
print("Dict Data=",employee_dict)
for k,v in employee_dict.items():
    print("{}-->{}".format(k,v))"""


---


# Python program to write JSON to a file
#JsonWrite.py
import json
# Data to be written
dictionary={"rollno" : 56,"name" : "Rossum", "cgpa" : 8.6}
with open("sample.json", "w") as fp:
    json.dump(dictionary, fp)
    print("Data written to file--verify")
```

