

PYTHON

Real Time Applications / Projects developed by using Python Language

==>By Using Python Programming, we can develop the following Applications.

1. Development of Web Applications (Web Sites)
- a) By using JAVA Program Lang---(Tech:Servlets,JSP,JDBC,Spring,...etc)
 - b) By Using C#.NET Program Lang--Tech ASP.NET
- c) By Using PYTHON Program Lang--Tech: Django,Flask, Pyramid, Bottle..
(Less Coding and More Meaning--Huge Modules / Libraries)
2. Gaming Applications.
3. Artificial Intelligence(ML, DL..etc)--Virtual Assitant--Alexa, Google , Siri..
4. Desktop GUI Applications.
5. Image Processing Applications
6. Text Processing Applications
7. Business Applications
8. Audio and Video Based Applications
9. Web Scraping / Web Harvesting Applications
10. Data Visualization
11. Scientific Applications--NASA
12. Software Development
13. Operating System Installers.
14. CAD and CAM based Applications
15. Embedded Applications
16. IOT Based Applications
17. Complex Math Operations
18. Language Development
19. Automation of Testing
20. Console Based Applications
21. Data Analysis and Data Analytics
22. Education Programming.....etc

Getting Started with Python

Index

- =>Python History
 - =>Versions of Python
 - =>DownLoading Process of Python
 - =>Installing Python Software
 - =>Features of python
-

Python History

=>Python Programming language Conceived in the year 1980.
 =>Python Programming language started its implementation (Bring into action) in the year 1989
 =>Python Programming language Officially released in the year 1991 Feb
 =>Python Programming language Developed by "GUIDO VAN ROSSUM".
 =>Python Programming language Developed at Centrum Wiskunde Informatica(CWI) in Nether Lands.
 =>Python Programming Language is one of Successor of ABC Language
 =>Python Programming Language Updation managed and maintained by a non-commercial Organisation called Python Software Foundation(PSF)
 =>The official website of PSF is www.python.org

Version of Python

=>Python programming Language contains 3 Types of Versions. They are

1. Python 1.x Where 1 is called Major Version and and x Represents 0 1 2 3 4 5 6 7 8 and Minor Version (Outdated)
 Python 2.x does not support backward Compatibility Python 1.x
2. Python 2.x Where 2 is called Major Version and and x Represents 0 1 2 3 4 5 6 7 and Minor Version(Outdated)
 Python 3.x does not support backward Compatibility with Python 2.x
3. Python 3.x Where 3 is called Major Version and x Represents 0 1 2 3 4 5 6 7 8 9 10 11 and Minor Version (Current Version)
 - Python 3.8 Python 3.9--Secured (MNC)
 - Python 3.10
 - Python 3.11 (Latest Version)
 - Python 3.12 (Future release)

Features of Python

=>Features of Language are nothing but Services / Facilities provided by language Developers and they are used by Language Programmers for developing Real Time Applications.
 =>Python Programming Provides 11 features. They are

1. Simple
2. FreeWare and Open Source
3. PlatForm Independent
4. Dynamically Typed
5. Interpreted
6. High Level
7. Robust (Strong)
8. Both Procedure Oriented and Object Oriented Programming lang.
9. Extensible
10. Embedded

11. Supports Third Party APIs such as NUMPY PANDAS MATPLOTLIB SCIPY
SCIKIT NLP SEABORN ...etc

1. simple

=>Python Programming Language is one of the SIMPLE Programming Language bcoz of the following Technical Factors.

Factor1:

=>Python Programming Provides "Rich Set of MODULES". So that Programmers can Re-Use the Pre-defined Code present in Pre-defined MODULES. So that writing python program is easy / simple.

=>Examples: qrcode,image,calendar,math,cmath,random...etc

Def. of MODULE:

=>A MODULE is a collection of Functions,Global Variables and Class Names

Factor2: (Memory Management)

=>Python Programming Provides In-Built Facility called "Garbage Collector". So that Garbage Collectors Collects Un-Used Memory Space and Improves the performance of Python Based Applications.

Def. of Garbage Collector:

A Garbage Collector is one of the background Python program which is running behind of regular python program and whose role is that to Collects Un-Used Memory Space and Improves the performance of Python Based Applications.

=>hence Garbage Collector takes care about automatic memory management.

Factor3:

=>Python Programming Provides User-Friendly Syntaxes. So the Python program Can develop Error-Free Programs within Limited span of time.

3. PlatForm Independent

=>A PlatForm is nothing but Type of OS being Used .

=>In this context, we are classifying the programming languages into Two Types. They are

1. Platform Dependent Languages:

=>Platform Dependent Languages are those Whose data Types takes Different Memory Spaces on Different Operating System.

2. PlatForm Independent Languages.

=>Platform InDependent Languages are those Whose data Types takes Same Memory Spaces on Different Operating System.

=>In the Case of Java , Pre-defined Data Types Takes Same Memory Space on Different Operating System and they allows us to store Single value.

=>In the Case of Java , Programmer-defined Data Types(classes) Takes their objects Multiple Values with Size Restrictions.

=>In The Case of Python language, All the Data Types are Classes and whose Objects in Python stores Un-Limited amount of Data and these object runs on all types of OSes.

=>JAVA, PYTHON

4. Dynamically Typed

=>In IT, We have two types of languages. They are

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

1. Static Typed Programming Languages

=>In Static Typed Programming Languages, It mandatory to use Data Types along with variable Names (Variable Declaration) Otherwise Compile Time Error.

Example:

```
int a,b,c ; // Variable Declaration--mandatory
a=10;
b=20
c=a+b
```

Examples: C,C++, Java, C#.net etc

2. Dynamically Typed Programming Languages

=>In Dynamically Typed Programming Languages, Programmers will not assign any data type. Internally Language Execution Environment will assign the data type automatically OR Implicitly based Type of Value Programmers uses data type will be assigned.

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

Examples: Python

=====

6. High Level Programming

=====

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

- 1. Low Level Programming Languages
 - 2. High Level Programming Languages
-

1. Low Level Programming Languages:

=>In Low Programming Languages, data is always stored in the form low level values such as Binary data, Octal Data and Hexa Decimal data. These Number System are not directly understandable end-users

Example : a=0b1010101010
 b=0xBEE
 c=0o23

2. High Level Programming Languages

=>In these languages, Internally, Even the programmer specifies the data in the form of Low Level Format such Binary data, Octal Data and Hexa Decimal data, automatically Python Programming Language Execution Environment Converts into High Level data, which is understandable by end-users . Hence Python is one of the High Level Programming Languages.

Examples:

```
>>> a=0b10101011110000
>>> b=0b10101111000
>>> print(a)-----22000
>>> print(b)-----1400
>>> a=0xBEE
>>> print(a)-----3054
>>> bin(22000)-----'0b10101011110000'
>>> hex(3054)-----'0xbEE'
```

=====

5. Interpreted Programming

=====

=>When we develop any python program, we must give some file name with an extension .py (File Name.py).

=>When we execute python program, two process taken place internally

- a) Compilation Process
- b) Execution Process.

=>In COMPILEMENTATION PROCESS, The python Source Code submitted to Python Compiler and It reads the source Code, Check for errors by verifying syntaxes and if no errors found then Python Compiler Converts into Intermediate Code called BYTE CODE with an extension .pyc (FileName.pyc). If errors found in source code then we get error displayed on the console.

=>In EXECUTION PROCESS, The PVM reads the Python Intermediate Code(Byte Code) and Line by Line and Converted into Machine Understable Code (Executable or binary Code) and It is read by OS and Processor and finally Gives Result.

=>Hence In Python Program execution, Compilation Process and Execution Process is taking place Line by Line conversion and It is one of the Interpretation Based Programming Language.

Definition of PVM (Python Virtual Machine)

=>PVM is one program in Python Software and whose role is to read LINE by LINE of Byte Code and Converted into Machine Understable Code (Executable or binary Code)

2. Freeware and Open Source

=>Freeware: Since Python Software can be downloaded Freely from www.python.org and hence Python is one of the Freeware.

=>Open Source:

=>The standard name of Python Lang is "Cpython"

=>Some Company Vendors came forward and customized "Cpython" for their In-house tools and such type 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) Iron Python or Ipython--->Used for Running C#.net Applications
 - 3) Ruby Python----->Used for Running Ruby Based Applications
 - 4) Anaconda Python--->Dealing with Hadoop / Big Data Applications
 - 5) Micro Python----->Used for developing Micro Controller Applications.
 - 6) Stackless Python--->Concurrency Based Applications.
-etc

7. Robust (Strong)

=>Python is one of the Robust (Strong) programming Lang bcoz of "Exception Handling".

=>Def. of Exception:

=>Every Runtime Error is called Exception(Invalid Invalid-->Runtime Error-->Exception)
=>All types of Runtime Error / Exceptions generates by default Technical Error Messages, which are understandable by programmers but not Application OR End Users. This is not a recommended Process.
=>Industry is Highly recommended to generate User Friendly Error Messages by using Exception Handling.

=>Def of Exception Handling

=>The Process of Converting Technical Error Messages into User Friendly Error Messages is called Exception Handling.

=====X=====

9 Extensible and 10 Embedded

Extensible:

=>A programming language is said to be "Extensible" iff It provides Programming Facilities to other languages
=>Example: Python Programming Provides its Programming Facilities to C,C+,Java..etc and hence Python is one of the Extensible Programming Lang.

Embedded:

=>A programming Lang is said to be Embedded iff It takes the Programming Services of Other languages.
=>For Example: Python Programming Embeds the Programming Services of C bcoz C is the Fastest Programming Lang.

=====X=====

Supports Third Party APIs Such as numpy, pandas, scipy, scikit, keras, matplotlib, nlp..etc

=>Most of the Supports Third Party APIs Such as numpy, pandas, scipy, scikit, keras, matplotlib, nlp..etc are providing Easiness to Python programmer in the case Complex Maths Calculations(Numpy), Business Analysis and Analytics (Pandas)..etc

Data Representation in Python

Index:

=>Importance of data/ values / Literals

=>Types of Literals

=>Importance of Identifiers / Variables

=>Rules for Using variables / Identifiers in Python

=>In Real Time , we need the DATA(Literals) taking Effective Decisions.

=>To take Effective Decisions on DATA, we must STORE DATA in MAIN MEMORY (RAM)

=>Primarly, The DATA Stored in main memory classified into 5 types. They are

- 1) Integer Literals (Ex sno,eno,htno,adcno,acno)
 - 2) floating Point Literals (Percent, totalbill..etc)
 - 3) Boolean Literals (True False)
 - 4) String Literals (Name, Place Names, product names..etc)
 - 5) date literals (DOB, DOJ,DOD,DOA..etc)
-

Rules for Using variables OR Identifiers inPython

=>To use variables / identifiers in our Python program we must follow the following rules.

Rule1: A variable name is a combination of Alphabets or Digits and a special symbol Under Score (_).

Rule2: First letter of the Variable name must starts either with an alphabet or a special symbol Under ----- Score (_).

Examples:

```

123=67----Invalid
-sal=56--invalid
$sal=67---invalid
sal=45----valid
#sal=67--invalid
2sal=56---invalid
_sal=56---valid
sal2=67---valid
__sal__ = 56--valid
__ = 56-----valid

```

Rule-3: No special symbol are allowed Within the variable name except Under Score (_)

Examples:

emp sal=56----Invalid

emp\$sal=56---invalid
 sal,sal=67---invalid
 emp_sal=56--valid
 emp-sal=78--invalid

Rule-4: No Keywords to be used as Variable names (bcoz keywords are the Reserved words and we can't change the meaning of key words.)

Examples:

if=45----Inavlid
 else=56--invalid
 while=56--incalid
 _if=45--valid
 if1=56--valid
 _while_1=56--valid

Rule-5: All the variables in Python are CASE SENSITIVE

Examples:

age=99--valid
 AGE=98---valid
 Age=97--valid
 aGe=98---valid

All the above variables are different.

Rule-6: It is recommended to takes Variables as User-Friendly Name
 (Not recommended to take big names)

===== Data Types in Python =====

=>The Purpose of data type in Python is that " To store the inputs / data / Literals in main memory of the

computer by allocating memory space".

=>Python Programming Contains 14 Data Types , which are classified into 6 categories. They are

I. Fundamental Category Data Types

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

II. Sequence Category Data Types

1. str
2. bytes
3. bytearray

- 4. range
- III. List Category Data Types (Collection Data Types OR Data Structures)
- 1. list
- 2. tuple
- IV. Set Category Data Types(Collection Data Types OR Data Structures)
- 1. set
- 2. frozenset
- V. Dict Category Data Type (Collection Data Types OR Data Structures)
- 1. dict
- VI. NoneType Category Data Type
- a. NoneType

I. Fundamental Category Data Types

=>The purpose of Fundamental Category Data Type 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 name and treated as Fundamental data type.
=>The purpose of 'int' data type is that " To Store Whole Numbers OR Integer Data or Integral Values (Numbers without Decimal values)". Ex: stno,acno,empno..etc

Examples:

Python Instructions:

```
>>> a=10
>>> print(a)-----10
>>> type(a)-----<class 'int'>
>>> id(a)-----1829952094736
```

OR

```
>>> print(a,type(a),id(a))-----10 <class 'int'> 1829952094736
```

Output:

```
>>> acno=1234
>>> print(acno,type(acno),id(acno))-----1234 <class 'int'> 1829953134480
```

```
>>> a=10
>>> b=20
>>> c=a+b
>>> print(a,type(a),id(a))-----10 <class 'int'> 1829952094736
>>> print(b,type(b),id(b))-----20 <class 'int'> 1829952095056
>>> print(c,type(c),id(c))-----30 <class 'int'> 1829952095376
>>> print(a,b,c)-----10 20 30
```

=>With int data type, we can store Different types of Number Systems Data.
=>In any Programming Language, we have 4 types of Number Systems. They are

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

1. Decimal Number System

=>Decimal Number System data understandable all Human Beings and it is one of the default number

system

=>This Number System contains

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

=>Base 10 Literals are called Decimal Number System values:

2. Binary Number System

=>Binary Number System data understandable by Operating Systems and Processors.

=>This Number System contains

Digits: 0 1 ---- Total number of digits=2
Base : 2

=>Base 2 literals are called Binary Number System values.

=>In Python Programming, To Store Binary Number System Data, Binary data must be preceded with a letter either 0b or 0B

Syntax: varname=0b Binary data

(OR)

varname=0B Binary data

=>Even though we store the data in the form of binary , internally It is converted into decimal number system

Examples:

```
>>> a=0b1110
```

```
>>> print(a,type(a))-----14 <class 'int'>
>>> a=0b11001
>>> print(a,type(a))-----25 <class 'int'>
>>> a=0b1111
>>> print(a,type(a))-----15 <class 'int'>
>>> a=0B11
>>> print(a,type(a))-----3 <class 'int'>
```

NOTE: To Convert Decimal Number System data into Binary Data, we use a pre-defined Function called

bin()

Syntax: var name=bin(decimal data)

Examples:

```
>>> x=bin(3)
>>> print(x)-----0b11
>>> x=bin(15)
>>> print(x)-----0b1111
>>> x=bin(25)
>>> print(x)-----0b11001
>>> x=bin(14)
>>> print(x)-----0b1110
```

3. Octal Number System

=>Octal Number System data understandable by Micro processor Kits.

=>This Number System contains

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

=>Base 8 literals are called Octal Number System values.

=>In Python Programming, To Store Octal Number System Data, Octal data must be preceded with a letter either 0o or 0O

Syntax: varname=0o Octal data

(OR)

varname=0O Octal data

=>Even we though we store the data in the form of Octal , internally It is converted into decimal number System

Examples:

```
>>> a=0o27
>>> print(a,type(a))-----23 <class 'int'>
>>> a=0O123
>>> print(a,type(a))-----83 <class 'int'>
>>> a=0O37
>>> print(a,type(a))-----31 <class 'int'>
```

NOTE: To Convert Decimal Number System data into Octal Data, we use a pre-defined Function called

```

oct()
Syntax: var name=oct(decimal data)
>>> x=oct(31)
>>> print(x)-----0o37
>>> x=oct(83)
>>> print(x)-----0o123
>>> x=oct(23)
>>> print(x)-----0o27
>>> x=oct(0b1010)
>>> print(x)-----0o12
>>> x=oct(0b1111)
>>> print(x)-----0o17

```

4. Hexa Decimal Number System

=> Hexa Decimal System Used in development of Operating 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

number of digits=16

Base :16

=>Base 16 literals are called Hexa Decimal Number System values.

=>In Python Programming, To Store Hexa Decimal Number System Data, Hexa Decimal data must be preceded with a letter either 0x or 0X

Syntax: varname=0x Hexa Decimal data
(OR)

varname=0X Hexa Decimal data

=>Even we though we store the data in the form of Hexa Decimal , internally It is converted into decimal number system

Examples:

```

>>> a=0XAC
>>> print(a,type(a))-----172 <class 'int'>
>>> a=0xBEE
>>> print(a,type(a))-----3054 <class 'int'>
>>> a=0xfAce
>>> print(a,type(a))-----64206 <class 'int'>

```

NOTE: To Convert Decimal Number System data into Hexa Decimal Data, we use a pre-defined Function called hex()

Syntax: var name=hex(decimal data)

Examples:

```

>>> x=hex(64206)
>>> print(x)-----0xface
>>> x=hex(3054)
>>> print(x)-----0xbbee

```

```
>>> x=hex(172)
>>> print(x)-----0xac
>>> x=hex(15)
>>> print(x)-----0xf
>>> x=hex(10)
>>> print(x)-----0xa
>>> x=hex(0b1110)
>>> print(x)-----0xe
>>> x=hex(60)
>>> print(x)-----0x3c
>>> x=hex(0o27)
>>> print(x)-----0x17
```

NOTE: bin(), oct() and hex() are called Base Conversion Functions.

Most Imp:

```
>>> x=0b1111
>>> print(x)-----15
>>> x=0b1112-----SyntaxError: invalid digit '2' in binary literal
```

```
>>> a=0o123
>>> print(a)-----83
>>> a=0o1238-----SyntaxError: invalid digit '8' in octal literal
```

```
>>> a=0xBEE
>>> print(a)-----3054
>>> a=0xFACER-----SyntaxError: invalid hexadecimal literal
```

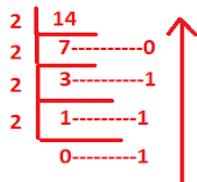
```
>>> a=0345---SyntaxError: leading zeros in decimal integer literals are not permitted;
use an 0o prefix for octal integers
```

```
>>> a=345 # Represents Decimal Number System Data.
>>> print(a)---345
=====X=====
```

Conversion from Decimal to Binary

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

Sol:



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

Conversion from Binary to Decimal

Q) $(1110)_2 \rightarrow (x)_{10}$

Sol:

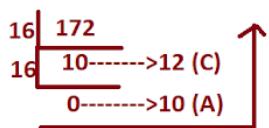
$$\begin{array}{r}
 1 & 1 & 1 & 0 \\
 3 & 2 & 1 & 0 \\
 \Rightarrow 2 & 2 & 2 & 2 \\
 \Rightarrow 1 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 \\
 \Rightarrow 1 \times 8 + 1 \times 4 + 1 \times 2 + 0 \times 1 \\
 \Rightarrow 8 + 4 + 2 + 0 \\
 \Rightarrow 14
 \end{array}$$

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

Conversion from Decimal to Hexa Decimal

Q) $(172)_{10} \rightarrow (x)_{16}$ find $x = AC$ OR ac

Sol:



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

Conversion from Hexa Decimal to decimal

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

Sol:

$$\begin{array}{r}
 A & C & 0 \\
 1 & 16 & 16 \\
 \Rightarrow A \times 16^1 + C \times 16^0 \\
 \Rightarrow 10 \times 16 + 12 \times 1 \\
 \Rightarrow 160 + 12 \\
 \Rightarrow 172
 \end{array}$$

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

| Conversion from Decimal to Octal | Conversion from Octal to Decimal |
|--|---|
| <p>Q) $(23)_{10} \rightarrow (x)_8$ find $x=27$</p> <p>Sol:</p>  <p>Hence $(23)_{10} \rightarrow (27)_8$</p> | <p>Q) $(27)_8 \rightarrow (x)_{10}$ find x</p> <p>Sol:</p> $\begin{array}{r} 2 & 7 \\ 1 & 0 \\ \Rightarrow & 8 & 8 \\ \Rightarrow & 2 \times 8^1 + 7 \times 8^0 \\ \Rightarrow & 16 + 7 \\ \Rightarrow & 23 \end{array}$ <p>hence $(27)_8 \rightarrow (23)_{10}$</p> |

2. float

Properties

- =>'float' is one of the pre-defined class name 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 points)".
- =>the float data type can also store floating point values in scientific Notation.
- =>The General format of scientific Notation is "Mantissa e Exponent"
- =>The advantage of scientific Notation is that To save memory space for storing floating point values.
- =>float data type does not support Binary , octal and Hexa decimal number system values. It can support only Decimal number system data in the form of Integer Part and Decimal part

Examples:

```
>>> a=0b1010.0b1111----SyntaxError: invalid decimal literal
>>> a=0xBEE.0xACC----SyntaxError: invalid decimal literal
>>> a=0o23.0b10101----SyntaxError: invalid decimal literal
```

3. bool

Properties

=>'bool' is one of pre-defined class name and treated as Fundamental data type.
 =>The purpose bool data type is that " To store True and False Values".
 =>True and False are the Keywords and Treated as values of bool data type.
 =>Internally True represented as 1 and False represented as 0

Examples:

```
>>> a=True
>>> b=False
>>> print(a,type(a))-----True <class 'bool'>
>>> print(b,type(b))-----False <class 'bool'>
>>> a=true----NameError: name 'true' is not defined (bcoz true is var and it is not
found).
```

```
>>> a=True
>>> b=False
>>> print(a+b)-----1
>>> print(True+True)-----0
>>> print(True*True)-----1
>>> print(True*False+True)-----1
>>> print(True+2.3)-----3.3
>>> print(2*True+3)-----5
>>> print(0b1111+True)----16
>>> print(0XA+True)-----9
>>> print(0XA+True-1)---8
>>> print(0XA+True*2)----8
>>> print(1.2+0b1010+False)---11.2
```

4. complex

=>'complex' is one of pre-defined class name and treated as Fundamental data type.
 =>The purpose of complex data type is that " To Store Complex Data".
 =>The general notation of complec data is show bellow

$a+bj$ or $a-bj$

Here 'a' is called Real Part

Here 'b' is called Imaginary part and 'j' represents $\sqrt{-1}$

=>Internally, Real and Imaginary Parts of Complex object are treated as float type.
 =>To extarct real and imaginary parts of Complex object, we have Two Pre-defined
 atrributes in complex object. They are

- 1) real
- 2) imag

=>Syntax: complexobj.real====>Will give Real part of Complex object
 complexobj.imag==>Will give Imaginary Part Complex Object

=>On the complex data, we can perform all types of arithmetic operations such as Add, sub , Mul etc.

Examples:

```
>>> a=2+3j
>>> print(a,type(a))-----(2+3j) <class 'complex'>
>>> b=2-3j
>>> print(b,type(b))-----(2-3j) <class 'complex'>
>>> c=2.5+5.6j
>>> print(c,type(c))-----(2.5+5.6j) <class 'complex'>
>>> d=3.4-5.7j
>>> print(d,type(d))-----(3.4-5.7j) <class 'complex'>
>>> e=3+4.5j
>>> print(e,type(e))-----(3+4.5j) <class 'complex'>
>>> f=3j
>>> print(f,type(f))-----3j <class 'complex'>
>>> g=-4j
>>> print(g,type(g))----- (-0-4j) <class 'complex'>
>>> h=-4.5j
>>> print(h,type(h))----- (-0-4.5j) <class 'complex'>

>>> a=2+3.5j
>>> print(a,type(a))-----(2+3.5j) <class 'complex'>
>>> print(a.real, type(a.real))---2.0 <class 'float'>
>>> print(a.imag, type(a.imag))----3.5 <class 'float'>
>>> a=10+20j
>>> print(a,type(a))-----(10+20j) <class 'complex'>
>>> print(a.real)-----10.0
>>> print(a.imag, type(a.imag))---20.0 <class 'float'>
>>> a=3j
>>> print(a,type(a))-----3j <class 'complex'>
>>> print(a.real)-----0.0
>>> print(a.imag, type(a.imag))----3.0 <class 'float'>
>>> a=-4j
>>> print(a,type(a))-----(-0-4j) <class 'complex'>
>>> print(a.real)----- -0.0
>>> print(a.imag, type(a.imag))--- -4.0 <class 'float'>

>>> a=2+3j
>>> b=3+4j
>>> c=a+b
>>> print(c,type(c))-----(5+7j) <class 'complex'>
>>> d=a-b
>>> print(d,type(d))-----(-1-1j) <class 'complex'>
>>> e=a*b
```

```
>>> print(e,type(e))-----(-6+17j) <class 'complex'>
-----
>>> print(True+3+4j)---(4+4j)
>>> print(True+3+4j)---(4+4j)
>>> print(2+3+4j)-----(5+4j)
>>> print(2*3+4j)-----(6+4j)
>>> print(2*3+4j+4)-----(10+4j)
>>> print(2*3+4j+4+2j)-----(10+6j)
```

II. Sequence Category Data Types

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

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

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

1. str

Index

=>Purpose of str

=>Types of strs

=>Notations for storing str data

=>Str Memory Management

=>Operations on str data

- a) Indexing
- b) Slicing

=>Programming Examples

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

=>The purpose of str data type is that "To store String data or text data or Alphanumeric data or numeric data or special symbols within double Quotes or single quotes or triple double quotes and triple single quotes. "

=>Def. of str:

=>str is a collection of Characters or Alphanumeric data or numeric data or any type data enclosed within double Quotes or single quotes or triple double quotes and triple single quotes. "

Types of Str data

=>In Python Programming, we have two types of Str Data. They are

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

1. Single Line String Data:

=>Syntax1:- varname=" Single Line String Data "
(OR)

=>Syntax2:- varname=' Single Line String Data '

=>With the help double Quotes (" ") and single Quotes (' ') we can store single line str data only but not possible to store multi line string data.

2. Multi Line String Data:

```
=>Syntax2:-    varname=' ' ' String Data1  
                           String Data2  
-----  
                           String data-n ' '
```

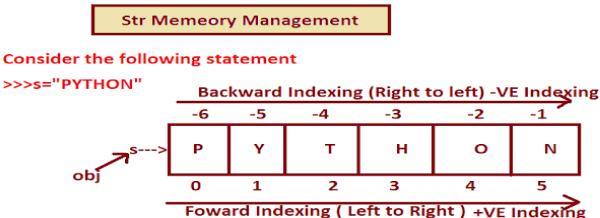
=>With the help triple double Quotes (" " " " ") and Triple single Quotes (' '' '' ') we can store single line str data and multi line string data.

```
>>> s1="Python Programming"
>>> print(s1,type(s1))-----Python Programming <class 'str'>
>>> s2='Java Programming'
>>> print(s2,type(s2))-----Java 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-56 " "
>>> print(addr1,type(addr1))
                Guido Van Rossum
                FNO:3-4, Red Sea Side
                Python Software Foundation
                Nether Lands-56 <class 'str'>
>>> addr2=' ' 'James Gosling
... HNO:13-45, Hill Side
... Sun Mocro System INC
... USA-567 ''
>>> print(addr2,type(addr2))
```

James Gosling
 HNO:13-45, Hill Side
 Sun Micro System INC
 USA-567 <class 'str'>

```
>>> s3="""Python Programming"""
>>> print(s3,type(s3))-----Python Programming <class 'str'>
>>> s4= '''Data Science'''
>>> print(s4,type(s4))-----Data Science <class 'str'>

>>> c1="A"
>>> print(c1,type(c1))-----A <class 'str'>
>>> c2='A'
>>> print(c2,type(c2))-----A <class 'str'>
>>> c3="""A"""
>>> print(c3,type(c3))-----A <class 'str'>
>>> c4="A"
>>> print(c4,type(c4))-----A <class 'str'>
>>> s5="Python3.10.5"
>>> print(s5,type(s5))-----Python3.10.5 <class 'str'>
>>> s6="ABCDabcd45678#$%^&*_kvr"
>>> print(s6,type(s6))-----ABCDabcd45678#$%^&*_kvr <class 'str'>
```



===== Operations on str data (part-1) =====

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

1. indexing
 2. slicing
-

1) indexing

=>The process of Obtaining single Value from given str obj by passing Valid Index Value is called

Indexing.

=>Syntax: strobj [Index]
=>here strobj is an object <class,'str'>
=>here [] represents Square Barckets always used Indexing Operations
=>here 'Index' represents either +Ve index OR -Ve Index
=>If we enter invalid index then we get IndexError as an error message

Examples

```
>>> s="PYTHON"
>>> print(s)-----PYTHON
>>> print(s[0]) ----- P
>>> print(s[-6])----- P
>>> print(s[-1])----- N
>>> print(s[5])----- N
>>> s[0]----- 'P'
>>> s[-1]----- 'N'
>>> s[-5]----- 'Y'
>>> s[-2]----- 'O'
>>> s[-3]----- 'H'
>>> s[12]-----IndexError: string index out of range
>>> s[-23]-----IndexError: string index out of range

>>> s="123_456"
>>> print(s,type(s))-----123_456 <class 'str'>
>>> s[2]----- '3'
>>> s[-5]----- '3'
>>> print(s[3])----- _
>>> print(s[-4])----- _
```

2) slicing:

=> The process of obtaining Range of Chars OR Sub String from given strobj is called Slicing Operation.
=>In Python programming, to perform Slicing Operations, we have 5 Syntaxes. . They are

Syntax-1: strobj[BEGIN : END]

=>This Syntax gives Range of Chars / Sub String from BEGIN Index to END-1 Index provided BEGIN<END Othwerwise we never get any result (we get ' ' as a result)

=>Examples:

```
>>> s="PYTHON"
>>> print(s,type(s))-----PYTHON <class 'str'>
>>> s[0:3]-----'PYT'
>>> print(s[0:3])-----PYT
>>> s[2:5]-----'THO'
>>> s[1:5]-----'YTHO'
```

```

>>> s[1:6]-----'YTHON'
>>> s[0:6]-----'PYTHON'
>>> s[4:6]-----'ON'
>>> s[6:4]-----''
>>> print(s[6:4])-----No Result
>>> print(s[14:6])-----No result
>>> s[14:6]-----''

-----
>>> s[-6:-2]-----'PYTH'
>>> print(s[-6:-2])-----PYTH
>>> s[-4:-1]-----'THO'
>>> s[-5:-3]-----'YT'
>>> s[-3:-5]-----''
>>> print(s[-3:-5])-----No Result

```

Special Example: strobj[Pos BEG Index: Neg END Index]

This Syntax gives range of Chars from BEG Index to Neg END-1 Index provided Pos BEG > Neg END Otherwise we never get any result.

Examples

```

>>> s="PYTHON"
>>> print(s,type(s))-----PYTHON <class 'str'>
>>> s[2:-2]-----'TH'
>>> s[1:-1]-----'YTHO'
>>> s[-1:1]-----''
>>> s[3:-1]-----'HO'
>>> s[1:-1]-----'YTHO'
>>> s[0:-1]-----'PYTHO'
>>> s[-1:0]-----''

```

Syntax2: strobj[BEGIN :]

=>In this Syntax, we specified BEGIN Index and we didn't specify END Index

=>If we don't specify END Index then PVM takes upto Last Character
(OR)

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

Examples:

```

>>> s="PYTHON"
>>> print(s)-----PYTHON
>>> s[0:]-----'PYTHON'
>>> s[2:]-----'THON'
>>> s[3:]-----'HON'
>>> s[4:]-----'ON'
>>> s[1:]-----'YTHON'
>>> s[5:]-----'N'
>>> s="PYTHON"

```

```
>>> print(s)-----PYTHON
>>> s[-3:]-----'HON'
>>> s[-6:]-----'THON'
>>> s[-4:]-----'THON'
>>> s[-2:]-----'ON'
>>> s[-1:]-----'N'
>>> s[-5:]-----'THON'
```

Syntax3: strobj[: END]

=>In this Syntax, we specified END Index and we didn't specify BEGIN Index
=>If we don't specify BEGIN Index then PVM takes from First Character
(OR)
=>If we don't specify BEGIN Index then PVM takes 0 Index as BEGIN Index.

Examples

```
>>> s="PYTHON"
>>> print(s)-----PYTHON
>>> s[:4]-----'PYTH'
>>> s[:2]-----'PY'
>>> s[:6]-----'THON'
>>> s[:3]-----'PYT'
>>> s[:5]-----'THON'
>>> s[:1]-----'P'
```

```
>>> s="THON"
>>> print(s)-----THON
>>> s[:-3]-----'THON'
>>> s[:-5]-----'THON'
>>> s[:-2]-----'THON'
>>> s[:-1]-----'THON'
>>> s[:-4]-----'THON'
```

Syntax4: strobj[:]

=>In this Syntax, we didn't specify BEGIN and END Indices.
=>=>If we don't specify BEGIN and END Indices then PVM takes from First Character to last Character.
(OR)
=>If we don't specify BEGIN and END Indices then PVM takes 0 Index as BEGIN Index and len(strobj)-1 as END Index.
=>In Simple Words, This always gives complete strob data.

Examples:

```
>>> s="THON"
>>> print(s)-----THON
>>> s[:]-----'THON'
```

```

>>> s[0:-----'PYTHON'
>>> s[:6]-----'PYTHON'
>>> s[-6:]-----'PYTHON'
-----
>>> s="Java Programming"
>>> print(s)-----Java Programming
>>> s[0:-----'Java Programming'
>>> s[-17:]-----'Java Programming'
>>> s[:len(s)]-----'Java Programming'
>>> s[:]-----'Java Programming'

```

Syntax5: strobj[BEGIN : END :STEP]

Rule-1:

Here BEGIN, END and STEP are either +ve or -ve

Rule-2:

If value of STEP is +VE then PVM gives range of Chars / Sub String from BEGIN to END-1 in FORWARD Direction by maintaining STEP (Equal Interval Value) prived BEGIN>END otherwise we never get any result (' ' is the result)

Rule-3:

If value of STEP is -VE then PVM gives range of Chars / Sub String from BEGIN to END+1 in BACKWARD Direction (i.e END+1 to BEGIN) by maintaining STEP (Equal Interval Value) provided BEGIN>END

Rule-4:

If we extract the data in FORWARD Direction and if we specify END Index as 0 then we never get any result

Rule-5:

If we extract the data in BACKWARD Direction and if we specify END Index as -1 then we never get any result

Examples:

```

>> s="PYTHON"
>>> print(s)-----PYTHON
>>> s[0:5:2]-----'PTO'
>>> s[0:5]-----'PYTHO'
>>> s[0:5:1]-----'PYTHO'
>>> s[0:5:3]-----'PH'
>>> s[0:6:4]-----'PO'
>>> s[0:6:5]-----'PN'
>>> s[0:6:1]-----'PYTHON'
>>> s[0:6:6]-----'P'
>>> s[0:6:600]-----'P'
>>> s[2:6:]-----'THON'

```

```

>>> s="PYTHON"
>>> print(s)-----PYTHON
>>> s[0:6:-1]-----"
>>> s[5:1:-1]-----'NOHT'

```

```

>>> s[::]-----'PYTHON'
>>> s[::-1]-----'NOHTYP'
>>> s[5:1:-2]-----'NH'
>>> s[5::-3]-----'NT'
>>> s[5::3]-----'N'
>>> s[::-2]-----'NHY'
>>> s[-1:-6:-1]-----'NOHTY'
>>> s[-2::-2]-----'OTP'
>>> s[-2::2]-----'O'

-----
>>> s="PYTHON"
>>> s[1:6:1]-----'YTHON'
>>> s[1:0:2]-----' ' (Rule-4)
>>> s[4:-1:-1]-----' ' (Rule-5)
>>> s[2:-1:-1]-----' ' (Rule-5)

```

Type Casting Techniques

=>The Process of Converting One Type of Possible Value into another type of Value is Called Type Casting.

=>In Programming Programming, we have 5 Types of Fundamental Type Casting Techniques. They are

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

1) int()

=>int() is used for converting Possible type of value into int type value.

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

Example-1: float type to int type---->Possible

```

>>> a=12.34
>>> print(a,type(a))-----12.34 <class 'float'>
>>> b=int(a)
>>> print(b,type(b))-----12 <class 'int'>
>>> a=0.999
>>> print(a,type(a))-----0.999 <class 'float'>
>>> b=int(a)
>>> print(b,type(b))-----0 <class 'int'>

```

Example-2: bool type to int type---->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'>
```

Example-3: complex type to int type---->Not Possible

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

Str Type to int type

Case-1: str int type to int type----> Possible

```
>>> a="10" # Str int
>>> print(a,type(a))-----10 <class 'str'>
>>> b=int(a)
>>> print(b,type(b))-----10 <class 'int'>
```

Case-2: str float type to int type---->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 type to int type---->Not Possible

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

Case-4: str complex type to int type---->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-4: Pure str type to int type---->Not Possible

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

2) float()

=>float() is used for converting Possible type of value into float type value.

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

Example-1: int type to float type---->Possible

```
>>> a=10
>>> print(a,type(a))-----10 <class 'int'>
>>> b=float(a)
>>> print(b,type(b))-----10.0 <class 'float'>
>>> a=0
>>> print(a,type(a))-----0 <class 'int'>
>>> b=float(a)
>>> print(b,type(b))-----0.0 <class 'float'>
```

Example-2: bool type to float type---->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-3: complex type to float type---->Not Possible

```
>>> a=2.5+2.3j
>>> print(a,type(a))-----(2.5+2.3j) <class 'complex'>
>>> b=float(a)---TypeError
```

Str Type to float type

Case-1: str int type to float type----> 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 type to float type--->Possible

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

```
>>> a="12.34.56"
>>> print(a,type(a))-----12.34.56 <class 'str'>
>>> b=float(a)-----ValueError: could not convert string to float: '12.34.56'
```

Case-3: str bool type to float type--->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 type to float type--->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-4: Pure str type to float type--->Not Possible

```
>>> a="Python3.11"
>>> print(a,type(a))-----Python3.11 <class 'str'>
>>> b=float(a)-----ValueError: could not convert string to float:
'Python3.11'
```

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

=>bool() is used for converting Possible type of value into bool type value.

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

=>ALL TYPES OF NON-ZERO VALUES ARE TREATED AS TRUE

=>ALL TYPES OF ZERO VALUES ARE TREATED AS FALSE

Example-1: int type to bool type--->Possible

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

```
>>> print(a,type(a))-----0 <class 'int'>
>>> b=bool(a)
>>> print(b,type(b))-----False <class 'bool'>
>>> a=-1233
>>> print(a,type(a))----- -1233 <class 'int'>
>>> b=bool(a)
>>> print(b,type(b))-----True <class 'bool'>
```

Example-2: float type to bool type----->Possible

Example-3: complex type to bool type----->Possible

```
>>> a=2+3.5j
>>> print(a,type(a))----- (2+3.5j) <class 'complex'>
>>> b=bool(a)
>>> print(b,type(b))----- True <class 'bool'>
>>> b=0+0.0j
>>> print(b,type(b))----- 0j <class 'complex'>
>>> a=bool(b)
>>> print(a,type(a))----- False <class 'bool'>
```

Str Type to bool type

Case-1: str int type to bool type----> Possible

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

```
>>> len(a)-----1
>>> a="000000"
>>> print(a,type(a))-----000000 <class 'str'>
>>> b=bool(a)
>>> print(b,type(b))-----True <class 'bool'>
>>> len(a)-----6
```

Case-2: str float type to bool type--->Possible

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

Case-3: str bool type to bool type--->Possible

```
>>> a="True"
>>> print(a,type(a))-----True <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'>
>>> len(a)-----5
```

Case-4: str complex type to bool type--->Possible

```
>>> a="2+3j"
>>> print(a,type(a))-----2+3j <class 'str'>
>>> b=bool(a)
>>> print(b,type(b))-----True <class 'bool'>
>>> len(a)-----4
>>> a="0+0.0j"
>>> print(a,type(a))-----0+0.0j <class 'str'>
>>> b=bool(a)
>>> print(b,type(b))-----True <class 'bool'>
```

Case-5: Pure str type to bool type--->Possible

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

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

4) complex()

=>complex() is used for converting Possible type of value into complex type value.
=>Syntax: varname=complex(int / float / bool / str)

Example-1: int type to complex type---->Possible

```
>>> a=10
>>> print(a,type(a))-----10 <class 'int'>
>>> b=complex(a)
>>> print(b,type(b))-----(10+0j) <class 'complex'>
```

Example-2: float type to complex type---->Possible

```
>>> a=-12.34
>>> print(a,type(a))------12.34 <class 'float'>
>>> b=complex(a)
>>> print(b,type(b))-----(-12.34+0j) <class 'complex'>
```

Example-3: bool type to complex type----> Possible

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

Str Type to complex type

Case-1: str int type to complex type---> Possible

```
>>> a="34"
>>> print(a,type(a))-----34 <class 'str'>
>>> b=complex(a)
>>> print(b,type(b))-----(34+0j) <class 'complex'>
```

Case-2: str float type to complex type--->Possible

```
>>> a="12.34"
```

```
>>> print(a,type(a))-----12.34 <class 'str'>
>>> b=complex(a)
>>> print(b,type(b))----(12.34+0j) <class 'complex'>
```

Case-3: str bool type to complex type--->Not Possible

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

Case-4: str complex type to complex type--->Possible

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

Case-4: Pure str type to complex type--->Not Possible

```
>>> a="Python+KVRj"
>>> print(a,type(a))-----Python+KVRj <class 'str'>
>>> b=complex(a)-----ValueError: complex() arg is a malformed string
```

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

=>str() can convert all types of values into str type values.
=>Syntax: varname=str(int/float/bol/complex)

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

```
>>> a=123.456
>>> print(a,type(a))-----123.456 <class 'float'>
>>> b=str(a)
>>> print(b,type(b))-----123.456 <class 'str'>
>>> b-----'123.456'
```

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

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

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

2. bytes

Properties:

=>'bytes' is one of the pre-defined class name and treated as sequence data type.
=>The internal Implementation bytes data type is that " To Implement End-to-End Encryption".
=>The bytes data type organizes the data within the range of (0,256). In Otherwords it allows us store 0 to 255 (256-1) values.
=>bytes data types doe not contain any symbolic natation for organizing or storing the data directly. But we can convert Other Type of Values into bytes type values by using bytes().

Syntax: varname=bytes(object) Here object can be str,int,float,bool,complex, list,tuple...etc

=>On the object of bytes , we ccan perform both Indexing and slicing Operations
=>An object of bytes maintains Insertion Order (Nothing but, whichever order, we insert the data, in the same order data will be displayed)
=>An object of bytes belongs to Immutable bcoz 'bytes' object does not support item assignment.

Examples:

```
>>> x=[10,34,56,256,37,0]
>>> print(x,type(x))-----[10, 34, 56, 256, 37, 0] <class 'list'>
>>> b=bytes(x)-----ValueError: bytes must be in range(0, 256)
>>> x=[-10,34,56,255,37,0]
>>> print(x,type(x))-----[-10, 34, 56, 255, 37, 0] <class 'list'>
>>> b=bytes(x)-----ValueError: bytes must be in range(0, 256)
```

```
>>> x=[10,0,56,255,37,56]
>>> print(x,type(x))-----[10, 0, 56, 255, 37, 56] <class 'list'>
>>> b=bytes(x)
>>> print(b,type(b))-----b'\n\x008\xff%8' <class 'bytes'>
```

```
>>> print(b[0])-----10
>>> print(b[-1])-----56
>>> print(b[2])-----56
>>> for v in b:
...     print(v)
...
10
0
56
255
```

```

37
56

>>> b[0:4]-----b'\n\x008\xff'
>>> for v in b[0:4]:
...     print(v)
...
10
0
56
255

>>> for v in b[::-1]:
...     print(v)
...
56
37
255
56
0
10

-----
>>> print(b,type(b),id(b))-----b'\n\x008\xff%8' <class 'bytes'> 1929024469056
>>> for v in b:
...     print(v)
...
10
0
56
255
37
56

>>> b[0]-----10 # bytes object belongs to Immutable object
>>> b[0]=123-----TypeError: 'bytes' object does not support item assignment
-----
```

===== Multability and Immutability =====

Mutability:

=>A Mutable Object is one, which allows us to perform modifications / additions in same Address.

Examples: list, set, dict, bytearray

Immutability:

=>An immutable object is one, It satisfies the following properties.

- 1) It never allows us to modify at the same address.
(It will modify and modified Values will place in new address)
- 2) Immutable objects does not support Item assignment.

Examples: int,float,bool, complex, str, bytes, range, tuple, set, frozenset, None

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

Properties:

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

=>The internal Implementation bytearray data type is that " To Implement End-to-End Encryption".

=>The bytearray data type organizes the data within the range of (0,256). In Otherwords it allows us store 0 to 255 (256-1) values.

=>bytearray data types doe not contain any symbolic natation for organizing or storing the data directly. But we can convert Other Type of Values into bytearray type values by using bytearray().

Syntax: varname=bytearray(object)

Here object can be str,int,float,bool,complex, list,tuple...etc

=>On the object of bytearray , we ccan perform both Indexing and slicing Operations

=>An object of bytearray maintains Insertion Order (Nothing but, whichever order, we insert the data , in the same order data will be displayed)

=>An object of bytearray belongs to mutable bcoz 'bytearray' allows us to perform different operation **at same address**.

NOTE: The Functionality bytearray is extactly similar to bytes but an object of bytes belongs to immutable where an object of bytearray is mutable.

Examples:

```
>>> lst=[234,12,34,256,17,90]
>>> print(lst,type(lst))-----[234, 12, 34, 256, 17, 90] <class 'list'>
>>> ba=bytearray(lst)-----ValueError: byte must be in range(0, 256)
>>> lst=[234,-12,0,34,256,17,90]
>>> print(lst,type(lst))-----[234, -12, 0, 34, 256, 17, 90] <class 'list'>
>>> ba=bytearray(lst)-----ValueError: byte must be in range(0, 256)

>>> lst=[234,0,12,34,255,17,90]
>>> print(lst,type(lst))-----[234, 0, 12, 34, 255, 17, 90] <class 'list'>
>>> ba=bytearray(lst)
>>> print(ba,type(ba),id(ba))--bytearray(b'\xeaa\x00\x0c\xff\x11Z') <class 'bytearray'>
1659347538032
>>> ba[0]-----234
>>> ba[-1]-----90
>>> for val in ba:
```

```

...     print(val)
...
234
0
12
34
255
17
90
>>> for val in ba[::-1]:
...     print(val)
...
90
17
255
34
12
0
234
>>> for val in ba[0:4]:
...     print(val)
...
234
0
12
34
>>> ba[0]----- 234
>>> ba[0]=123 # Update
>>> for val in ba:
...     print(val)
...
123
0
12
34
255
17
90
>>> print(ba,type(ba),id(ba))---bytearray(b'\x00\x0c\xff\x11Z') <class 'bytearray'>
1659347538032

```

4. range

Properties:

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

- =>The purpose of range data type is that " To Store range of Integer values by maintaining equal interval of values(Step)"
- =>An object of range data type belongs to Immutable bcoz range values can't updated / Chnaged
- =>On the object of range, we can perform Both Indexing and Slicing Operations.
- =>To Store range of Integer values by maintaining equal interval of values(Step), range data type provides 3 Syntaxes. They are
-

Syntax-1: varname=range(value)

- =>This Syntax generate range of values from 0 to Value-1
- =>here varname is an object <class, 'range'>
-

Examples:

```
>>> r=range(6)
>>> print(r,type(r))-----range(0, 6) <class 'range'>
>>> for v in r:
...     print(v)
...
0
1
2
3
4
5
>>> for v in r:
...     print(v,end=" ")-----0 1 2 3 4 5
>>> for v in range(10):
...     print(v)
...
0
1
2
3
4
5
6
7
8
9
```

Syntax-2: varname=range(Start,Stop)

- =>This Syntax generate range of values from start to stop-1
- =>here varname is an object <class, 'range'>
-

Examples:

```
>>> r=range(20,26)
```

```

>>> print(r,type(r))
range(20, 26) <class 'range'>
>>> for val in r:
...     print(val)
...
20
21
22
23
24
25
>>> for val in range(100,106):
...     print(val)
...
100
101
102
103
104
105
-----
>>> r=range(100,106)
>>> print(r,type(r))-----range(100, 106) <class 'range'>
>>> r[0]-----100
>>> r[-1]-----105
>>> r[120]-----IndexError: range object index out of range
>>> for val in r[::-1]:
...     print(val)
...
105
104
103
102
101
100
>>> for val in r[::-2]:
...     print(val)
...
100
102
104
>>> r[0]=123-----TypeError: 'range' object does not support item assignment
-----
```

NOTE:- Syntax-1 and Syntax-2 are maintaining Equal Interval of value (ie, default step val is 1) in forward direction.

Syntax-3: varname=range(Start,Stop, Step)

=>This Syntax generate range of values from start to stop-1 by maintaining Equal Interval of value with Step.

=>here varname is an object <class, 'range'>

Examples:

```
>>> r=range(100,111,2)
>>> print(r,type(r))-----range(100, 111, 2) <class 'range'>
>>> for val in r:
...     print(val)
```

```
...
100
102
104
106
108
110
```

```
>>> for val in range(10,101,30):
...     print(val)
```

```
...
10
40
70
100
```

Q1) 0 1 2 3 4 5 6 7 8 9 10-----range(0,11) OR range(11)

```
>>> for val in range(0,11):
...     print(val)
```

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

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

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

```
...     print(val)
```

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

15

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

>>> for val in range(300,306):

... print(val)

...

300

301

302

303

304

305

Q4) 10 12 14 16 18 20--range(10,21,2)

>>> for val in range(10,21,2):

... print(val)

...

10

12

14

16

18

20

Q5) 100 110 120 130 140 150--range(100,151,10)

>>> for val in range(100,151,10):

... print(val)

...

100

110

120

130

140

150

Q6) 300 400 500 600 700 800 900 1000--range(300,1001,100)

>>> for val in range(300,1001,100):

... print(val)

...

300

400

500

600

700

800

900

1000

Q7) -1 -2 -3 -4 -5 -6 -7 -8 -9 -10---range(-1,-11,-1)

>>> for val in range(-1,-11,-1):

... print(val)

...

-1

-2

-3

-4

-5

-6

-7

-8

-9

-10

Q8) -10 -9 -8 -7 -6 -5----- range(-10,-4) or range(-10,-4,1)

>>> for val in range(-10,-4):

... print(val)

...

-10

-9

-8

-7

-6

-5

(OR)

>>> for val in range(-10,-4,1):

... print(val)

...

-10

-9

-8

-7

-6

-5

Q9) -20 -22 -24 -26 -28 -30---range(-20,-31,-2)

>>> for val in range(-20,-31,-2):

... print(val)

...

-20

-22

-24

-26

-28

-30

Q10) -100 -90 -80 -70 -60 -50-----range(-100,-40,10)

```
>>> for val in range(-100,-40,10):  
...     print(val)  
...  
-100  
-90  
-80  
-70  
-60  
-50
```

Q11) -5 -4 -3 -2 -1 0 1 2 3 4-----range(-5,5,1) OR range(-5,5)

```
>>> for val in range(-5,5):  
...     print(val)
```

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

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

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

Q12) -50 -40 -30 -20 -10 0 10 20 30---range(-50,40,10)

```
>>> for val in range(-50,40,10):  
...     print(val)
```

```
...  
-50  
-40  
-30  
-20  
-10  
0
```

```
10
20
30
```

MiSc Questions

```
>>> r=range(100,201,10)
>>> print(r[-1])-----200
>>> print(r[1])-----110
>>> print(r[-2])-----190
>>> for val in r[::-2]:
...     print(val)
...
100
120
140
160
180
200
>>> for val in r[::-4]:
...     print(val)
...
100
140
180
>>> for val in r[1::2]:
...     print(val)
...
110
130
150
170
190
>>> for val in r[1::2][::-1]:
...     print(val)
...
190
170,150,130,110
```

List Category Data Types (Collection Data Types OR Data Structures)

=>The purpose of List Category Data Types is that "To store Multi Values either of Same type or Different Types Or Both Types with Unique and Duplicates in single object".

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

1. list (mutable)
 2. tuple (immutable)
-

1. list

Index

=>Properties of list

=>Operations on list

- a) indexing
 - b) Slicing

=>Pre-defined Functions in list

=>Programming Examples

=>Inner OR Nested List

=>Pre-defined Functions in list

=>Programming Examples

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 Multi Values either of Same type or Different Types or Both Types with Unique and Duplicates in single object".

=>The elements of list must be stored or organized within Square Brackets [] and Elements separated by comma.

=>An object of list maintains Insertion Order

=>An object of list belongs to Mutable.

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

=>In Python Programming, we have Two Types of List objects. They are

- 1. empty list
 - 2. non-empty list

1. Empty List:

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

=>Syntax: listobj=[]
OR
listobj=list()

2. Non-Empty List:

=>An Non-Empty List is one, which contains elements and whose length > 0
Examples:

`listobj=[Val1,Val2,.....val-n]`
(OR)
`listobj=list(object)`

Examples

```
>>> l1=[10,20,30,10,20,-45]
>>> print(l1,type(l1))-----[10, 20, 30, 10, 20, -45] <class 'list'>
```

```

>>> l2=[10,"Rossum","Python",34.56,True,2+3j,"Python"]
>>> print(l2,type(l2))-----[10, 'Rossum', 'Python', 34.56, True, (2+3j), 'Python']
<class 'list'>
-----
>>> l1=[10,"Rossum",34.56,"Python"]
>>> print(l1,type(l1),id(l1))-----[10, 'Rossum', 34.56, 'Python'] <class 'list'>
1151407161536
>>> l1[0]-----10
>>> l1[-1]-----'Python'
>>> l1[1:3]-----['Rossum', 34.56]
>>> l1[::-1]-----['Python', 34.56, 'Rossum', 10]
>>> print(l1,type(l1),id(l1))-----[10, 'Rossum', 34.56, 'Python'] <class 'list'>
1151407161536
>>> l1[0]=100 # Indexed Based Updation
>>> print(l1,type(l1),id(l1))-----[100, 'Rossum', 34.56, 'Python'] <class 'list'>
1151407161536
>>> l1[0:2]=[200,"Guido Van"] # Sliced Based Updation
>>> print(l1,type(l1),id(l1))-----[200, 'Guido Van', 34.56, 'Python'] <class 'list'>
1151407161536
-----
>>> l1=[]
>>> print(l1,type(l1),len(l1))-----[] <class 'list'> 0
      OR
>>> l2=list()
>>> print(l2,type(l2),len(l2))-----[] <class 'list'> 0
>>> l3=[10,"Travis",45.67,10,True]
>>> print(l3,type(l3),len(l3))----- [10, 'Travis', 45.67, 10, True] <class 'list'>
5
>>> s="PYTHON"
>>> l4=list(s)
>>> print(l4,type(l4),len(l4))-----['P', 'Y', 'T', 'H', 'O', 'N'] <class 'list'> 6
>>>

```

===== Pre-defined Functions in list =====

=>Along with the operations of Indexing and Slicing, we can perform Various Operations on list object by using Pre-defined Functions present in list object. They are

1) append()

Syntax: listobj.append(Value)

=>This Function is used for adding "value" to listobj at end(Adding values at end is called Appending)

Examples:

```
>>> l1=[]
>>> print(l1,type(l1),id(l1),len(l1))-----[] <class 'list'> 1151407161472      0
>>> l1.append(10)
>>> l1.append("KVR")
>>> print(l1,type(l1),id(l1),len(l1))-----[10, 'KVR'] <class 'list'>
1151407161472      2
>>> l1.append(True)
>>> l1.append("Hyd")
>>> l1.append(False)
>>> print(l1,type(l1),id(l1),len(l1))----- [10, 'KVR', True, 'Hyd', False] <class 'list'>
1151407161472      5
```

2) insert()

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

=>This Function is used for Inserting the specified values at Specified valid Index.

=>If we specify Invalid +Ve Index Then the Value Inserted at End.

=>If we specify Invalid -Ve Index Then the Value Inserted at BEGINning

Examples:

```
>>> l1=[10,"Rossum",34.56,"Python"]
>>> print(l1,len(l1))-----[10, 'Rossum', 34.56, 'Python'] 4
>>> l1.insert(2,"HYD")
>>> print(l1,len(l1))-----[10, 'Rossum', 'HYD', 34.56, 'Python'] 5
>>> l1[-2]=45.67
>>> print(l1,len(l1))-----[10, 'Rossum', 'HYD', 45.67, 'Python'] 5
>>> l1=[10,"Rossum",34.56,"Python"]
>>> print(l1,len(l1))-----[10, 'Rossum', 34.56, 'Python'] 4
>>> l1.insert(2,['HYD',34.56])
>>> print(l1,len(l1))-----[10, 'Rossum', ['HYD', 34.56], 34.56, 'Python'] 5
>>> l1=[10,"Rossum",34.56,"Python"]
>>> print(l1,len(l1))-----[10, 'Rossum', 34.56, 'Python'] 4
>>> l1.insert(-1,"Java")
>>> print(l1,len(l1))-----[10, 'Rossum', 34.56, 'Java', 'Python'] 5
```

```
>>> l1=[10,"Rossum",34.56,"Python"]
>>> print(l1,len(l1))-----[10, 'Rossum', 34.56, 'Python'] 4
>>> l1.insert(15,"KVR")
>>> print(l1,len(l1))-----[10, 'Rossum', 34.56, 'Python', 'KVR'] 5
>>> l1.insert(-15,"HYD")
>>> print(l1,len(l1))-----['HYD', 10, 'Rossum', 34.56, 'Python', 'KVR'] 6
```

3) remove()---Based on Values / Content

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

=>This Function is used for removing First Occurrence of Specified Value from list object

=>If the specified value does not exist then we get ValueError

Examples:

```
>>> l1=[10,"Rossum",34.56,10,"Java","Python"]
>>> print(l1,id(l1))-----[10, 'Rossum', 34.56, 10, 'Java', 'Python']
1151402904000
>>> l1.remove(10)
>>> print(l1,id(l1))-----['Rossum', 34.56, 10, 'Java', 'Python'] 1151402904000
>>> l1.remove(10)
>>> print(l1,id(l1))-----['Rossum', 34.56, 'Java', 'Python'] 1151402904000
>>> l1.remove("Rossum")
>>> print(l1,id(l1))-----[34.56, 'Java', 'Python'] 1151402904000
>>> l1.remove(100)----ValueError: list.remove(x): x not in list
-----
```

```
>>> l1=[]
>>> l1.remove(100)-----ValueError: list.remove(x): x not in list
>>> [].remove(10.23)-----ValueError: list.remove(x): x not in list
>>> list().remove("Python")-----ValueError: list.remove(x): x not in list
-----
```

4) pop(index)---Index Based

Syntax: listobj.pop(index)

=>This Function is used for removing Values of list object Based on Index.

=>If we enter Invalid Index then we get IndexError.

Examples:

```
>>> l1=[10,"Rossum",34.56,10,"Java","Python"]
>>> print(l1,id(l1))-----[10, 'Rossum', 34.56, 10, 'Java', 'Python'] 1151407166400
>>> l1.pop(3)-----10
>>> print(l1,id(l1))-----[10, 'Rossum', 34.56, 'Java', 'Python'] 1151407166400
>>> l1.pop(0)-----10
>>> print(l1,id(l1))-----['Rossum', 34.56, 'Java', 'Python'] 1151407166400
>>> l1.pop(-2)-----'Java'
>>> print(l1,id(l1))-----['Rossum', 34.56, 'Python'] 1151407166400
>>> l1.pop(10)-----IndexError: pop index out of range
>>> [].pop(2)-----IndexError: pop from empty list
>>> list().pop(0)-----IndexError: pop from empty list
>>> [10,20,30,40].pop(12)-----IndexError: pop index out of range
-----
```

5) pop()

Syntax: listobj.pop()

=>This Function is used for removing last element OR Last Indexed Element always.

=>If we call pop() on empty list then we get IndexError.

Examples:

```
>>> l1=[10,20,"Python","Java","Hyd",23.45]
>>> print(l1,id(l1))-----[10, 20, 'Python', 'Java', 'Hyd', 23.45] 1979565690752
-----
```

```

>>> l1.pop()-----23.45
>>> print(l1,id(l1))-----[10, 20, 'Python', 'Java', 'Hyd'] 1979565690752
>>> l1.pop()-----'Hyd'
>>> print(l1,id(l1))-----[10, 20, 'Python', 'Java'] 1979565690752
>>> l1.pop()-----'Java'
>>> print(l1,id(l1))-----[10, 20, 'Python'] 1979565690752
>>> l1.pop()-----'Python'
>>> print(l1,id(l1))-----[10, 20] 1979565690752
>>> l1.pop()-----20
>>> print(l1,id(l1))-----[10] 1979565690752
>>> l1.pop()-----10
>>> print(l1,id(l1))-----[] 1979565690752
>>> l1.pop()-----IndexError: pop from empty list
>>> list().pop()-----IndexError: pop from empty list
>>> [].pop()-----IndexError: pop from empty list

```

NOTE: del operator

Syntax1: `del listobj[index]` ---Removing Elements of list based on Index
 Syntax2: `del listobj[Begin:End]`--Removing Elements of list based on Slicing
 Syntax3: `del listobj`----Removing Entire List object

Examples

```

>>> l1=[10,20,"Python","Java","Hyd",23.45]
>>> print(l1,id(l1))-----[10, 20, 'Python', 'Java', 'Hyd', 23.45] 1979565690752
>>> del l1[-3] # Based Indexing
>>> print(l1,id(l1))-----[10, 20, 'Python', 'Hyd', 23.45] 1979565690752
>>> del l1[1] # Based Indexing
>>> print(l1,id(l1))-----[10, 'Python', 'Hyd', 23.45] 1979565690752
>>> del l1[1:3]
>>> print(l1,id(l1))-----[10, 23.45] 1979565690752
>>> l1=[10,20,"Python","Java","Hyd",23.45]
>>> print(l1,id(l1))-----[10, 20, 'Python', 'Java', 'Hyd', 23.45] 1979574587072
>>> del l1[::-2] # Based on Slicing
>>> print(l1,id(l1))-----[20, 'Java', 23.45] 1979574587072
>>> del l1-----# Removing Complete object
>>> print(l1,id(l1))-----NameError: name 'l1' is not defined

```

6) count()

=>Syntax: listobj.count(Value)

=>This Function is used for Counting Number of Occurrences of Specified Value.
 =>If the specified value does not exist then we get 0 as count.

Examples:

```

>>> l1=[10,20,10,30,40,10,20,10,50,10]
>>> print(l1)-----[10, 20, 10, 30, 40, 10, 20, 10, 50, 10]
>>> l1.count(10)-----5

```

```

>>> l1.count(20)-----2
>>> l1.count(30)-----1
>>> l1.count(40)-----1
>>> l1.count(50)-----1
>>> l1.count(500)-----0
>>> l1.count(0)-----0
>>> l1.count("Python")----0
>>> l2=["Apple","Kiwi","Apple","Apple","Banana"]
>>> print(l2)-----['Apple', 'Kiwi', 'Apple', 'Apple', 'Banana']
>>> l2.count("Apple")-----3
>>> l2.count("apple")-----0

```

7) copy()

Syntax: listobj2=listobj1.copy()

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

=>Examples:

```

>>> l1=[10,"Rossum"]
>>> print(l1,id(l1))-----[10, 'Rossum'] 1979570325952
>>> l2=l1.copy() # Sahhlow Copy
>>> print(l2,id(l2))-----[10, 'Rossum'] 1979574587072
>>> l1.append("Python")
>>> l2.insert(1,"Java")
>>> print(l1,id(l1))-----[10, 'Rossum', 'Python'] 1979570325952
>>> print(l2,id(l2))-----[10, 'Java', 'Rossum'] 1979574587072

```

Deep Copy

```

>>> l1=[10,"Rossum"]
>>> print(l1,id(l1))-----[10, 'Rossum'] 1979565690752
>>> l2=l1 # Deep Copy
>>> print(l2,id(l2))-----[10, 'Rossum'] 1979565690752
>>> l1.append("Python")
>>> print(l1,id(l1))-----[10, 'Rossum', 'Python'] 1979565690752
>>> print(l2,id(l2))-----[10, 'Rossum', 'Python'] 1979565690752

```

8. index()

Syntax: listobj.index(value)

=>This Function is used for obtaining First Occurence Index of Specified value.

=>If the specified value does not exist then we get ValueError

Examples:

```

>>> l1=[10,20,30,40,50]
>>> print(l1)-----[10, 20, 30, 40, 50]
>>> l1.index(10)-----0
>>> l1.index(50)-----4

```

```
>>> l1.index(30)-----2
-----
>>> l1=[10,20,10,40,50,10,20]
>>> print(l1)-----[10, 20, 10, 40, 50, 10, 20]
>>> l1.index(10)-----0
>>> for i,v in enumerate(l1):
...     print(i,"-->",v)
...
          0 --> 10
          1 --> 20
          2 --> 10
          3 --> 40
          4 --> 50
          5 --> 10
          6 --> 20
```

```
>>> l1=[10,20,30,40,50]
>>> print(l1)-----[10, 20, 30, 40, 50]
>>> l1.index(100)-----ValueError: 100 is not in list
```

9) reverse()

Syntax: listobj.reverse()

=>This Function is used for obtaining Reverse of given list of elements (Front Elements Back and Back to Elements to Front)

Examples:

```
>>> l1=[10,"Travis",34.56,True,2+3j]
>>> print(l1,id(l1))-----[10, 'Travis', 34.56, True, (2+3j)] 1979570325952
>>> l1.reverse()
>>> print(l1,id(l1))-----[(2+3j), True, 34.56, 'Travis', 10] 1979570325952
```

```
>>> l1=[10,"Travis",34.56,True,2+3j]
>>> print(l1,id(l1))-----[10, 'Travis', 34.56, True, (2+3j)] 1979574587072
>>> l2=l1.reverse() # Special Point
>>> print(l2,type(l2))-----None <class 'NoneType'>
>>> print(l1,type(l1))-----[(2+3j), True, 34.56, 'Travis', 10] <class 'list'>
```

10) sort()

=>Syntax1: listobj.sort()---Gives in Ascending Order

=>Syntax2: listobj.sort(reverse=False)---Gives in Ascending Order

=>Syntax3: listobj.sort(reverse=True)---Gives in Descending Order

Examples:

```
>>> l1=[10,-34,23,67,10,-6]
>>> print(l1,id(l1))-----[10, -34, 23, 67, 10, -6] 1979565690752
>>> l1.sort()
>>> print(l1,id(l1))-----[-34, -6, 10, 10, 23, 67] 1979565690752
```

```

>>> l1.reverse()
>>> print(l1,id(l1))-----[67, 23, 10, 10, -6, -34] 1979565690752
-----
>>> l1=["KIWI","APPLE","GUAVA","WMILLON","TRUMP","TRAVIS"]
>>> print(l1)-----['KIWI', 'APPLE', 'GUAVA', 'WMILLON', 'TRUMP',
'TRAVIS']
>>> l1.sort()
>>> print(l1)-----['APPLE', 'GUAVA', 'KIWI', 'TRAVIS', 'TRUMP',
'WMILLON']
>>> l1.reverse()
>>> print(l1)-----['WMILLON', 'TRUMP', 'TRAVIS', 'KIWI', 'GUAVA', 'APPLE']
-----
>>> l1=[10,"python",2+3j,True]
>>> l1.sort()-----TypeError: '<' not supported between instances of 'str' and 'int'
-----
>>> l1=[10,-34,23,67,10,-6]
>>> print(l1,id(l1))-----[10, -34, 23, 67, 10, -6] 1979574587072
>>> l1.sort(reverse=True)
>>> print(l1,id(l1))-----[67, 23, 10, 10, -6, -34] 1979574587072
-----
>>> l1=[10,-34,23,67,10,-6]
>>> print(l1,id(l1))-----[10, -34, 23, 67, 10, -6] 1979565690752
>>> l1.sort(reverse=False)
>>> print(l1,id(l1))-----[-34, -6, 10, 10, 23, 67] 1979565690752
-----
>>> l1=[10,-34,23,67,10,-6]
>>> print(l1,id(l1))-----[10, -34, 23, 67, 10, -6] 1979574587072
>>> l1.sort()
>>> print(l1,id(l1))-----[-34, -6, 10, 10, 23, 67] 1979574587072
-----
```

11) extend()

Syntax: listobj1.extend(listobj2)

=>This Function is used for extending the functionality of listobj1 with the elements of listobj2(Merging) only but not multiple list objects.

=>**NOTE:** listobj1.extend(listobj2, listobj3...listobj-n)----Error

=>In order to merge multiple list objects elements into single list object, we use + operator

=>Syntax: listobj1=listobj1+listobj2+....+listobj-n

Examples:

```

>>> l1=[10,20,30,40]
>>> l2=["Python","Java","C"]
>>> print(l1,id(l1))-----[10, 20, 30, 40] 1979565690752
>>> print(l2,id(l2))-----['Python', 'Java', 'C'] 1979574587072
>>> l1.extend(l2)
>>> print(l1,id(l1))-----[10, 20, 30, 40, 'Python', 'Java', 'C'] 1979565690752
>>> print(l2,id(l2))-----['Python', 'Java', 'C'] 1979574587072
-----
>>> l1=[10,20]
```

```

>>> l2=["Python","C"]
>>> l3=[1.2,0.2]
>>> print(l1)-----[10, 20]
>>> print(l2)-----['Python', 'C']
>>> print(l3)-----[1.2, 0.2]
>>> l1.extend(l2,l3)-----TypeError: list.extend() takes exactly one argument (2
given)
-----
>>> l1=l1+l2+l3
>>> print(l1)-----[10, 20, 'Python', 'C', 1.2, 0.2]
-----
```

===== Inner or Nested List =====

=>The Process of Defining one list inside of another list is called Inner or Nested List
=>Syntax:- listobj=[Val1, Val2....[Val11,Val12...Val1n] , [Val21,Val22..Val2n...], Val-n]
=>Here Val1,Val2...Val-n are called Values of Outer List
=>Here Val11,Val12...Val-1n are called Values of one Inner List
=>Here Val21,Val22...Val-2n are called Values of another Inner List
=>In inner list we can perform Both Indexing and Slicing Operations
=>On Inner List, we can apply all pre-defined functions of list.

Examples:

```

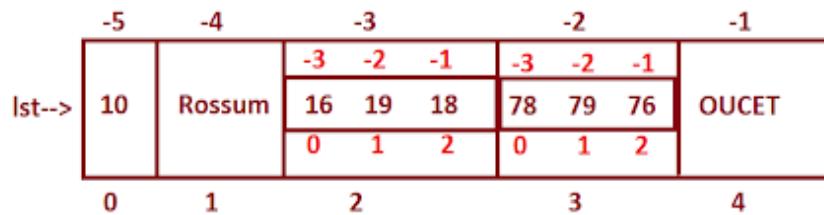
>>> studlist=[100,"DLPrince",[18,16,19],[78,66,79],"OUCET"]
>>> print(studlist,type(studlist))--[100, 'DLPrince', [18, 16, 19], [78, 66, 79], 'OUCET']-
<class 'list'>
>>> print(studlist[2],type(studlist[2]))---[18, 16, 19] <class 'list'>
>>> print(studlist[-2],type(studlist[-2]))---[78, 66, 79] <class 'list'>
>>> studlist[-3].append(17)
>>> print(studlist,type(studlist))--[100, 'DLPrince', [18, 16, 19, 17], [78, 66, 79],
                                         'OUCET'] <class 'list'>
>>> studlist[3].append(68)
>>> print(studlist,type(studlist))---[100, 'DLPrince', [18, 16, 19, 17], [78, 66, 79, 68],
                                         'OUCET'] <class 'list'>
>>> studlist[2].sort()
>>> print(studlist,type(studlist))---[100, 'DLPrince', [16, 17, 18, 19], [78, 66, 79, 68],
                                         'OUCET'] <class 'list'>
>>> studlist[3].sort(reverse=True)
>>> print(studlist,type(studlist))----[100, 'DLPrince', [16, 17, 18, 19], [79, 78, 68, 66],
                                         'OUCET'] <class 'list'>
>>> studlist[2].pop(-2)----18
>>> studlist[-2].remove(68)
>>> print(studlist,type(studlist))--[100, 'DLPrince', [16, 17, 19], [79, 78, 66], 'OUCET']
                                         <class 'list'>
>>> studlist[2].clear()
>>> print(studlist,type(studlist))--[100, 'DLPrince', [], [79, 78, 66], 'OUCET'] <class
                                         'list'>
```

```

>>> studlist[2].append(14)
>>> print(studlist,type(studlist))---[100, 'DLPrince', [14], [79, 78, 66], 'OUCET'] <class
'list'>
>>> del studlist[2]
>>> del studlist[-2]
>>> print(studlist,type(studlist))---[100, 'DLPrince', 'OUCET'] <class 'list'>
>>> studlist.insert(2,[17,15,14,19])
>>> print(studlist,type(studlist))---[100, 'DLPrince', [17, 15, 14, 19], 'OUCET'] <class
'list'>
>>> studlist.append([66,78,65,76])
>>> print(studlist,type(studlist))---[100, 'DLPrince', [17, 15, 14, 19], 'OUCET', [66, 78,
65, 76]] <class 'list'>
>>> studlist[-2]---'OUCET'
>>> len(studlist)----5
>>> len(studlist[2])---4
>>> len(studlist[-1])---4

```

Memory Mgmt Diagram for Inner List



2. tuple

Index

- =>Properties of tuple
- =>Operations on tuple
 - a) indexing
 - b) Slicing
- =>Pre-defined Functions in tuple
- =>Programming Examples

=>Inner OR Nested tuple
 =>Pre-defined Functions in nested tuple
 =>Combination of list and tuples--Most Imp

Properties of 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 Multi Values either of Same type or Different Types or Both Types with Unique and Duplicates in single object".
 =>The elements of tuple must be stored or organized within braces () and Elements separated by comma.
 =>An object of tuple maintains Insertion Order
 =>An object of tuple belongs to Immutable bcoz tuple' object does not support item assignment.
 =>On the object of tuple, we can perform both Indexing and Slicing Operations.
 =>In Python Programming, we have Two Types of tuple objects. They are
 1. empty tuple
 2. non-empty tuple

1. Empty tuple:

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

=>**Syntax:** **tplobj=()**
 OR
tplobj=tuple()

2. Non-Empty tuple:

=>An Non-Empty tuple is one, which contains elements and whose length > 0
 Examples:

tplobj=[(a1,Val2,.....val-n)]
 (OR)
tplobj=tuple(object)
 (OR)
tplobj=val1,val2....val-n

NOTE: It is always recommended to Choose tuple data type for representing OR Storing Constant Values.

Examples

```
>>> t1=(10,20,30,10,40,56)
>>> print(t1,type(t1))----(10, 20, 30, 10, 40, 56) <class 'tuple'>
>>> t2=(10,"Rossum",45.67,"HYD")
>>> print(t2,type(t2))----(10, 'Rossum', 45.67, 'HYD') <class 'tuple'>

>>> t3=10,"KVR","HYD","PYTHON",2+3j
>>> print(t3,type(t3))----(10, 'KVR', 'HYD', 'PYTHON', (2+3j)) <class 'tuple'>
>>> len(t3)-----5
```

```

>>> t4=tuple()
>>> print(t4,type(t4))-----() <class 'tuple'>
>>> len(t4)-----0
>>> t5=()
>>> print(t5,type(t5))-----() <class 'tuple'>
>>> len(t5)-----0
-----
>>> t2=(10,"Rossum",45.67,"HYD")
>>> print(t2,type(t2),id(t2))-----(10, 'Rossum', 45.67, 'HYD') <class 'tuple'>
2586850634352
>>> t2[0]-----10
>>> t2[0:3]-----(10, 'Rossum', 45.67)
>>> t2[::-1]-----('HYD', 45.67, 'Rossum', 10)
>>> t2=(10,"Rossum",45.67,"HYD")
>>> print(t2,type(t2),id(t2))-----(10, 'Rossum', 45.67, 'HYD') <class 'tuple'>
2586850639632
>>> t2[0]=100-----TypeError: 'tuple' object does not support item
assignment
-----
```

===== Pre-defined Function in tuple =====

=>We know that on the object of tuple we can perform Both Indexing and Slicing Operations.
=>Along with these operations, we can also perform other operations by using the following pre-defined Functions.

- 1)index()
- 2)count()

Examples:

```

>>> t1=(10,"RS",45.67)
>>> print(t1,type(t1))-----(10, 'RS', 45.67) <class 'tuple'>
>>> t1.index(10)-----0
>>> t1.index("RS")-----1
>>> t1=(10,"RS",45.67)
>>> print(t1,type(t1))-----(10, 'RS', 45.67) <class 'tuple'>
>>> t1.count(10)-----1
>>> t1.count(100)-----0
>>> t1=(10,0,10,10,20,0,10)
>>> print(t1,type(t1))-----(10, 0, 10, 10, 20, 0, 10) <class 'tuple'>
>>> t1.count(10)-----4
>>> t1.count(0)-----2
>>> t1.count(100)-----0
-----
```

The Functions not present in tuple

append()

```
insert()
remove()
clear()
pop(index)
pop()
reverse()
sort()
copy()
extend()
```

NOTE:- By Using del Operator we can't delete values of tuple object By using Indexing and slicing but we can delete entire object

Examples:

```
>>> t1=(10,-34,0,10,23,56,76,21)
>>> print(t1,type(t1))----(10, -34, 0, 10, 23, 56, 76, 21) <class 'tuple'>
>>> del t1[0]----TypeError: 'tuple' object doesn't support item deletion
>>> del t1[0:4]---TypeError: 'tuple' object does not support item deletion
>>> del t1 # Here we are removing complete object.
>>> print(t1,type(t1))----NameError: name 't1' is not defined.
```

MOST IMP:

sorted(): This Function is used for Sorting the data of immutable object tuple and gives the sorted data in the form of list.

=>Syntax: **listobj=sorted(tuple object)**

Examples:

```
>>> t1=(10,23,-56,-1,13,15,6,-2)
>>> print(t1,type(t1))----(10, 23, -56, -1, 13, 15, 6, -2) <class 'tuple'>
>>> t1.sort()-----AttributeError: 'tuple' object has no attribute 'sort'
>>> x=sorted(t1)
>>> print(x,type(x))----[-56, -2, -1, 6, 10, 13, 15, 23] <class 'list'>
>>> print(t1,type(t1))----(10, 23, -56, -1, 13, 15, 6, -2) <class 'tuple'>
>>> t1=tuple(x) # Converted sorted list into tuple
>>> print(t1,type(t1))----(-56, -2, -1, 6, 10, 13, 15, 23) <class 'tuple'>
>>> t2=t1[::-1]
>>> print(t2,type(t2))----(23, 15, 13, 10, 6, -1, -2, -56) <class 'tuple'>
      OR
>>> t1=(10,-4,12,34,16,-6,0,15)
>>> print(t1,type(t1))----(10, -4, 12, 34, 16, -6, 0, 15) <class 'tuple'>
>>> l1=list(t1)
>>> print(l1,type(l1))----[10, -4, 12, 34, 16, -6, 0, 15] <class 'list'>
>>> l1.sort()
>>> print(l1,type(l1))----[-6, -4, 0, 10, 12, 15, 16, 34] <class 'list'>
>>> t1=tuple(l1)
>>> print(t1,type(t1))----(-6, -4, 0, 10, 12, 15, 16, 34) <class 'tuple'>
>>>t1=t1[::-1]
```

```
>>> print(t1,type(t1))-----(34, 16, 15, 12, 10, 0, -4, -6) <class 'tuple'>
```

===== Inner (OR) Nested tuple =====

=>The Process of Defining One tuple in another tuple is called Inner or Nested tuple

=>**Syntax:-**

tplobj1=(Val1,Val2....(Val11,Val12....Val1n).....(Val21,Val22...Val2n).....Val-n)

=>Here (Val11,Val12....Val1n) is called One Inner OR Nested tuple

(Val21,Val22...Val2n) is called another Inner OR Nested tuple

=> (Val1,Val2....(Val11,Val12....Val1n).....(Val21,Val22...Val2n)....Val-n) is called Outer tuple

=>All the pre-defined Functions of tuple can be applied on Inner or Nested tuple.

=>On Inner or Nested tuple we can perform Index and Slicing Operations.

```
>>> sf=(10,"RS", (17,18,16), (78,66,79), "OUCET")
```

```
>>> print(sf,type(sf))-----(10, 'RS', (17, 18, 16), (78, 66, 79), 'OUCET') <class 'tuple'>
```

```
>>> print(sf[0])----10
```

```
>>> print(sf[2],type(sf[2]),type(sf))-----(17, 18, 16) <class 'tuple'> <class 'tuple'>
```

```
>>> print(sf[0:3])-----(10, 'RS', (17, 18, 16))
```

```
>>> sf=(10,"RS", [17,18,16],(78,66,79),"OUCET")
```

```
>>> print(sf,type(sf))-----(10, 'RS', [17, 18, 16], (78, 66, 79), 'OUCET') <class 'tuple'>
```

```
>>> print(sf[2],type(sf[2]),type(sf))-----[17, 18, 16] <class 'list'> <class 'tuple'>
```

```
>>> sf[2].append(12)
```

```
>>> print(sf[2],type(sf[2]),type(sf))-----[17, 18, 16, 12] <class 'list'> <class 'tuple'>
```

```
>>> print(sf,type(sf))-----(10, 'RS', [17, 18, 16, 12], (78, 66, 79), 'OUCET') <class 'tuple'>
```

```
>>> sf[3].append(12)--AttributeError: 'tuple' object has no attribute 'append'
```

```
>>> sf=[10,"RS",[17,18,16],(78,66,79),"OUCET"]
```

```
>>> print(sf,type(sf))-----[10, 'RS', [17, 18, 16], (78, 66, 79), 'OUCET'] <class 'list'>
```

```
>>> print(sf[2],type(sf[2]),type(sf))-----[17, 18, 16] <class 'list'> <class 'list'>
```

```
>>> print(sf[3],type(sf[3]),type(sf))-----(78, 66, 79) <class 'tuple'> <class 'list'>
```

NOTE:

=>One can define One List in another List

=>One can define One Tuple in another Tuple

=>One can define One List in another Tuple (tuple of lists)

=>One can define One tuple in another List (list of tuples)

```
>>> print(t1,type(t1))
```

```
(10, 'Rossum', [16, 18, 17], ('CSE', 'AI', 'DS'), 'OUCET') <class 'tuple'>
```

```
>>> print(t1[2],type(t1[2]))-----[16, 18, 17] <class 'list'>
```

```
>>> print(t1[3],type(t1[3]))-----('CSE', 'AI', 'DS') <class 'tuple'>
```

```

>>> t1[2].append("KVR")
>>> print(t1,type(t1))--(10, 'Rossum', [16, 18, 17, 'KVR'], ('CSE', 'AI', 'DS'), 'OUCET')
<class 'tuple'>
>>> t1[3].append("Python")----AttributeError: 'tuple' object has no attribute 'append'
>>> t1[2][1]-----18
>>> t1[2][-2]-----17
>>> t1[2][-3]-----18
>>> k=sorted(t1[-2])
>>> k-----['AI', 'CSE', 'DS']
>>> t1[3]=k-----TypeError: 'tuple' object does not support item assignment
>>> l1=list(t1)---
>>> l1-----[10, 'Rossum', [16, 18, 17, 'KVR'], ('CSE', 'AI', 'DS'), 'OUCET']
>>> l1[3]=k
>>> l1-----[10, 'Rossum', [16, 18, 17, 'KVR'], ('AI', 'CSE', 'DS'), 'OUCET']
>>> y=tuple(l1[3])
>>> l1[3]=y
>>> l1-----[10, 'Rossum', [16, 18, 17, 'KVR'], ('AI', 'CSE', 'DS'), 'OUCET']
>>> t2=tuple(l1)
>>> t2-----(10, 'Rossum', [16, 18, 17, 'KVR'], ('AI', 'CSE', 'DS'), 'OUCET')

-----
>>> t1=('AI', 'cSE', 'DS')
>>> k=sorted(t1)
>>> k-----['AI', 'DS', 'cSE']
-----
```

===== V. Set Category Data Types(Collection Data Types OR Data Structures) =====

=>The purpose of Set Category Data Types is that "To store Multi Values either of Same type or Different Types Or Both Types with Unique in single object ".
=>Set Category Data Types never allows us to store Duplicate Values.
=>Set Category Data Types contains 2 Data types. They are

1. set
2. frozenset

===== 1. set =====

Index

- =>Properties of set
- =>Types of sets
- =>Operations on sets
- =>Pre-defined Functions in sets
- =>Inner OR Nested Sets
- =>Pre-defined Functions in nested sets
- =>sets with list and tuple
- =>Programming Examples

Properties of set

=>'set' is one of the pre-defined class name and treated as set data type
=>The purpose of set data type is that " To store Multi Values either of Same type or Different Types Or Both Types with Unique in single object ".
=>The elements of set must be organized / stored within curly braces {} and elements must be separated by comma.
=>An object of set never maintains Insertion Order bcoz set concept displays any one of the Possibility.
=>On the object of set, we can't perform Indexing and Slicing Operations bcoz set object never maintains Insertion Order.
=>An object of set belongs to Both Immutable ('set' object does not support item assignment) and Mutable (we can add the data to set object at same address).
=>With set data type, we can create two types of set objects. They are

1. Empty Set
 2. Non-Empty Set
-

>>Empty Set

=>An empty set is one, which does not contain any elements and whose length is 0
=>Syntax: varname=set()

Non-Empty Set

=>A non empty set is one, which contains elements and whose length > 0
=>Syntax: varname={Val1, val2...val-n}
 (OR)
 varname=set(object)

Examples:

```
>>> s1={10,10,10,10,10,10}
>>> print(s1,type(s1))-----{10} <class 'set'

>>> s1={10,20,30,40,10,20,60,70}
>>> print(s1,type(s1))-----{20, 70, 40, 10, 60, 30} <class 'set'
>>> s2={10,"SaiRam","Python",45.67,True,"Python"}
>>> print(s2,type(s2))-----{True, 'SaiRam', 10, 'Python', 45.67} <class 'set'
>>> s2[0]-----TypeError: 'set' object is not subscriptable
>>> s2[-1]-----TypeError: 'set' object is not subscriptable
```

```
>>> s1={}
>>> print(s1,type(s1))-----{} <class 'dict'
>>> s1=set()
>>> print(s1,type(s1), len(s1))-----set() <class 'set'> 0
>>> s1={10,"Rossum",45.67}
```

```

>>> print(s1,type(s1), len(s1))-----{10, 'Rossum', 45.67} <class 'set'> 3
-----
>>> t1=(10,20,30,40,10,10)
>>> print(t1,type(t1))-----(10, 20, 30, 40, 10, 10) <class 'tuple'>
>>> s1=set(t1)
>>> print(s1,type(s1))-----{40, 10, 20, 30} <class 'set'>
>>> l1=[10,10,10,"Python","Python","Rossum"]
>>> print(l1,type(l1))-----[10, 10, 10, 'Python', 'Python', 'Rossum'] <class 'list'>
>>> s1=set(l1)
>>> print(s1,type(s1))-----{10, 'Rossum', 'Python'} <class 'set'>
-----
>>> s2={10,"SaiRam","Python",45.67,True,"Python"}
>>> print(s2,type(s2),id(s2))-----{True, 'SaiRam', 10, 'Python', 45.67} <class
'set'> 1923469348864
>>> s2[0]=10-----TypeError: 'set' object does not support item assignment
>>> s2.add("KVR")
>>> print(s2,type(s2),id(s2))-----{True, 'SaiRam', 'KVR', 10, 'Python', 45.67} <class
'set'> 1923469348864
-----
```

===== Pre-defined Functions in sets =====

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

1) clear()

=>Syntax: **setobj.clear()**

=>This function is used for removing all the elements of set

=>When we call clear() upon on empty set object then we get "no output" OR None

Examples

```

>>> s2={10,"SaiRam","Python",45.67,True,"Python"}
>>> print(s2,len(s2), id(s2))-----{True, 'SaiRam', 10, 'Python', 45.67} 5
1923469350208
>>> s2.clear()
>>> print(s2,len(s2), id(s2))-----set() 0 1923469350208
-----
```

```

>>> print(s2)-----set()
>>> s2.clear()---- Empty / No result
      OR
>>> print(s2.clear())-----None
-----
```

```

>>> set().clear()---- Empty / No result
>>> print(set().clear())-----None
-----
```

2) add()

=>Syntax: setobj.add(Value)

=>This Function is used for adding any type of values of set object

Examples:

```
>>> s1={10,"Lalitha"}
>>> print(s1,type(s1),id(s1))-----{'Lalitha', 10} <class 'set'> 1923469349536
>>> s1.add("Python")
>>> print(s1,type(s1),id(s1))-----{'Lalitha', 10, 'Python'} <class 'set'>
1923469349536
>>> s1.add("HYD")
>>> print(s1,type(s1),id(s1))-----{'HYD', 'Lalitha', 10, 'Python'} <class 'set'>
1923469349536
```

```
>>> s1=set()
>>> print(s1,type(s1),id(s1))-----set() <class 'set'> 1923469349536
>>> s1.add(10)
>>> s1.add("sai Teja")
>>> s1.add(11.11)
>>> print(s1,type(s1),id(s1))-----{'sai Teja', 10, 11.11} <class 'set'>
1923469349536
```

3) remove()

Syntax: setobj.remove(Value)

=>This Function is used for removing the element of non-empty set object

=>if Value does not exist then we get KeyError

Examples:

```
>>> s2={10,"SaiRam","Python",45.67,True,"Python"}
>>> print(s2,id(s2))-----{True, 'SaiRam', 10, 'Python', 45.67}
1923469348864
>>> s2.remove("Python")
>>> print(s2,id(s2))-----{True, 'SaiRam', 10, 45.67} 1923469348864
>>> s2.remove(10)
>>> print(s2,id(s2))-----{True, 'SaiRam', 45.67} 1923469348864
>>> s2.remove(True)
>>> print(s2,id(s2))-----{'SaiRam', 45.67} 1923469348864
>>> s2.remove(100)-----KeyError: 100
```

```
>>> s1=set()
>>> s1.remove(10)-----KeyError: 10
>>> set().remove(11.11)-----KeyError: 11.1
```

4) discard()

=>Syntax: setobj.discard(Value)

=>This Function is used for Removing the Value from non-empty set object if Value present.

=>If the Value not present in set object then discard() will not give any error (No KeyError).

Examples:

```
>>> s2={10,"SaiRam","Python",45.67,True,"Python"}
>>> print(s2,id(s2))-----{True, 'SaiRam', 10, 'Python', 45.67} 1923469348416
>>> s2.discard(10)
>>> print(s2,id(s2))-----{True, 'SaiRam', 'Python', 45.67} 1923469348416
>>> s2.discard(True)
>>> print(s2,id(s2))-----{'SaiRam', 'Python', 45.67} 1923469348416
>>> s2.discard("Python")
>>> print(s2,id(s2))-----{'SaiRam', 45.67} 1923469348416
>>> s2.discard(100)-----No Output will Come / does not generate KeyError
>>> print(s2,id(s2))-----{'SaiRam', 45.67} 1923469348416
>>> s2.remove(100)-----KeyError: 100
```

```
>>> set().discard(100)-----Nothing is removed and No Output will Come
>>> set().remove(100)-----Nothing is removed But KeyError Came
                           KeyError: 100
```

5) pop()

=>Syntax: setobj.pop()

=>This Function is used for removing any Arbitrary Key (value of set) from non-empty setobject.

=>if we call pop() on empty set object then we get KeyError

Examples:

```
>>> s1={10,"Rossum",23.45,True,2+3j,"Python"}
>>> s1.pop()-----True
>>> s1.pop()-----23.45
>>> s1.pop()-----10
>>> s1.pop()-----'Rossum'
>>> s1.pop()-----'Python'
>>> s1.pop()-----(2+3j)
>>> s1.pop()-----KeyError: 'pop from an empty set'
>>> set().pop()-----KeyError: 'pop from an empty set'
```

Extra Care Point:

```
>>> s1={10,23.45,45,67,True,234,"Python",23}
>>> print(s1)-----{True, 67, 10, 234, 'Python', 45, 23, 23.45}
#After displaying the the output of set object, pop() removes always first elements
>>> s1.pop()-----True
>>> s1.pop()-----67
>>> s1.pop()-----10
```

```
>>> s1.pop()----234
>>> s1.pop()----'Python'
>>> s1.pop()----45
>>> s1.pop()----23
>>> s1.pop()----23.45
>>> s1.pop()----KeyError: 'pop from an empty set'
```

6) copy()

Syntax: `setobj2=setobj1.copy()`

=>This Function is used for Copying the content of One set object into another set object.

Examples:

```
>>> s1={10,"Anand",45.67,"OUCET"}
>>> print(s1,id(s1),type(s1))----{10, 'OUCET', 45.67, 'Anand'} 1272815226432
<class 'set'>
>>> s2=s1.copy() # Shallow Copy
>>> print(s2,id(s2),type(s2))----{10, 'OUCET', 45.67, 'Anand'} 1272815227776
<class 'set'>
>>> s1.add("Python")
>>> s2.add("Java")
>>> print(s1,id(s1),type(s1))----{10, 'Anand', 45.67, 'Python', 'OUCET'}
1272815226432 <class 'set'>
>>> print(s2,id(s2),type(s2))---{10, 'Java', 'Anand', 45.67, 'OUCET'} 1272815227776
<class 'set'>
```

7) issubset()

=>Syntax: `setobj1.issubset(setobj2)`

=>This Function Returns True provided all the elements of setobj1 present in setobj2

(OR)

=>This Function Returns True provided setobj2 Contains all the elements of setobj1

Examples:

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

8) issuperset()

Syntax: **setobj1.issuperset(setobj2)**

=>This Function Returns True Provided setobj1 contains all the elements of setobj2
(OR)

=>This Function Returns True Provided all the elements of setobj2 present in setobj1

Examples:

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

9) isdisjoint()

Syntax:- **setobj1.isdisjoint(setobj2)**

=>This Function Returns True Provided setobj1 and setobj2 does not contains common elements.

=>If there exist atleast one common element then It returns False.

Examples:

```
>>> s1={'a','e','i','u','o'}
>>> s2={'b','c',10,20,30}
>>> s3={'a','e','x','y','30'}
>>> s4={100,200}
>>> print(s1)-----{'o', 'e', 'a', 'i', 'u'}
>>> print(s2)-----{20, 10, 'b', 30, 'c'}
>>> print(s3)-----{'30', 'e', 'x', 'y', 'a'}
>>> print(s4)-----{200, 100}
>>> s1.isdisjoint(s2)-----True
>>> s1.isdisjoint(s3)-----False
>>> s1.isdisjoint(s4)-----True
>>> s2.isdisjoint(s3)-----True
>>> s2.isdisjoint(s4)-----True
```

10) union()

Syntax: **setobj3=setobj1.union(setobj2)**

(OR)

Syntax: **setobj3=setobj2.union(setobj1)**

=>This Function is used for obtaining all the elements of setobj1 and setobj2 uniquely.

Examples:

```
>>> s1={10,20,30}
>>> s2={30,40,50,60}
>>> print(s1)-----{10, 20, 30}
>>> print(s2)-----{40, 50, 60, 30}
>>> s3=s1.union(s2)
>>> print(s3,type(s3))-----{50, 20, 40, 10, 60, 30} <class 'set'>
>>> s4=s2.union(s1)
>>> print(s4,type(s4))-----{50, 20, 40, 10, 60, 30} <class 'set'>
```

11) intersection()

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

Syntax: setobj3=setobj1.intersection(setobj2)

=>This Function is used for obtaining common elements of setobj1 and setobj2 .

Examples:

```
>>> s1={10,20,30}
>>> s2={30,40,50,60}
>>> print(s1)-----{10, 20, 30}
>>> print(s2)-----{40, 50, 60, 30}
>>> s3=s1.intersection(s2)
>>> print(s3,type(s3))-----{30} <class 'set'>
>>> s4=s2.intersection(s1)
>>> print(s4,type(s4))-----{30} <class 'set'>
```

12) difference()

Syntax: setobj3=setobj1.difference(setobj2)

=>This Function eliminates common elements from both setobj1 and setobj2 and Takes Remaining Elements from setobj1 and Place them setobj3.

(OR)

Syntax: setobj3=setobj2.difference(setobj1)

=>This Function eliminates common elements from both setobj2 and setobj1 and Takes Remaining Elements from setobj2 and Place them setobj3.

Examples:

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

```
>>> print(s3,type(s3))-----{10, 20} <class 'set'
>>> s4=s2.difference(s1)
>>> print(s4,type(s4))-----{40, 50, 60} <class 'set'>
```

13) symmetric_difference()

Syntax: **setobj3=setobj1.symmetric_difference(setobj2)**
 (OR)

Syntax: **setobj3=setobj2.symmetric_difference(setobj1)**

=>This Function eliminates common elements from both setobj1 and setobj2 and Takes Remaining Elements from both setobj1 and setobj2 and Place them in setobj3.

Formula:

setobj3=setobj2.symmetric_difference(setobj1)
 is Eqv to
setobj3 = setobj1.union(setobj2).difference(setobj1.intersection(setobj2))

Examples:

```
>>> s1={10,20,30}
>>> s2={30,40,50,60}
>>> print(s1)-----{10, 20, 30}
>>> print(s2)-----{40, 50, 60, 30}
>> s3=s1.symmetric_difference(s2)
>>> print(s3,type(s3))-----{40, 10, 50, 20, 60} <class 'set'>
>>> s4=s2.symmetric_difference(s1)
>>> print(s4,type(s4))-----{40, 10, 50, 20, 60} <class 'set'>
#By Formula
>>> s5=s1.union(s2).difference(s1.intersection(s2))
>>> print(s5,type(s5))-----{40, 10, 50, 20, 60} <class 'set'>
*****
```

Problem Statement (See Paint File)

```
>>> cp={"sachin","rohit","kohli"}
>>> tp={"kohli","rossum","dennis"}
>>> print(cp)-----{'sachin', 'rohit', 'kohli'}
>>> print(tp)-----{'dennis', 'rossum', 'kohli'}
>>> allcptp=cp|tp # Bitwise OR
>>> print(allcptp)-----{'rossum', 'rohit', 'dennis', 'sachin', 'kohli'}
>>> allcptp=cp.union(tp)
>>> print(allcptp)-----{'rossum', 'rohit', 'dennis', 'sachin', 'kohli'}
```

```
>>> cp={"sachin","rohit","kohli"}
>>> tp={"kohli","rossum","dennis"}
>>> print(cp)-----{'sachin', 'rohit', 'kohli'}
>>> print(tp)-----{'dennis', 'rossum', 'kohli'}
```

```

>>> bothcptp=cp.intersection(tp)
>>> print(bothcptp)-----{'kohli'}
>>> bbothcptp=cp&tp # Biwise and
>>> print(bbothcptp)-----{'kohli'}
-----
>>> cp={"sachin","rohit","kohli"}
>>> tp={"kohli","rossum","dennis"}
>>> print(cp)-----{'sachin', 'rohit', 'kohli'}
>>> print(tp)-----{'dennis', 'rossum', 'kohli'}
>>> onlycp=cp.difference(tp)
>>> print(onlycp)-----{'sachin', 'rohit'}
>>> bonlycp=cp-tp
>>> print(bonlycp)-----{'sachin', 'rohit'}
>>> onlytp=tp.difference(cp)
>>> print(onlytp)-----{'dennis', 'rossum'}
>>> bonlytp=tp-cp
>>> print(bonlytp)-----{'dennis', 'rossum'}
-----
>>> cp={"sachin","rohit","kohli"}
>>> tp={"kohli","rossum","dennis"}
>>> print(cp)-----{'sachin', 'rohit', 'kohli'}
>>> print(tp)-----{'dennis', 'rossum', 'kohli'}
>>> exclcptp=cp.symmetric_difference(tp)
>>> print(exlcptp)-----{'rossum', 'rohit', 'dennis', 'sachin'}
>>> exclcptp=cp^tp # Bitwise XOR Operator
>>> print(exlcptp)-----{'rossum', 'rohit', 'dennis', 'sachin'}

```

14) update()

Syntax: setobj1.update(setobj2)

=>This Function is used for adding / updating setobj1 elements with setobj2 elements

Examples:

```

>>> s1={10,20,30}
>>> s2={15,25,35}
>>> print(s1)-----{10, 20, 30}
>>> print(s2)-----{25, 35, 15}
>>> s1.update(s2)
>>> print(s1)-----{35, 20, 25, 10, 30, 15}
>>> print(s2)-----{25, 35, 15}

```

```

>>> s1={10,20,30}
>>> s2={10,20,30}
>>> print(s1)-----{10, 20, 30}
>>> print(s2)-----{10, 20, 30}
>>> s1.update(s2)
>>> print(s1)-----{20, 10, 30}
>>> print(s2)-----{10, 20, 30}

```

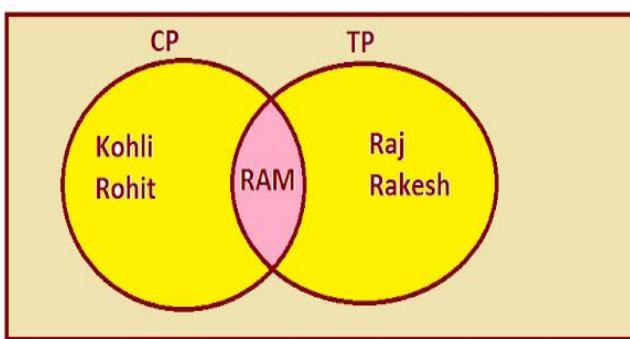
```
>>> s1={10,20,30}
>>> s2={10,20,35}
>>> print(s1)-----{10, 20, 30}
>>> print(s2)-----{10, 35, 20}
>>> s1.update(s2)
>>> print(s1)-----{35, 20, 10, 30}
>>> print(s2)-----{10, 35, 20}
```

Consider the following

```
cp={"Kohli","Rohit","Ram"}
tp={"Ram","Raj","Rakesh"}
```

Find The following

- Q1) Find all the Unique players who are playing all the Games --union()
- Q2) Find all the Players who are Playing Both Cricket and Tennis--intersection()
- Q3) Find all the Players who are Playing only Cricket But not tennis--difference()
- Q4) Find all the Players who are Playing only Tennis But not cricket --difference()
- Q5) Find all the Players who are EXCLUSIVELY Playing Cricket and tennis--



symmetric_difference()
e()

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 either Similar Type or Different Type or Both the Types in Single Object with Unique Values".
- =>The elements of frozenset must be obtained from different objects like set , tuple and list

Syntax: frozensetobj=frozenset(set/list/tuple)

- =>An Object of frozenset never maintains Insertion Order bcoz PVM can display any one of the possibility of elements of frozenset object.
- =>On the object of frozenset, we can't perform Indexing and Slicing Operations bcoz frozenset object never maintains Insertion Order.
- =>An object of frozenset belongs to Immutable bcoz 'frozenset' object does not support item assignment and not possible to modify / Change.
- =>we can create two types of frozenset objects. They are
 - a) Empty frozenset

b) Non-Empty frozenset

a) Empty frozenset:

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

=>**Syntax:** **frozensetobj=frozenset()**

b) Non-Empty frozenset:

=>A Non-Empty frozenset is one, which contains elements and whose length is >0

=>**Syntax:** **frozensetobj=frozenset({ val1, val2,val-n })**

=>**Syntax:** **frozensetobj=frozenset((val1, val2,val-n))**

=>**Syntax:** **frozensetobj=frozenset([val1, val2,val-n])**

=>Note: The Functionality of Frozenset is exactly similar to Set but set object belongs to both Mutable and Immutable whereas frozenset object belongs to immutable.

Examples:

```
>>> s1={10,20,30,40,10}
>>> print(s1,type(s1))-----{40, 10, 20, 30} <class 'set'>
>>> fs=frozenset(s1)
>>> print(fs,type(fs))-----frozenset({40, 10, 20, 30}) <class 'frozenset'>
>>> t1=(10,"Rossum",34.56,"Python")
>>> fs=frozenset(t1)
>>> print(fs,type(fs))-----frozenset({10, 34.56, 'Python', 'Rossum'}) <class
'frozenset'>
>>> fs[0]-----TypeError: 'frozenset' object is not subscriptable
>>> fs[0:3]-----TypeError: 'frozenset' object is not subscriptable
>>> len(fs)-----4
>>> fs1=frozenset()
>>> print(fs1,type(fs1))-----frozenset() <class 'frozenset'>
>>> len(fs1)-----0
>>>
>>> fs1.add(10)-----AttributeError: 'frozenset' object has no attribute 'add'
>>> fs-----frozenset({10, 34.56, 'Python', 'Rossum'})
>>> fs[0]=123-----TypeError: 'frozenset' object does not support item
assignment
>>> fs1.remove(10)-----AttributeError: 'frozenset' object has no attribute
'remove'
```

=====

Nested or Inner Properties of Set

=====

=>We can define

- a) Defining set inside of another set NOT POSSIBLE
- b) Defining set inside of another Tuple POSSIBLE
- c) Defining set inside of another List POSSIBLE

- d) Defining tuple inside of Another Set POSSIBLE**
e) Defining list inside of another set NOT POSSIBLE
-

Examples:

```
>>> s1={10,"Rossum",{10,15,16}, "OUCET"}----TypeError: unhashable type: 'set'
>>> s1=[10,"Rossum",[10,15,16], "OUCET"}----TypeError: unhashable type: 'list'
>>> s1={10,"Rossum", (10,15,16), "OUCET"}----
```

```
>>> print(s1,type(s1))---{'Rossum', 10, 'OUCET', (10, 15, 16)} <class 'set'>
-----
```

```
>>> t1=(10,"Rossum", {10,15,16}, "OUCET")
>>> print(t1[2],type(t1[2]),type(t1))---{10, 15, 16} <class 'set'> <class 'tuple'>
>>> l1=[10,"Rossum", {10,15,16}, "OUCET"]
>>> print(l1[2],type(l1[2]),type(l1))---{10, 15, 16} <class 'set'> <class 'list'>
```

===== Pre-Defined Functions in frozenset =====

=>frozenset contains the following Functions

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

NOTE:

```
>>> fs1=frozenset({10,20,30,409})
>>> print(fs1,type(fs1),id(fs1))-----frozenset({409, 10, 20, 30}) <class 'frozenset'>
```

2068835340960

```
>>> fs2=fs1.copy()
>>> print(fs2,type(fs2),id(fs2))-----frozenset({409, 10, 20, 30}) <class 'frozenset'>
```

2068835340960

=>In General, Immutable Object content is Not Possible to copy(in the case of tuple). Where as in the case of frozenset, we are able to copy its content to another frozenset object. Here Original frozenset object and copied frozenset object contains Same Address and Not at all possible to Modify / Change their content.

=>frozenset does not contain the following Functions

- a) clear()

- b) add()
 - c) remove()
 - d) discard()
 - e) pop()
 - f) update()
-

V) Dict Category Data Types (Collection Data Types or Data Structures)

=>'dict' is one of the pre-defined class and treated as Dict Data Type.
 =>The purpose of dict data type is that "To store (Key,value) in single variable"
 =>In (Key,Value), the values of Key is Unique and Values of Value may or may not be unique.
 =>The (Key,value) must be organized or stored in the object of dict within Curly Braces {} and they separated by comma.
 =>An object of dict does not support Indexing and Slicing bcoz Values of Key itself considered as Indices.
 =>In the object of dict, Values of Key are treated as Immutable and Values of Value are treated as mutable.
 =>Originally an object of dict is mutable bcoz we can add (Key,Value) externally.
 =>We have two types of dict objects. They are

- a) Empty dict
- b) Non-empty dict

a) Empty dict

=>Empty dict is one, which does not contain any (Key,Value) and whose length is 0
 =>**Syntax:-** **dictobj1= { }**
 or
 dictobj=dict()

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

dictobj[Key1]=Val1
 dictobj[Key2]=Val2

dictobj[Key-n]=Val-n

Here Key1,Key2...Key-n are called Values of Key and They must be Unique
 Here Val1, Val2...Val-n are called Values of Value and They may or may not be unique.

b) Non-Empty dict:

=>Non-Empty dict is one, which contains (Key,Value) and whose length is >0
 =>**Syntax:-** **dictobj1= { Key1:Val1,Key2:Val2.....Key-n:Valn}**

Here Key1,Key2...Key-n are called Values of Key and They must Unique
 Here Val1, Val2...Val-n are called Values of Value and They may or may not be unique.

Examples:

```
>>> d1={10:"Python",20:"Data Sci",30:"Django"}
>>> print(d1,type(d1))----{10: 'Python', 20: 'Data Sci', 30: 'Django'} <class 'dict'>
>>> d2={10:3.4,20:4.5,30:5.6,40:3.4}
>>> print(d2,type(d2))----{10: 3.4, 20: 4.5, 30: 5.6, 40: 3.4} <class 'dict'>
>>> len(d1)-----3
>>> len(d2)-----4


---


>>> d3={}
>>> print(d3,type(d3))----{} <class 'dict'>
>>> len(d3)-----0
>>> d4=dict()
>>> print(d4,type(d4))----{} <class 'dict'>
>>> len(d4)-----0


---


>>> d2={10:3.4,20:4.5,30:5.6,40:3.4}
>>> print(d2)----{10: 3.4, 20: 4.5, 30: 5.6, 40: 3.4}
>>> d2[0]-----KeyError: 0
>>> d2[10]-----3.4
>>> d2[10]=10.44
>>> print(d2)----{10: 10.44, 20: 4.5, 30: 5.6, 40: 3.4}


---


>>> d2={10:3.4,20:4.5,30:5.6,40:3.4}
>>> print(d2,type(d2),id(d2))---{10: 3.4, 20: 4.5, 30: 5.6, 40: 3.4} <class 'dict'>
2090750380736
>>> d2[50]=5.5
>>> print(d2,type(d2),id(d2))---{10: 3.4, 20: 4.5, 30: 5.6, 40: 3.4, 50: 5.5} <class 'dict'>
2090750380736


---


>>> d3={}
>>> print(d3,type(d3),id(d3))----{} <class 'dict'> 2090750332992
>>> d3["Python"]=1
>>> d3["Java"]=3
>>> d3["C"]=2
>>> d3["GO"]=1
>>> print(d3,type(d3),id(d3))----{'Python': 1, 'Java': 3, 'C': 2, 'GO': 1} <class 'dict'>
2090750332992


---


>>> d4=dict()
>>> print(d4,type(d4),id(d4))----{} <class 'dict'> 2090754532032
>>> d4[10]="Apple"
>>> d4[20]="Mango"
>>> d4[30]="Kiwi"
```

```

>>> d4[40]="Sberry"
>>> d4[50]="Orange"
>>> print(d4,type(d4),id(d4))---{10: 'Apple', 20: 'Mango', 30: 'Kiwi', 40: 'Sberry', 50:
'Orange'}
2090754532032
>>> d4[10]="Guava"
>>> print(d4,type(d4),id(d4))----{10: 'Guava', 20: 'Mango', 30: 'Kiwi', 40: 'Sberry', 50:
'Orange'}
                                                <class 'dict'> 2090754532032
-----
```

```

>>> d2={10:3.4,20:4.5,30:5.6,40:3.4}
>>> print(d2,type(d2),id(d2))---{10: 3.4, 20: 4.5, 30: 5.6, 40: 3.4} <class 'dict'>
2090754531520
>>> d2[50]=1.2
>>> print(d2,type(d2),id(d2))---{10: 3.4, 20: 4.5, 30: 5.6, 40: 3.4, 50: 1.2} <class
'dict'>
                                                2090754531520
-----
```

#Dict in List

```

>>> l1=[10,{"Python":"RS","Numpy":"TV"},20,30]
>>> print(l1,type(l1))----[10, {'Python': 'RS', 'Numpy': 'TV'}, 20, 30] <class 'list'>
>>> print(l1[1],type(l1[1]))--{'Python': 'RS', 'Numpy': 'TV'} <class 'dict'>
>>> for k,v in l1[1].items():
...     print(k,"-->",v)
...
    Python --> RS
    Numpy --> TV
>>> l1[1]["Numpy"]="TRAVIS"
>>> print(l1,type(l1))--[10, {'Python': 'RS', 'Numpy': 'TRAVIS'}, 20, 30] <class 'list'>
-----
```

```

>>> t1=(10,{"Python":"RS","Numpy":"TV"},20,30)
>>> print(t1,type(t1))---(10, {'Python': 'RS', 'Numpy': 'TV'}, 20, 30) <class 'tuple'>
>>> print(t1[-3],type(t1[-3]))----{'Python': 'RS', 'Numpy': 'TV'} <class 'dict'>
>>> t1=(10,{"Python":"RS","Numpy":"TV"},20,30)
>>> print(t1,type(t1))---(10, {'Python': 'RS', 'Numpy': 'TV'}, 20, 30) <class 'tuple'>
>>> for k,v in t1[-3].items():
...     print(k,"-->",v)
...
    Python --> RS
    Numpy --> TV
-----
```

OR

```

>>> for k in t1[-3].keys():
...     print(k,"-->",t1[-3].get(k))
...
    Python --> RS
    Numpy --> TV
-----
```

#Dict in tuple

```
>>> t1=(10,{"Python":"RS","Numpy":"TV","Pandas":"MC"},20,30)
>>> print(t1,type(t1))
(10, {'Python': 'RS', 'Numpy': 'TV', 'Pandas': 'MC'}, 20, 30) <class 'tuple'>
>>> for k in t1[1]:
...     print(k,"-->",t1[1].get(k))
...
    Python --> RS
    Numpy --> TV
    Pandas --> MC
    OR
>>> for k in t1[1]:
...     print(k,"-->",t1[1][k])
...
    Python --> RS
    Numpy --> TV
    Pandas --> MC
```

#dict in set is NOT POSSIBLE, bcoz dict object is mutable

```
>>> s1={10,{"Python":"RS","Numpy":"TV","Pandas":"MC"},20,30,40}
--ERROR-----TypeError: unhashable type: 'dict'
```

Converting Tuple of List into dict

```
>>> l1=[(10,"Python"),(20,"Java"),(30,"ML"),(40,"DL")]
>>> print(l1,type(l1))[(10, 'Python'), (20, 'Java'), (30, 'ML'), (40, 'DL')] <class 'list'>
>>> for v in l1:
...     print(v,type(v))
...
    (10, 'Python') <class 'tuple'>
    (20, 'Java') <class 'tuple'>
    (30, 'ML') <class 'tuple'>
    (40, 'DL') <class 'tuple'>
>>> d1=dict(l1)
>>> print(d1,type(d1))
{10: 'Python', 20: 'Java', 30: 'ML', 40: 'DL'} <class 'dict'>
>>> for k,v in d1.items():
...     print(k,v)
...
    10 Python
    20 Java
    30 ML
    40 DL
>>> x=d1.items()
>>> print(x,type(x))
```

```

dict_items([(10, 'Python'), (20, 'Java'), (30, 'ML'),
(40, 'DL')]) <class 'dict_items'>
>>> l2=list(x)
>>> print(l2,type(l2))
[(10, 'Python'), (20, 'Java'), (30, 'ML'), (40, 'DL')] <class 'list'>
-----  

zip() with dict()  

-----
>>> l1=(10,20,30,40)
>>> l2=("Tushara","Aswin","Kalyan","Praveen")
>>> z=zip(l1,l2)
>>> print(z,type(z))---<zip object at 0x000002476C1DC7C0> <class 'zip'>
>>> for sno,sname in z:
...     print(sno,"-->",sname)
...
    10 --> Tushara
    20 --> Aswin
    30 --> Kalyan
    40 --> Praveen
-----  

>>> l1=(10,20,30,40)
>>> l2=("Tushara","Aswin")
>>> z=zip(l1,l2)
>>> print(z,type(z))---<zip object at 0x000002476C1DC9C0> <class 'zip'>
>>> for sno,sname in z:
...     print(sno,"-->",sname)
...
    10 --> Tushara
    20 --> Aswin
-----  

>>> l1=(10,20,30,40)
>>> l2=("Tushara","Aswin","Kalyan","Praveen")
>>> z=zip(l1,l2)
>>> d=dict(z)
>>> print(d,type(d))--{10: 'Tushara', 20: 'Aswin', 30: 'Kalyan', 40: 'Praveen'} <class
'dict'>
>>> for k,v in d.items():
...     print(k,"-->",v)
...
    10 --> Tushara
    20 --> Aswin
    30 --> Kalyan
    40 --> Praveen
=====
```

Pre-defined Functions in dict

=>On the object of dict, we can Varaious Operations by using Pre-defined Functions present in dict object. They are

1. clear()

=>Syntax: dictobj.clear()
=>This Function is used for removing all the elements of dict object.
=>If we call clear() upon empty dictobj then we None as Result.

Examples:

```
>>> d1={10:"Apple",20:"Mango",30:"Kiwi",40:"Guava"}  

>>> print(d1,type(d1))-----{10: 'Apple', 20: 'Mango', 30: 'Kiwi', 40: 'Guava'}  

<class 'dict'>  

>>> len(d1)-----4  

>>> d1.clear()  

>>> print(d1,type(d1))-----{} <class 'dict'>  

>>> len(d1)-----0  

>>> d1.clear()-----No Output  

      OR  

>>> print(d1.clear())-----None
```

2) copy()

Syntax: dictobj2=dictobj1.copy()

=>This Function is used for Copying the content of One dict object into another dict object.

Examples:

```
>>> d1={10:"Apple",20:"Mango",30:"Kiwi",40:"Guava"}  

>>> d2=d1.copy()  

>>> print(d1,id(d1))-----{10: 'Apple', 20: 'Mango', 30: 'Kiwi', 40: 'Guava'}  

2126281134400  

>>> print(d2,id(d2))-----{10: 'Apple', 20: 'Mango', 30: 'Kiwi', 40: 'Guava'}  

2126281134208  

>>> d1[50]="wmellon"  

>>> d2[45]="sberry"  

>>> print(d1,id(d1))-----{10: 'Apple', 20: 'Mango', 30: 'Kiwi', 40: 'Guava', 50:  

'wmellon'} 2126281134400  

>>> print(d2,id(d2))---{10: 'Apple', 20: 'Mango', 30: 'Kiwi', 40: 'Guava', 45: 'sberry'}  

2126281134208
```

3) pop()

Syntax: dictobj.pop(Key)

=>This Function is used for removing (Key,Value) from dictobj by passing value of Key.
=>If the value of key does not exist then we get KeyError.

Examples:

```

>>> d1={10:"Apple",20:"Mango",30:"Kiwi",40:"Guava"}
>>> print(d1,id(d1))-----{10: 'Apple', 20: 'Mango', 30: 'Kiwi', 40: 'Guava'}
2126281182144
>>> d1.pop(10)-----'Apple'
>>> print(d1,id(d1))-----{20: 'Mango', 30: 'Kiwi', 40: 'Guava'} 2126281182144
>>> d1.pop(30)-----'Kiwi'
>>> print(d1,id(d1))-----{20: 'Mango', 40: 'Guava'} 2126281182144
>>> d1.pop(40)-----'Guava'
>>> print(d1,id(d1))-----{20: 'Mango'} 2126281182144
>>> d1.pop(20)-----'Mango'
>>> print(d1,id(d1))-----{} 2126281182144
>>> d1.pop(40)-----KeyError: 40
>>> {}.pop(123)-----KeyError: 123
>>> dict().pop(12)----KeyError: 12

```

4) **popitem()**

Syntax: `dictobj.popitem()`

=>This Function is used for removing always last (key,Value) from non-empty dict object

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

Examples:

```

>>> d1={10:"Apple",20:"Mango",30:"Kiwi",40:"Guava"}
>>> print(d1,id(d1))-----{10: 'Apple', 20: 'Mango', 30: 'Kiwi', 40: 'Guava'}
2126285444032
>>> d1.popitem()-----(40, 'Guava')
>>> print(d1,id(d1))-----{10: 'Apple', 20: 'Mango', 30: 'Kiwi'} 2126285444032
>>> d1.popitem()-----(30, 'Kiwi')
>>> print(d1,id(d1))-----{10: 'Apple', 20: 'Mango'} 2126285444032
>>> d1.popitem()-----(20, 'Mango')
>>> print(d1,id(d1))-----{10: 'Apple'} 2126285444032
>>> d1.popitem()-----(10, 'Apple')
>>> print(d1,id(d1))-----{} 2126285444032
>>> d1.popitem()-----KeyError: 'popitem(): dictionary is empty'
>>> {}.popitem()----KeyError: 'popitem(): dictionary is empty'
>>> dict().popitem()----KeyError: 'popitem(): dictionary is empty'

```

5) **get()**

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

=>This Function is used for obtaining Value of Value by passing Value of Key

=>Here Varname is an object and holds / stores Value of value Based on Key

=>if the value of Key does not exist then we get None (But not KeyError)

(OR)

Syntax: `dictobj[Key]`

=>With this Syntax also, we get Value of Value by passing value of Key

=>if the value of Key does not exist then we get KeyError

Examples:

```
>>> d1={10:"Apple",20:"Mango",30:"Kiwi",40:"Guava"}
>>> print(d1,id(d1))-----{10: 'Apple', 20: 'Mango', 30: 'Kiwi', 40: 'Guava'}
2126285444480
>>> val=d1.get(10)
>>> print(val)-----Apple
>>> val=d1.get(40)
>>> print(val)-----Guava
>>> val=d1.get(100)
>>> print(val)-----None
>>> val=d1.get(20)
>>> print(val)-----Mango
(OR)
>>> d1={10:"Apple",20:"Mango",30:"Kiwi",40:"Guava"}
>>> print(d1,id(d1))-----{10: 'Apple', 20: 'Mango', 30: 'Kiwi', 40: 'Guava'}
2126285444032
>>> d1[10]-----'Apple'
>>> d1[20]-----'Mango'
>>> d1[30]-----'Kiwi'
>>> d1[300]-----KeyError: 300
```

6) keys()

Syntax: varname=dictobj.keys()

=>This Function is used for obtaining list of Values of Key in the object dict_keys
=>If we call keys() on empty dict object then we get empty object of dict_keys class

Examples:

```
>>> d1={10:"Apple",20:"Mango",30:"Kiwi",40:"Guava"}
>>> print(d1,id(d1))-----{10: 'Apple', 20: 'Mango', 30: 'Kiwi', 40: 'Guava'}
2126285444480
>>> ks=d1.keys()
>>> print(ks)-----dict_keys([10, 20, 30, 40])
>>> print(ks,type(ks))-----dict_keys([10, 20, 30, 40]) <class 'dict_keys'>
>>> d1=dict()
>>> print(d1,id(d1))-----{} 2126285444544
>>> ks=d1.keys()
>>> print(ks,type(ks))-----dict_keys([]) <class 'dict_keys'>
>>> {}.keys()-----dict_keys([])
>>> dict().keys()-----dict_keys([])
(OR)
>>> d1={10:"Apple",20:"Mango",30:"Kiwi",40:"Guava"}
>>> print(d1,id(d1))
{10: 'Apple', 20: 'Mango', 30: 'Kiwi', 40: 'Guava'} 2126285444480
>>> for k in d1.keys():
...     print(k)
...
...
```

```
10
20
30
40
```

5) values()

Syntax: varname=dictobj.valuea()

=>This Function is used for obtaining list of Values of Value in the object dict_values
=>If we call values() on empty dict object then we get empty object of dict_values class

Examples:

```
>>> d1={10:"Apple",20:"Mango",30:"Kiwi",40:"Guava"}
>>> print(d1,id(d1))-----{10: 'Apple', 20: 'Mango', 30: 'Kiwi', 40: 'Guava'}
2126285444032
>>> vs=d1.values()
>>> print(vs,type(vs))-----dict_values(['Apple', 'Mango', 'Kiwi', 'Guava']) <class
'dict_values'>
>>> d1={}
>>> vs=d1.values()
>>> print(vs,type(vs))-----dict_values([]) <class 'dict_values'>
>>> dict().values()-----dict_values([])
(OR)
>>> d1={10:"Apple",20:"Mango",30:"Kiwi",40:"Guava"}
>>> print(d1,id(d1))
{10: 'Apple', 20: 'Mango', 30: 'Kiwi', 40: 'Guava'} 2126285444608
>>> for v in d1.values():
...     print(v)
...
Apple
Mango
Kiwi
Guava
>>> for k in d1.keys():
...     print(k,"-->",d1.get(k))
...
10 --> Apple
20 --> Mango
30 --> Kiwi
40 --> Guava
>>> for k in d1.keys():
...     print(k,"-->",d1[k])
...
10 --> Apple
20 --> Mango
30 --> Kiwi
40 --> Guava
```

```
>>> for x in d1:
...     print(x)
...
10
20
30
40
>>> for x in d1:
...     print(x,"--->",d1[x])
...
10 ---> Apple
20 ---> Mango
30 ---> Kiwi
40 ---> Guava
```

8) items()

Syntax: varname=dictobj.items()

=>This Function is used for obtaining (Key,value) from non-empty dict object
=>If we call items() on empty dict object then we get empty object of dict_items class

=>Examples

```
>>> d1={10:"Apple",20:"Mango",30:"Kiwi",40:"Guava"}
>>> print(d1,id(d1))-----{10: 'Apple', 20: 'Mango', 30: 'Kiwi', 40: 'Guava'}
2126285444608
>>> its=d1.items()
>>> print(its,type(its))--dict_items([(10, 'Apple'), (20, 'Mango'), (30, 'Kiwi'), (40,
'Guava')]) <class
'dict_items'>
>>> d1={}
>>> print(d1,id(d1))-----{} 2126285444032
>>> its=d1.items()
>>> print(its,type(its))-----dict_items([]) <class 'dict_items'>
>>> dict().items()-----dict_items([])
```

9) update()

=>Syntax: dictobj1.update(dictobj2)

=>This Function is used for updating / adding the values of dictobj1 with dictobj2 values

=>Examples:

```
>>> d1={10:"Apple",20:"Mango"}
>>> d2={30:"Kiwi",40:"Guava"}
>>> print(d1,type(d1))-----{10: 'Apple', 20: 'Mango'} <class 'dict'>
>>> print(d2,type(d2))-----{30: 'Kiwi', 40: 'Guava'} <class 'dict'>
```

```

>>> d1.update(d2)
>>> print(d1,type(d1))-----{10: 'Apple', 20: 'Mango', 30: 'Kiwi', 40: 'Guava'} <class
'dict'>
>>> print(d2,type(d2))-----{30: 'Kiwi', 40: 'Guava'} <class 'dict'>
-----
>>> d1={10:"Apple",20:"Mango"}
>>> d2={30:"Kiwi",10:"Guava"}
>>> print(d1,type(d1))-----{10: 'Apple', 20: 'Mango'} <class 'dict'>
>>> print(d2,type(d2))-----{30: 'Kiwi', 10: 'Guava'} <class 'dict'>
>>> d1.update(d2)
>>> print(d1,type(d1))-----{10: 'Guava', 20: 'Mango', 30: 'Kiwi'} <class 'dict'>
>>> print(d2,type(d2))-----{30: 'Kiwi', 10: 'Guava'} <class 'dict'>
-----
>>> d1={10:"Apple",20:"Mango"}
>>> d2={10:"Kiwi",20:"Guava"}
>>> print(d1,type(d1))-----{10: 'Apple', 20: 'Mango'} <class 'dict'>
>>> print(d2,type(d2))-----{10: 'Kiwi', 20: 'Guava'} <class 'dict'>
>>> d1.update(d2)
>>> print(d1,type(d1))-----{10: 'Kiwi', 20: 'Guava'} <class 'dict'>
>>> print(d2,type(d2))-----{10: 'Kiwi', 20: 'Guava'} <class 'dict'>

```

Special Points:

=>In dict object, one define another dict object

Examples:

```

>>>
d1={"TS":{"HYD":"ITHUB", "SEC":"BUSHUB"}, "AP":{"VIJ":"EDU", "VIZ":"PORT"}, "BAN
G":"ITHUB"}
>>> print(d1,type(d1))
{'TS': {'HYD': 'ITHUB', 'SEC': 'BUSHUB'}, 'AP': {'VIJ': 'EDU', 'VIZ': 'PORT'}, 'BANG':
'ITHUB'} <class 'dict'>
>>> print(d1["TS"],type(d1["TS"]))-----{'HYD': 'ITHUB', 'SEC': 'BUSHUB'} <class
'dict'>
>>> print(d1["AP"],type(d1["AP"]))-----{'VIJ': 'EDU', 'VIZ': 'PORT'} <class 'dict'>
>>> print(d1["BANG"],type(d1["BANG"]))---ITHUB <class 'str'>
>>> for k,v in d1.items():
...     print(k,"<->",v)
...
    TS <-> {'HYD': 'ITHUB', 'SEC': 'BUSHUB'}
    AP <-> {'VIJ': 'EDU', 'VIZ': 'PORT'}
    BANG <-> ITHUB

```

=>In dict object, one can defined List,tuple and set objects

Examples:

```

>>> d1={10:["Python","Data Science","Django"],20:(("java","adv
java"),30:{"HTML","CSS","JS","XML"} }

```

```

>>> print(d1,type(d1))
                {10: ['Python', 'Data Science', 'Django'], 20: ('java', 'adv
java'), 30: {'JS', 'HTML', 'XML', 'CSS'}} <class 'dict'>
>>> print(d1[10],type(d1[10]))---['Python', 'Data Science', 'Django'] <class 'list'>
>>> print(d1[20],type(d1[20]))---('java', 'adv java') <class 'tuple'>
>>> print(d1[30],type(d1[30]))---{'JS', 'HTML', 'XML', 'CSS'} <class 'set'>
>>> for k in d1.keys():
...     print(k,"-->",d1.get(k))
...
10 --> ['Python', 'Data Science', 'Django']
20 --> ('java', 'adv java')
30 --> {'JS', 'HTML', 'XML', 'CSS'}
>>> for k in d1:
...     print(k,"-->",d1[k])
...
10 --> ['Python', 'Data Science', 'Django']
20 --> ('java', 'adv java')
30 --> {'JS', 'HTML', 'XML', 'CSS'}
>>> for its in d1.items():
...     print(its)
...
(10, ['Python', 'Data Science', 'Django'])
(20, ('java', 'adv java'))
(30, {'JS', 'HTML', 'XML', 'CSS'})
>>> for k,v in d1.items():
...     print(k,"-->",v)
...
10 --> ['Python', 'Data Science', 'Django']
20 --> ('java', 'adv java')
30 --> {'JS', 'HTML', 'XML', 'CSS'}

```

=>In list object, One can define, dict, list,tuple and set and other values also

Examples:

```

>>>
l1=[10,"Rossum", {"cm":16,"c++":17,"pyt":15},("B.Tech","CSE"),{1,2,3},[1.2,3.4,5.6] ]
>>> for v in l1:
...     print(v,"-->",type(v))
...
10 --> <class 'int'>
Rossum --> <class 'str'>
{'cm': 16, 'c++': 17, 'pyt': 15} --> <class 'dict'>
('B.Tech', 'CSE') --> <class 'tuple'>
{1, 2, 3} --> <class 'set'>
[1.2, 3.4, 5.6] --> <class 'list'>

```

=>In tuple object, One can define, dict, list,tuple and set and other values also

Examples:

```
>>>
t1=(10,"Rossum",{"cm":16,"c++":17,"pyt":15},("B.Tech","CSE"),{1,2,3},{1.2,3.4,5.6} )
>>> for v in t1:
...     print(v,"-->",type(v))
...
    10 --> <class 'int'>
    Rossum --> <class 'str'>
    {'cm': 16, 'c++': 17, 'pyt': 15} --> <class 'dict'>
    ('B.Tech', 'CSE') --> <class 'tuple'>
    {1, 2, 3} --> <class 'set'>
    [1.2, 3.4, 5.6] --> <class 'list'>
```

=>In set object, One can't define, dict, list and set and other values also (except tuple)

Examples:

```
>>>
s1={10,"Rossum", {"cm":16,"c++":17,"pyt":15},("B.Tech","CSE"),{1,2,3},{1.2,3.4,5.6} }
                                                Traceback (most recent call last):
                                                File "<stdin>", line 1, in <module>
                                                TypeError: unhashable type: 'dict'
```

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 to develop Programs in Python

Program

=>A Program is a Collection OR Set of Optimized Instructions
 =>Optimized Instructions are nothing taking Less Execution Time and Less Memory Spaces
 =>The purpose of Writing Programs is that "To Solve the real Time Problems".
 =>In Otherwords, Real Time Problem Solving Approaches always done by writing Programs.
 =>In Python Programming, We write Block of Statements and we save on some file name with an extension .py (FileName.py)

=>In Python Language, we have 2 approaches for developing Programs. They are

1. Interactive Approach
 2. Batch Mode Approach
-

1. Interactive Approach

=>In This mode of development, Programmer can Issue One Statement at a time and gets Result of that Statement at a time.
 =>This Mode of Development is Useful to test one instructions at a time But not useful to develop group of instructions for solving Problems in real Time bcoz This mode is not allowing to save the Instructions on filename.
 =>To avoid this Problem and solve Real Time Problem Solving, we use Batch Mode Approach.

Examples Softwares: Python Command Prompt Python IDLE Shell

2. Batch Mode Approach

=>This Mode of Development is useful for developing Real Time Problems Solving by writing Group of Instructions.
 =>In this mode development we write of Group of Instructions and we save on some file name with an extnsion .py (FileName.py)

Examples:

- 1) Python IDLE Shell(Installation of Python Software)
- 2) Third Party IDES (Install Explicitly)

- Pycharm
- Jupiter Note Book
- VS Code
- Spider
- Google Clab
- Sublime text
- Edit Plus...etc (best for learning)

#This program adds Two Values

```

a=10
b=20
c=a+b
print(a)
print(b)
print(c)
-----
#Program for adding of two values
a=10
b=20
c=a+b
print("====")
print("Val of a=",a)
print("Val of b=",b)
print("sum=",c)
print("====")
-----
#program adding two values
a=float(input("Enter Value of a:"))
b=float(input("Enter Value of b:"))
c=a+b
print("*****")
print("Val of a=",a)
print("Val of b=",b)
print("sum=",c)
print("*****")
-----
print("Wel come to python classes")
print("Batch Mode Programming")

```

=====

Copy Techniques in Python

=====

=>In Python Programming, we have Two 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) The Memory Address of Both the Objects are DIFFERENT**

c) The Modifications are Independent (Whatever the changes we do on one object, Which are not reflected to another object)

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

Syntax: object2=object1.copy()

Examples:

```
>>> l1=[10,"Rossum",23.45]
>>> print(l1,id(l1))-----[10, 'Rossum', 23.45] 1996956062656
>>> l2=l1.copy() # Shallow Copy
>>> print(l2,id(l2))-----[10, 'Rossum', 23.45] 1996959406400
>>> l1.append("Python")
>>> l2.insert(1,"Nether")
>>> print(l1,id(l1))-----[10, 'Rossum', 23.45, 'Python'] 1996956062656
>>> print(l2,id(l2))-----[10, 'Nether', 'Rossum', 23.45] 1996959406400
```

2. Deep Copy

=>The Properties of Deep Copy are

- a) Initial Content of Both the Objects are SAME
- b) The Memory Address of Both the Objects are SAME**
- c) The Modifications are Dependent (Whatever the changes we do on one object, Which are reflected to another object and both of them are pointing same memory space)

=>To Implement the Deep Copy Technique, we use Assignment Operator

Syntax: **object2=object1**

Examples:

```
>>> l1=[10,"Rossum",23.45]
>>> print(l1,id(l1))-----[10, 'Rossum', 23.45] 1996956038208
>>> l2=l1 # Deep Copy
>>> print(l2,id(l2))-----[10, 'Rossum', 23.45] 1996956038208
>>> l1.append("Python")
>>> print(l1,id(l1))-----[10, 'Rossum', 23.45, 'Python'] 1996956038208
>>> print(l2,id(l2))-----[10, 'Rossum', 23.45, 'Python'] 1996956038208
>>> l2.insert(1,"Nether")
>>> print(l1,id(l1))-----[10, 'Nether', 'Rossum', 23.45, 'Python'] 1996956038208
>>> print(l2,id(l2))-----[10, 'Nether', 'Rossum', 23.45, 'Python'] 1996956038208
```

Display the result of python program on Console (OR) print()

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

=>In otherwords, print() is a pre-defined function, which is used for displaying the result of python Program on the console.

=>print() can be used in the following ways with different Syntaxes.

Syntax-1 : **print(Var1,Var2,.....Var-n)**

(OR)

print(Var)

(OR)

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

(OR)

print(Val)

=>This Syntax displays single value on the console.

Examples:

```
>>> a=10
>>> print(a)-----10
>>> a=10
>>> b="KVR"
>>> print(a,b)-----10 KVR
>>> print(100)-----100
>>> print(100,"Rossum")-----100 Rossum
```

Syntax-2: print(Message)

 (OR)
 print(Msg1,Msg2.....,Msg-n)
 (OR)
 print(Msg1+Msg2+,...+Msg-n)

=>Here Message is one of the Str Data.

=>This Syntax displays the message(s) on the console

=>Here we used + operator for contacatation of Str data (Messages) but not
 Possible to concatenate with
 Numerical values.

Examples:

```
>>> print("Hello Python World")-----Hello Python World
>>> print("Hello","Python","world")-----Hello Python world
>>> print("Hello"+ "Python" + "World")-----HelloPythonWorld
>>> print("Hello" + " Python" + " World")-----Hello Python World
>>> print("Hello" + " " + "Python" + " " + "World")-----Hello Python World
>>> print("Hello"+10)-----TypeError: can only concatenate str (not "int") to str
>>> print("Hello"+ "10")-----Hello10
>>> print("Hello"+str(10))-----Hello10
```

Syntax-3: print(Messages Cum Values)

 (OR)
 print(Values cum Messages)

=>This Syntax displays Messages cum Values OR Values cum Messages

Examples:

```
>>> a=10
>>> print("Val of a=",a)-----Val 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: print(Message Cum Values with format()))

(OR)

```
print(Values cum Messages with format())
```

(OR)

```
print("{} ,{} ,... ,{} ".format(var1,var2....var-n))
```

=>here format() is used for supplying the Specified Variable Values to empty {} braces.

Examples:

```
>>> a=100  
>>> print("Val of a={} ".format(a))-----Val of a=100  
>>> print("{} is the val of a".format(a))-----100 is the val of a
```

```
>>> a=100
>>> b=200
>>> c=a+b
>>> print("Sum={}".format(c))-----Sum=300
>>> print("{} is the sum".format(c))---300 is the sum
>>> print("Sum of {} and {}={}".format(a,b,c))----Sum of 100 and 200=300
```

```
>>> sno=10
>>> sname="Rossum"
>>> marks=34.56
>>> print("Roll Number {} Name:{} and Marks={}".format(sno,sname,marks))
                    Roll Number 10
```

Syntax-5 : print(Message Cum Values with format Specifiers)

OR

print(Values cum Messages with format Specifiers)
(OR)

=>Here Format specifiers represents
%d is used for int data,
%f is used for float data
%s is used for str data

=>if any value does not contain any format specifier then we must convert into str type by using str(Value) and use %s

Examples:

```
>>> a=100
>>> print("Val of =%d" %a)-----Val of =100
>>> print("%d is the val of a" %a)----100 is the val of a
```

```
>>> a=10
>>> b=20
>>> c=a+b
>>> print("sum of %d and %d=%d" %(a,b,c))---sum of 10 and 20=30
>>>>> a=100
>>> print("Val of =%d" %a)----Val of =100
>>> print("%d is the val of a" %a)----100 is the val of a
```

```
>>> a=10
>>> b=20
>>> c=a+b
>>> print("sum of %d and %d=%d" %(a,b,c))---sum of 10 and 20=30
```

```
>>> a=10
>>> b=2.3
>>> c="Python"
>>> print("Val of a=%d Value of b=%f and val of c=%s" %(a,b,c))
                           Val of a=10 Value of b=2.300000 and val of
c=Python
>>> print("Val of a=%d Value of b=%0.2f and val of c=%s" %(a,b,c))
                           Val of a=10 Value of b=2.30 and val of
c=Python
>>> print("Val of a=%d Value of b=%0.3f and val of c=%s" %(a,b,c))
                           Val of a=10 Value of b=2.300 and val of
c=Python
```

```
>>> a=23
>>> b=34
>>> c=a+b
>>> print("Sum of %d and %d=%d" %(a,b,c))---Sum of 23 and 34=57
>>> print("Sum of %f and %f=%f" %(a,b,c))---Sum of 23.000000 and
34.000000=57.000000
>>> print("Sum of %0.2f and %0.2f=%0.3f" %(a,b,c))--Sum of 23.00 and
34.00=57.00
```

```
>>> a=1.2
>>> b=4.6
>>> c=a+b
>>> print("Sum of %f and %f=%f" %(a,b,c))---Sum of 1.200000 and
4.600000=5.800000
>>> print("Sum of %0.2f and %0.2f=%0.3f" %(a,b,c))---Sum of 1.20 and 4.60=5.800
```

```

>>> print("Sum of %d and %d=%d" %(a,b,c))---Sum of 1 and 4=5
-----
>>> a=10
>>> b=2.4
>>> print("sum({},{}]={} ".format(a,b,a+b))-----sum(10,2.4)=12.4
>>> print("sum(%d,%f)=%f" %(a,b,a+b))-----sum(10,2.400000)=12.400000
>>> print("sum(%d,%0.2f)=%0.2f" %(a,b,a+b))-----sum(10,2.40)=12.40
-----
>>> lst=[10,"Rossum",True,2+3j,(10,20,30)]
>>> print(lst,type(lst))-----[10, 'Rossum', True, (2+3j), (10, 20, 30)] <class 'list'>
>>> print("content of list=",lst)-----content of list= [10, 'Rossum', True, (2+3j), (10,
20, 30)]
>>> print("content of list={}".format(lst))---content of list=[10, 'Rossum', True, (2+3j),
(10, 20, 30)]
-----
>>> print("content of list=%d" %lst)---TypeError: %d format: a real number is
required, not list
>>> print("content of list=%s" %str(lst))---content of list=[10, 'Rossum', True, (2+3j),
(10, 20, 30)]
-----
>>> d1={10:"Apple",20:"mango",30:"Kiwi"}
>>> print("content of d1=",d1)---content of d1= {10: 'Apple', 20: 'mango', 30: 'Kiwi'}
>>> print("content of d1={}".format(d1))--content of d1={10: 'Apple', 20: 'mango', 30:
'Kiwi'}
>>> print("content of d1=%d" %d1)---TypeError: %d format: a real number is
required, not dict
>>> print("content of d1=%s" %str(d1))---content of d1={10: 'Apple', 20: 'mango', 30:
'Kiwi'}
>>> for k,v in d1.items():
...     print("%d-->%s" %(k,v))
...
...                                         10-->Apple
...                                         20-->mango
...                                         30-->Kiwi
>>> for k,v in d1.items():
...     print("{}-->{}".format(k,v))
...
...                                         10-->Apple
...                                         20-->mango
...                                         30-->Kiwi

```

Syntax-6 : print(Values, end=" ")

This Syntax display the values in same line

Examples:

```

>>> for val in r:
...     print(val,end=" ")----0 1 2 3 4 5

```

```
>>> lst=[10,"Rossum",True,2+3j,(10,20,30)]
>>> for val in lst:
...     print(val,end="-->")-----10-->Rossum-->True-->(2+3j)-->(10, 20, 30)
```

Special Points

```
>>> s="python"
>>> print(s)-----python
>>> s=s+s+s
>>> print(s)-----pythonpythonpython
```

```
>>> s="python"
>>> print(s*3)-----pythonpythonpython---repetition Operator (*)
>>> s="python"
>>> print(3*s)-----pythonpythonpython
>>> a=10
>>> b=2
>>> print(a*b)-----20--Mul Operator (*)
>>> print("Hello")----Hello
>>> print("Hello"*4)-----HelloHelloHelloHello
```

```
>>> lst=[10,"KVR"]
>>> print(lst*3)-----[10, 'KVR', 10, 'KVR', 10, 'KVR']
```

```
>>> print("=")----- =
>>> print("=====")----- =====
>>> print("=*10)-=====*
>>> print("#"*20)--- ##########
>>> print("*"*40)--- *****
```

```
>>> #zip()
>>> lst1=[10,20,30,40]
>>> lst2=["RS","TR","MC","DR"]
>>> z=zip(lst1,lst2)
>>> print(z,type(z))-----<zip object at 0x00000254A34A17C0> <class 'zip'>
>>> for rno,name in z:
...     print("{}-->{}".format(rno,name))
...
```

```
10-->RS
20-->TR
30-->MC
40-->DR
```

```
>>> for rno,name,cls in zip(lst1,lst2,['X','XII','MCA','B.Tech']):
...     print("{}-->{}-->{}".format(rno,name,cls))
...
```

```
10-->RS-->X
20-->TR-->XII
30-->MC-->MCA
40-->DR-->B.Tech
```

```

>>> lst1=[10,20,30,40]
>>> lst2=["RS","TR"]
>>> for rno,name,cls in zip(lst1,lst2,['X','XII','MCA']):
...     print("{}-->{}-->{}".format(rno,name,cls))
...
    10-->RS-->X
    20-->TR-->XII

-----
```

```

>>> lst1=[10,20,30,40]
>>> lst2=["RS","TR","MC","DR"]
>>> z=zip(lst1,lst2)
>>> print(z,type(z))-----<zip object at 0x00000254A34A18C0> <class 'zip'>
>>> d=dict(z)
>>> print(d,type(d))-----{10: 'RS', 20: 'TR', 30: 'MC', 40: 'DR'} <class 'dict'>
>>> for k,v in d.items():
...     print("{}-->{}".format(k,v))
...
    10-->RS
    20-->TR
    30-->MC
    40-->DR

-----
```

```

>>> for k in d:
...     print("{}-->{}".format(k,d[k]))
...
    10-->RS
    20-->TR
    30-->MC
    40-->DR

-----
```

```

>>> for k in d:
...     print("{}-->{}".format(k,d.get(k)))
...
    10-->RS
    20-->TR
    30-->MC
    40-->DR

```

Reading the Data Dynamically from Key Board

=>To Read the Data Dynamically from Key Board , we have Two Pre-defined Functions. They are

- 1) input()
- 2) input(Message)

1) input()

Syntax: varname=input()

=>The purpose of input() is that To read the data dynamically from Key Board.

=>input() can read any type of data from key board in the form of str and placed in LHS Var i,e varname.

Examples: Refer DataReadEx1.py, DataReadEx2.py, DataReadEx3.py, DataReadEx4.py

2) input(Message)

Syntax: varname=input("Message")

=>Here Message Represents user-prompting message

=>input("Message") also reads any type of value from key board in the form of str and placed in LHS Var i,e varname and additionally it gives User-Prompting Message.

Examples:

Refer DataReadEx5.py, DataReadEx6.py, DataReadEx7.py

#This Program read any values and display

```
#DataReadEx1.py
print("Enter val of a:")
a=input()
print("Val of a={} and its type={}".format(a,type(a)))
x=float(a) # Type cating from str type to float type
print("Val of x={} and its type={}".format(x,type(x)))
```

#This Program reads two values and find their product

```
#DataReadEx2.py
print("Enter First Value:")
a=input()
print("Enter Second Value:")
b=input()
#Type cating
x1=float(a)
x2=float(b)
x3=x1*x2
print("*50")
print("\tFirst Value:{}\n".format(x1))
print("\tSecond Value:{}\n".format(x2))
print("\tMul({},{})={}\n".format(x1,x2,x3))
print("*50")
```

#This Program reads two values and find their product

```
#DataReadEx3.py
print("Enter Two Values:")
a=input()
b=input()
#Type cating
x1=float(a)
x2=float(b)
```

```
x3=x1*x2
print("*50)
print("\tFirst Value:{}".format(x1))
print("\tSecond Value:{}".format(x2))
print("\tMul({},{}]={})".format(x1,x2,x3))
print("*50)
```

```
#This Program reads two values and find their product
#DataReadEx4.py
print("Enter Two Values:")
x1=float( input() )
x2=float(input())
x3=x1*x2
print("*50)
print("\tFirst Value:{}".format(x1))
print("\tSecond Value:{}".format(x2))
print("\tMul({},{}]={})".format(x1,x2,x3))
print("*50)
```

```
#This Program reads two values and find their product
#DataReadEx5.py
a=input("Enter First Value:")
b=input("Enter Second Value:")
x1=float(a)
x2=float(b)
print("Mul({},{}]={})".format(x1,x2,x1*x2))
```

```
#This Program reads two values and find their product
#DataReadEx6.py
a=float(input("Enter First Value:"))
b=float(input("Enter Second Value:"))
print("Mul({},{}]={})".format(a,b,a*b))
```

```
#This Program reads two values and find their product
#DataReadEx7.py
print("mul={}".format(float(input("Enter First Value:"))*float(input("Enter Second
Value:"))))
```

```
#CircleAreaPeri.py
r=float(input("Enter Radius:"))
ac=3.14*r*r
pc=2*3.14*r
print("*50)
print("\t\tRadius={}".format(r))
print("\t\tArea of Circle={}".format(ac))
print("\t\tPerimeter if Circle={}".format(pc))
print("*50)
```

```
#SimpleInt.py
#accepting input
```

```

p=float(input("Enter Principle Amount:"))
t=float(input("Enter Time:"))
r=float(input("Enter Rate of interest:"))
#cal si and totamt
si=(p*t*r)/100
totamt=p+si
#display the result
print("****50")
print("Principle Amount={}".format(p))
print("Time={}".format(t))
print("Rate of interest={}".format(r))
print("-"*50)
print("Simple Interest={}".format(si))
print("TOTAL AMOUNT TO PAY={}".format(totamt))
print("****50")
=====
```

Operators and Expressions in Python

=>An Operator is a symbol, which is used to perform Certain Operation on data and gives Result.

=>If any Operator connected with Two or More Objects / variables then it is called Expression.

=>In Other An Expression is a collection of Objects / Variables Connected with Operator.

=>In Python Propgramming, 7 types of Operators. They are

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

NOTE1: Short hand Operators---we discuss in Python

NOTE2: Python does not contain Pre-Increment / Decrement (++, --) and Post Increment / Decrement

NOTE3: Python does not Contain C,C++,java, C#.net Ternary Operator---- ? :

NOTE4: Python Programming its own Ternary Operator ----- if...else Operator

1. Arithmetic Operators

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

=>If two or more Variables / objects connected with Arithmetic Operators then we call it as Arithmetic Expression.

=>Python programming contains 7 type of Arithmetic Operators. They are given in the following table.

| SLNO | SYMBOL | MEANING | EXAMPLES: |
|-------|--------|-----------------|---|
| <hr/> | | | |
| a=10 | b=3 | | |
| 1 | + | Addition | print(a+b)----13 |
| 2 | - | Substraction | print(a-b)----7 |
| 3 | * | Multiplication | print(a*b)----30 |
| 4 | / | Division | print(a/b)>3.333333333335 (Float Quotient) |
| 5. | // | Floor Division | print(a//b)---->3 (int Quotient) |
| 6. | % | Modulo Division | print(a%b)---->1 |
| 7. | ** | Exponentiation | print(a**b)---1000 (power) |

2. Assignment 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 Assignment Operator is single equal to (=).

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

1. Single Line Assignment
 2. Multi Line Assignment
-

1. Single Line Assignment:

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

=>With Single Line Assignment 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 Assignment:

=>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
>>> sno,sname,marks=10,"Rossum",34.56
>>> print(sno,sname,marks)-----10 Rossum 34.56
```

```
#AopEx1.py
a=int(input("Enter Value of a:"))
b=int(input("Enter value of b:"))
print("*50")
print("Arithmic Operators Results")
print("*50)
print("\tSum({},{})={}".format(a,b,a+b))
print("\tSub({},{})={}".format(a,b,a-b))
print("\tMul({},{})={}".format(a,b,a*b))
print("\tNormal Div({},{})={}".format(a,b,a/b))
print("\tFloor Div({},{})={}".format(a,b,a//b))
print("\tMod({},{})={}".format(a,b,a%b))
print("\tPow({},{})={}".format(a,b,a**b))
print("*50)
```

```
#AopEx2.py
n=int(input("Enter a number for finding its square root:"))
res=n**0.5
print("Sqrt({})={}".format(n,res))
print("=====OR-format specifier=====")
print("Sqrt(%d)=%0.2f" %(n,res))
print("=====OR-round()=====")
print("Sqrt({})={}".format(n,round(res,2)))
```

```
#program for accepting any two values and swap them
#SwapEx1.py
a=input("Enter Value of a:")
b=input("Enter Value of b:")
print("*50)
print("Original Value of a={}".format(a))
```

```

print("Original Value of b={}".format(b))
print("*50)
#swaping logic
b,a,b # multi line assigment
print("Swapped Value of a={}".format(a))
print("Swapped Value of b={}".format(b))
print("*50)

-----
#program for accepting any two values and swap them
#SwapEx2.py
a,b=input("Enter Value of a:"),input("Enter Value of b:")
print("*50)
print("Original Value of a={}".format(a))
print("Original Value of b={}".format(b))
print("*50)
#swaping logic
t=a
a=b
b=t
print("Swapped Value of a={}".format(a))
print("Swapped Value of b={}".format(b))
print("*50)
-----
```

3. Relational Operators (Comparision Operators)

- =>The purpose of Relational Operator is that " To compare two values".
- =>if two or More variables / Object connected with relational operators then we it as Relational Expression.
- =>The result of Relational Expressions is always either True or False .
- =>The Relational Expression is called Test Condition.
- =>In Python Programming, we have 6 types of Relational Operators. They are given in the folowing table.

| SLNO | SYMBOL | MEANING | EXAMPLES |
|------|--------|-------------------------------|---------------------------------------|
| 1. | > | greater than | 10>20----->False 10>5----->True |
| 2. | < | less than | 10<20----->True 10<2----->False |
| 3. | == | Equality (Double Equal to) | 10==20----->False 10==10----->True |
| 4. | != | Not Equal to | 10!=20----->True |

| | | | |
|----|----|--------------------------|--------------------------------------|
| | | | 10!=10----->False |
| 5. | >= | greater than or equal to | 10>=20----->False 10>=5----->True |
| 6. | <= | less than or equal to | 10<=20----->True 10<=5----->False |

===== 4. Logical Operators (Comparision Operators) =====

=>The purpose of Logical Operators is that "To combine Two or More Relational Expressions.".

=>If two or More Relational Expressions connected with Logical Operators then we call it as Logical Expression.

=>Logical Expression also called Compound Conditions and whose result can be either True or False.

=>In Python Programming, we have 3 Types of Logical operators. They are given in the following Table.

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

1) 'and' Operator

=>The Functionality of 'and' operator can be expressed by using the following Truth table

| | RelExpr1 | RelExpr2 | RelExpr1 and RelExpr2 |
|--|----------|----------|-----------------------|
| | False | False | False |
| | True | False | False |
| | False | True | False |
| | True | True | True |

Examples:

```
>>> True and False-----False
>>> False and False-----False
>>> True and True-----True
>>> False and True-----False
```

Examples:

```
>>> print(10>20 and 30>20)-----False
>>> print(10>20 and 40>20 and 30>15)-----False
>>> print(10>2 and 20>40 and 40>20)-----False
>>> print(10>2 and 100>20 and 400>40)-----True
```

Short Circuit Evaluation in the case 'and' Operator

=>if the Logical Expression (Combination of Rel Expressions) contains Multiple Relational Expressions and if First Relational Expression is False then PVM will not evaluate Rest of the Relational Expressions which are present in Logical Expression and then the result of entire Logical Expression is False. This Process of Evaluation is called Short Circuit Evaluation

2) 'or' Operator

=>The Functionality of 'or' operator can be expressed by using the following Truth table

| RelExpr1 | RelExpr2 | RelExpr1 or RelExpr2 |
|----------|----------|----------------------|
| False | False | False |
| True | False | True |
| False | True | True |
| True | True | True |

Examples:

```
>>> False or False-----False
>>> True or False-----True
>>> False or True-----True
>>> True or True-----True
```

Examples:

```
>>> 10>2 or 20>3-----True
>>> 10>20 or 40>20 or 50>60---True
>>> 10>20 or 23>45 or 56>78-----False
>>> 10>2 or 45>67 or 56>78 or 67>89---True
```

Short Circuit Evaluation in the case of 'or' Operator

=>if the Logical Expression (Combination of Rel Expressions) contains Multiple Relational Expressions and if First Relational Expression is True then PVM will not evaluate Rest of the Relational Expressions which are present in Logical Expression

and then the result of entire Logical Expression is True. This Process of Evaluation is called Short Circuit Evaluation

3) 'not' Operator

=>The Functionality of 'not' operator can be expressed by using the following Truth table

| RelExpr1 | not RelExpr1 |
|----------|--------------|
| False | True |
| True | False |

Examples:

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

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

```
>>> 10>2 and 20>4-----True
>>> not (10>2 and 20>4)-----False
```

```
>>> 10>2 or 40>60-----True
>>> not (10>2 or 40>60)-----False
```

Special Points

```
>>> not True-----False
>>> not False-----True
```

```
>>> not 255-----False
>>> not -132-----False
>>> not 120-120-----True
>>> not "PYTHON"-----False
>>> not not "python"-----True
```

```
>>> 100 and 200-----200
>>> 100 and 200 and 400-----400
>>> 100 and 0-----0
>>> -100 and -234 and -456-----456
>>> 123 and 0 and 456-----0
>> "KVR" and "PYTHON" and "False" -----'False'
>>> "KVR" and "PYTHON" and True-----True
>>> " " and " " and " " -----"
```

```
>>> "KVR" or "Python" or "HYD"-----'KVR'
>>> 0 or 0 or 23-----23
```

```
>>> 100 or "Python" or "True"-----100
>>> 100 or 200 and 400-----100
>>> 100 and 200 or 400 and 123 or 456-----200
>>> True and True or False and True or 0 and "Python"-----True
```

```
#RelOpEx1.py
a=int(input("Enter Value of a:"))
b=int(input("Enter Value of b:"))
print("*50)
print("Result of relational Operators")
print("*50)
print("\t{} > {}-->{}".format(a,b,a>b))
print("\t{} < {}-->{}".format(a,b,a<b))
print("\t{} == {}-->{}".format(a,b,a==b))
print("\t{} != {}-->{}".format(a,b,a!=b))
print("\t{} >= {}-->{}".format(a,b,a>=b))
print("\t{} <= {}-->{}".format(a,b,a<=b))
print("*50)
```

===== 5. Bitwise Operators (Most Imp) =====

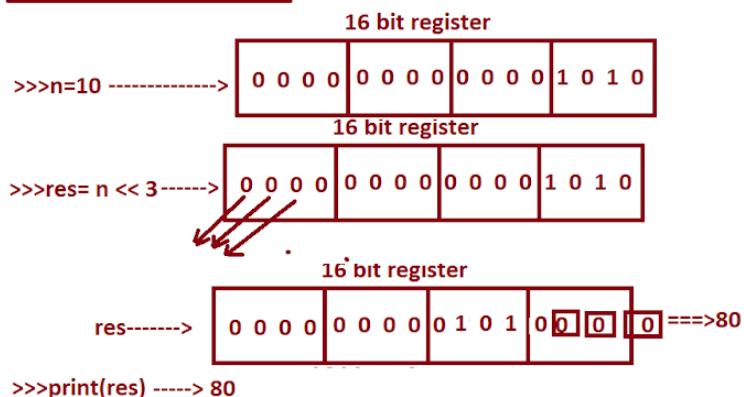
=>The purpose of Bitwise Operators is that "To Perform the operations on the data in the form of Bits".

=>Bitwise Operators are applicable on Integer data only but not applicable to float type values bcoz float can't maintain certainty.

=>All Types of Bitwise Operators Converts given Integer data into Binary Format and starts operating in the form of Bit by Bit and finally gives the result in Decimal Number System format and hence these operators are named as Bitwise Operators.

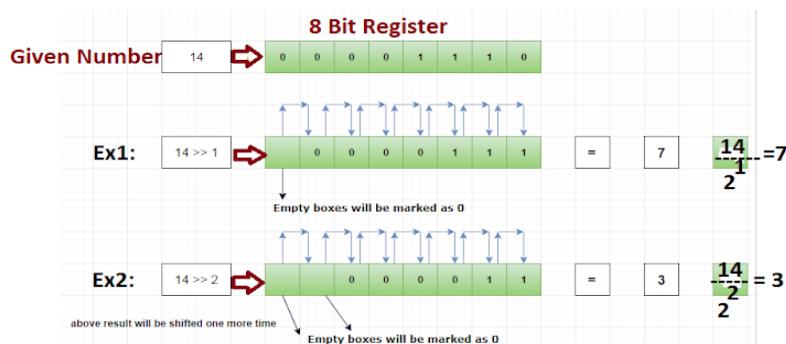
=>In Python Programming, we have 6 types of Bitwise Operators. They are

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

1) Left shift operator (<<)

Formula:- res = Number<<No. of Bits
 No. of Bits
 res = Number x 2

Examples: res=10<<3 -----> 10 x 2³
 10 x 8=80

2) Right Shift Operator(>>) : syntax: res=number >> No.of Bits**1. Bitwise Left Shift Operator (<<)**

Syntax: varname=GivenNumber << No. of Bits

Concept: The Bitwise Left Shift Operator (<<) shifts No. of Bits Towards Left Side By adding No. of Zero at Right Side (depends on No. of Bits). So that result value will displayed in the form of decimal number system.

Examples:

```
>>> a=10
>>> b=3
>>> c=a<<b
>>> print(c)-----80
>>> print(10<<3)-----80
>>> print(3<<2)-----12
>>> print(100<<2)----400
```

2. Bitwise Right Shift Operator (>>)

Syntax: varname=GivenNumber >> No. of Bits

Concept: The Bitwise Right Shift Operator (>>) shifts No. of Bits Towards Right Side -----By Adding No. of Zero at Left Side (depends on No. of Bits). So that result value will displayed in the form of decimal number system .

Examples:

```
>>> a=10
>>> b=3
>>> c=a>>b
>>> print(c)-----1
>>> print(10>>2)-----2
>>> print(25>>4)-----1
>>> print(24>>3)-----3
>>> print(50>>4)-----3
>>> print(45>>2)-----11
```

3. Bitwise AND Operator (&)

=>Syntax: var name= Var1 & Var2

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

| Var1 | Var2 | Var1 & Var2 |
|------|------|-------------|
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 0 | 0 | 0 |
| 1 | 1 | 1 |

Examples:

```
>>>a=10----->1010
>>>b=3-----> 0011
```

```
>>>c=a&b----->0010
>>>print(c)---2
```

```
>>> 10&3-----2
>>> 10 and 3-----3
>>> 5&4-----4
>>> 10&15-----10
>>> 10 and 15-----15
```

4. Bitwise OR Operator (|)

=>Syntax: var name= Var1 | Var2

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

| Var1 | Var2 | Var1 Var2 |
|------|------|-------------|
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 0 | 0 | 0 |
| 1 | 1 | 1 |

Examples:

```
>>>a=10----->1010
>>>b=15----->1111
>>>c=a|b----->1111
>>>print(c)---15
```

```
>>> print(3|4)-----7
>>> print(5|4)-----5
```

5. Bitwise Complement Operator (~)

=>The Internal Functionality of Bitwise Complement Operator (~) is that "To Invert the Bits of Given Number ".

=>The formula for computing Complement of Given number = - (Number+1)

=>Syntax: varname = ~ Number

Examples

```
>>> print(~10)-----11
>>> print(~98)-----97
>>> print(~1000)-----1001
```

6. Bitwise XOR Operator (^)

=>**Syntax:** var name= Var1 ^ Var2

=>Similar Bits is Zero and Dis-similar bits is 1

=>The Functionality of Bitwise xOR Operator (^) is expressed by using the following Truth table.

| Var1 | Var2 | Var1 ^ Var2 |
|------|------|-------------|
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 0 | 0 | 0 |
| 1 | 1 | 0 |

Examples:

```
>>>a=2-----0010
>>>b=3-----0011
```

```
>>>c=a^b-----0001---Result is 1
```

```
>>>print(c)
>>> print(10^15)-----5
```

Special Logic

```
>>>a=10
>>>b=20
>>>print(a,b) # 10 20
>>>a=a^b
```

```
>>>b=a^b
>>>a=a^b
>>>print(a,b) # 20 10
```

Examples:

```
>>>n=5          0 1 0 1
>>>res = ~n      1 0 1 0
It is the value of -6
```

Proof: I want to store -5 ----> 2's complement of 5

- 1) Take Bin Bits of 5 = 0 1 0 1
- 2) 1's Complement of 5= 1 0 1 0
- 3) 2's Complement of 5= 1 's Complement of 5 + 1

$$\begin{array}{r} 1001 + 1 \\ \hline => 1001 \\ 0001 \\ \hline \end{array}$$
2's complement 6: 1 0 1 0 (also ~ 5)

Examples:

```
>>>n=4          0 1 0 0
>>>res=~n      1 0 1 1 (Which is nothing but value of -5)
>>>print(res)--> -5
```

Proof: I want to store -5 ----> 2's complement of 5

- 1) Take Bin of 5 = 0 1 0 1
- 2) 1's Complement of 5= 1 0 1 0
- 3) 2's Complement of 5= 1 's Complement of 5 + 1

$$\begin{array}{r} & \\ => 1010 \\ => 0001 \\ \hline \end{array}$$
2's Complement of 5 1 0 1 1 (also ~ 4)

Swapping Logic By using Bitwise XOR (^)

```
>>>a=4
>>>b=3
>>>a=a^b          a (4)    b (3)
>>>               4- 7- 3   3- 4
>>>               0100
>>>               0011----->7
>>>               0111
>>>               0011----->4
>>>               0100----->3
>>>               0111
>>>               0100----->3
>>>               0011
```

```
#program for swapping two numerical values by using XOR
#SwapAop.py
a=int(input("Enter Value of a:"))
b=int(input("Enter Value of b:"))
print("*50")
print("Original value of a={}".format(a)) #a=10
print("Original value of b={}".format(b)) # b=20
print("*50")
#swapping logic
a=a+b
b=a-b
a=a-b
```

```

print("Swapped value of a={}".format(a))
print("Swapped value of b={}".format(b))
print("*50)
-----
#program for swapping two numerical values by using XOR
#SwapXOR.py
a=int(input("Enter Value of a:"))
b=int(input("Enter Value of b:"))
print("*50)
print("Original value of a={}".format(a))
print("Original value of b={}".format(b))
print("*50)
#swapping logic
a=a^b
b=a^b
a=a^b
print("Swapped value of a={}".format(a))
print("Swapped value of b={}".format(b))
print("*50)
-----
```

===== 6. Membership Operators =====

=>The purpose of Membership Operators is that "To Check the existence of Value in Iterable Object".

=>An Iterable Object is one, which contains More than One Value

=> Examples : str,bytes,bytearray,range,list,tuple,set,frozenset,dict are called Iterable objects.

=>In Python Programming, we have 2 types of Membership Operators. They are

1. in
2. not in

1. in

Syntax: **Value in IterableObject**

----- if the "Value" present in IterableObject then 'in' operator returns True
if the "Value" not present in IterableObject then 'in' operator returns False

2. not in

Syntax: **Value not in IterableObject**

----- if the "Value" not present in IterableObject then 'not in' operator returns True
if the "Value" present in IterableObject then 'not in' operator returns False

Examples:

```
>>> s="PYTHON"
>>> print(s,type(s))-----PYTHON <class 'str'>
>>> "Y" in s
True
>>> "O" in s
True
>>> "O" not in s
False
>>> "K" not in s
True
>>> "K" in s
False
>>> "o" in s
False
>>> "o" not in s
True
-----
>>> s="PYTHON"
>>> print(s,type(s))-----PYTHON <class 'str'>
>>> "PYT" in s
True
>>> "PYT" not in s
False
>>> "HON" not in s
False
>>> "hon" in s
False
>>> "hon" not in s
True
-----
>>> s="PYTHON"
>>> print(s,type(s))-----PYTHON <class 'str'>
>>> "PON" in s
False
>>> "PTO" in s
False
>>> "PN" in s
False
>>> "PH" in s
False
>>> "PH" not in s
True
>>> "PHN" not in s
True
>>> "THN" not in s
True
>>> "THN" in s
False
-----
>>> s="PYTHON"
```

```

>>> print(s,type(s))
PYTHON <class 'str'>
>>> "NOH" in s
False
>>> "OTP" in s
False
>>> "OTP" not in s
True
>>> "NTP" not in s
True
>>> "NTP" in s
False
>>> "HON" not in s[::-1]
True
>>> "PTO" not in s[::-1][::-1][::2]
False
>>> "PTO" not in s[::-1][::2]
True

```

7. Identity Operators

=>The purpose of Identity Operators is that "To Compare the memory address of two objects".

=>In Python Programming, we have 2 Identity Operators. They are

1. is operator
 2. is not operator
-

1. is operator

Syntax: object1 is object2

=>"is" operator returns True provided Object1 and Object2 contains Same Address

=>"is" operator returns False provided Object1 and Object2 contains Different Address

2. is not operator

Syntax: object1 is not object2

=>"is not" operator returns True provided Object1 and Object2 contains Different Address

=>"is not" operator returns False provided Object1 and Object2 contains Same Address

Examples:

Note1: When the objects Participated in DEEP Copy then on those objects "is" operator returns True and "is not" Operator returns False.

```
>>> d1={10:"Python",20:"Apple",30:"Java"}
>>> print(d1,type(d1),id(d1))----{10: 'Python', 20: 'Apple', 30: 'Java'} <class 'dict'>
2890240333888
>>> d2=d1 # Deep Copy
>>> print(d2,type(d2),id(d2))----{10: 'Python', 20: 'Apple', 30: 'Java'} <class 'dict'>
2890240333888
>>> d1 is d2-----True
>>> d1 is not d2-----False
```

Note2: When the objects Participated in SHALLOW Copy then on those objects "is" operator returns False and "is not" Operator returns True.

```
>>> d1={10:"Python",20:"Apple",30:"Java"}
>>> print(d1,type(d1),id(d1))----{10: 'Python', 20: 'Apple', 30: 'Java'} <class 'dict'>
2890240331776
>>> d2=d1.copy() # Shallow Copy
>>> print(d2,type(d2),id(d2))----{10: 'Python', 20: 'Apple', 30: 'Java'} <class 'dict'>
2890243948800
>>> d1 is d2-----False
>>> d1 is not d2----True
```

NOTE:3:

=>In The case of Dict,List,Tuple,Set,Frozenset,range,bytes,bytearray,complex,float Data type related Different Object of Same type with same data then Those Objects of Same type contains Different Address and if we apply "is" operator then it returns False and if apply "is not" Operator then it return True

```
>>> d1={10:"Python",20:"Apple",30:"Java"}
>>> print(d1,type(d1),id(d1))----{10: 'Python', 20: 'Apple', 30: 'Java'} <class 'dict'>
2890240333888
>>> d2={10:"Python",20:"Apple",30:"Java"}
>>> print(d2,type(d2),id(d2))----{10: 'Python', 20: 'Apple', 30: 'Java'} <class 'dict'>
2890240331776
>>> d1 is d2-----False
>>> d1 is not d2----True
```

```
>>> a=None
>>> b=None
>>> print(a,type(a),id(a))----None <class 'NoneType'> 140710431128776
>>> print(b,type(b),id(b))----None <class 'NoneType'> 140710431128776
>>> a is b-----True
>>> a is not b-----False
```

NOTE:4:

=>In The case of str Data type related Different Objects of Same type with same data with same meaning and same case then Those Objects of Same type contains

Same Address and if we apply "is" operator then it returns True and if apply "is not" Operator then it return False

```
>>> s="PYTHON"
>>> t="PYTHON"
>>> print(s,type(s),id(s))-----PYTHON <class 'str'> 2890240510576
>>> print(t,type(t),id(t))
PYTHON <class 'str'> 2890240510576
>>> s is t-----True
>>> s is not t-----False
```

===== Ternary operator in Python (if else operator) =====

=>The name of Ternary operator in Python is " if else " operator

=>**Syntax:**

----- **varname= Expr1 if Test Cond else Expr2**

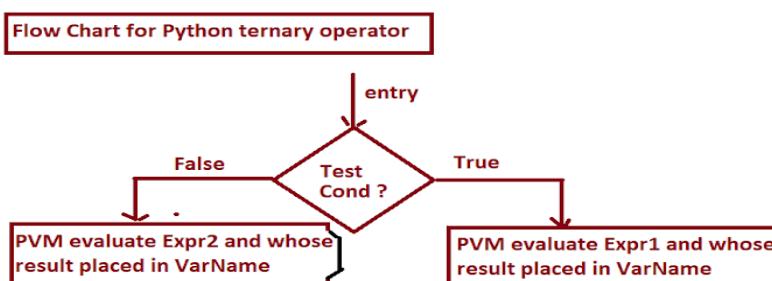
Explanation:

=>here Test Cond is either True or False.

=>If the result of Test Cond is True then PVM executes Expr1 and whose result placed in LHS Varname.

=>If the result of Test Cond is False then PVM executes Expr2 and whose result placed in LHS Varname.

=>In if..else ternary operator at any point of either Expr1 or Expr2 will execute and whose result placed in LHS varname.



```
#TernaryOpEx1.py
a=int(input("Enter Value of a:"))#10
b=int(input("Enter Value of b:"))#20
bv = a if a>b else b
print("big({},{}]={}.".format(a,b,bv))

-----
#TernaryOpEx2.py
a=int(input("Enter Value of a:"))#10
b=int(input("Enter Value of b:"))#10
bv =a if a>b else b if b>a else "Both values are equal"
print("Big({},{}]={}.".format(a,b,bv))

-----
#TerbaryOpEx3.py
a=int(input("Enter Value of a:"))#1000
b=int(input("Enter Value of b:"))#1000
c=int(input("Enter Value of c:"))#1000
bv=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 values are equal"
print("Big({},{}={})={}.".format(a,b,c,bv))

-----
#TerbaryOpEx3.py
a=int(input("Enter Value of a:"))#1000
b=int(input("Enter Value of b:"))#1000
c=int(input("Enter Value of c:"))#1000
bv=a if b<=a>c else b if a<b>=c else c if a<=c>b else "ALL values are equal"
print("Big({},{}={})={}.".format(a,b,c,bv))

-----
#TerbaryOpEx5.py
val=input("Enter any value / word:") # 1222
res="Plindrome" if val==val[::-1] else "Not Palindrome"
print("{}' is {}.".format(val,res))
```

Flow Control Statements in Python

Index

=>Purpose of Flow Control Statements in Python

=>Types of Flow Control Statements

 a) Conditional OR Selection OR Branching Statements

 i) Simple if statement

 ii) if...else statement

 iii) if..elif..else statement

 iv) match case statement

 =>Programming Examples

 b) Looping OR Iterative OR Repetative Statements

 i) while Loop OR while..else Loop

 ii) for loop OR for ..else loop

 =>Programming Examples

c) Transfer Flow Statements

- i) break
- ii) continue
- iii) pass
- iv) return

=>Programming Examples

- =>Inner OR Nested Loop
- i) while loop in while loop
 - ii) for loop in for loop
 - iii) while loop in for loop
 - iv) for loop in while loop

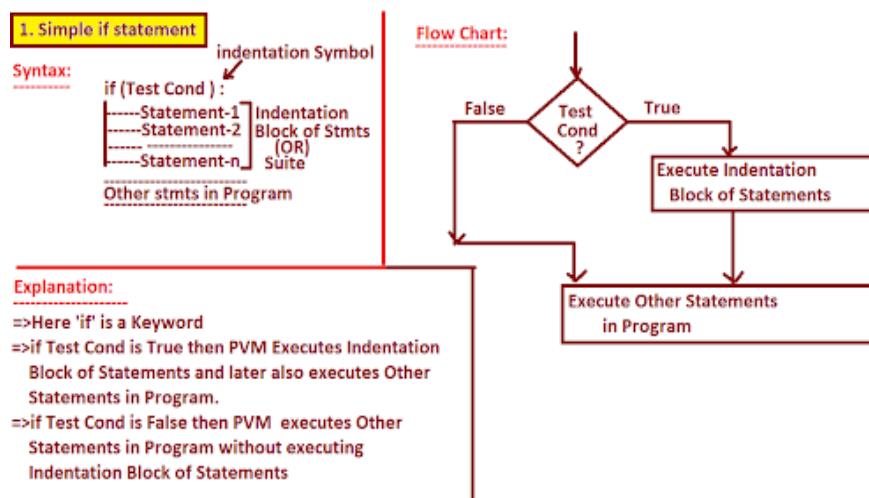
Conditional OR Selection OR Branching Statements

=>The purpose of Conditional OR Selection OR Branching Statements is that "To Perform Certain Operation Only once depends on test condition".

=>In Otherwords, Conditional OR Selection OR Branching Statements are used performing either X-Operation in the case True or Perform Y-Operation only once in the case of False.

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

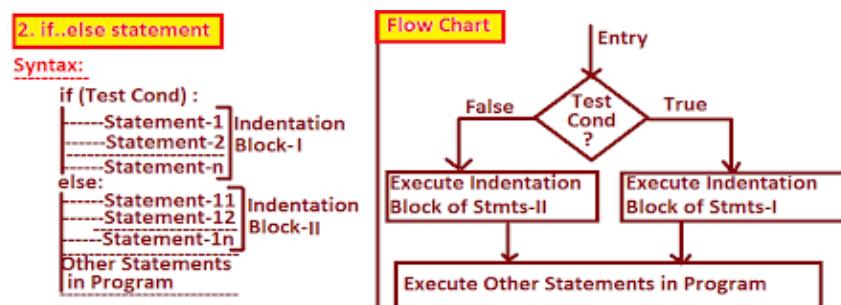
1. Simple if statement
 2. if..else statement
 3. if..elif..else statement
 4. match case statement (only from Python 3.10 Version)
-



```
#Moviee.py
ch=input("Do u have a Ticket(yes/no):")
```

```
if(ch.lower() == "yes"):
    print("enter into theater")
    print("Watch moviee")
    print("Eat Pop Corn")
print("Goto Home ")
print("Read Python Notes")
```

```
#EvenOdd.py
n=int(input("Enter a Number:"))
if(n%2==0):
    print("{} is EVEN".format(n))
if(n%2!=0):
    print("{} is Odd".format(n))
```

**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.

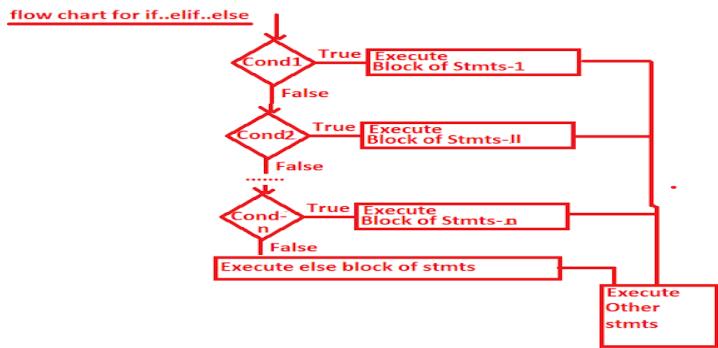
if..elif..else statement:

Syntax:

```
if ( Test Cond 1):
    Block of Stmtns-I
elif( Test Cond 2):
    Block of Stmtns-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.



```

#DigitEx1.py
d=int(input("Enter a digit:"))
if(d==0):
    print("{} is Zero".format(d))
else:
    if(d==1):
        print("{} is ONE".format(d))
    else:
        if(d==2):
            print("{} is TWO".format(d))
        else:
            if(d==3):
                print("{} is THREE".format(d))
            else:
                if(d==4):
                    print("{} is FOUR".format(d))
                else:
                    if(d==5):
                        print("{} is FIVE".format(d))
                    else:
                        if(d==6):
                            print("{} is SIX".format(d))
                        else:
                            if(d==7):
                                print("{} is SEVEN".format(d))
                            else:
                                if(d==9):
                                    print("{} is NINE".format(d))
                                else:
                                    if(d==8):
                                        print("{} is EIGHT".format(d))
                                    else:
                                        print("{} is NUMBER".format(d))
print("{} is NUMBER".format(d))
  
```

```
#DigitEx2.py
d=int(input("Enter a digit:"))
if(d==0):
    print("{} is Zero".format(d))
elif(d==1) :
    print("{} is ONE".format(d))
elif(d==2):
    print("{} is TWO".format(d))
elif(d==3):
    print("{} is THREE".format(d))
elif(d==4):
    print("{} is FOUR".format(d))
elif(d==5):
    print("{} is FIVE".format(d))
elif(d==6):
    print("{} is SIX".format(d))
elif(d==7):
    print("{} is SEVEN".format(d))
elif(d==8):
    print("{} is EIGHT".format(d))
elif(d==9):
    print("{} is NINE".format(d))
else:
    print("{} is NUMBER".format(d))
print("Program execution Completed")
```

```
#DigitEx3.py
d=int(input("Enter a digit:"))
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==6):
    print("{} is SIX".format(d))
if(d==7):
    print("{} is SEVEN".format(d))
if(d==8):
    print("{} is EIGHT".format(d))
if(d==9):
    print("{} is NINE".format(d))
if(d>9) :
    print("{} is NUMBER".format(d))
```

```

if(d<0):
    print("{} is Negative Digit/ NUMBER".format(d))

print("Program execution Completed")
-----
#DigitEx4.py
dobj={0:"ZERO",1:"ONE",2:"TWO",3:"THREE",4:"FOUR",5:"FIVE",6:"SIX",7:"SEVEN",
",8:"EIGHT",9:"NINE"}
d=int(input("Enter a digit:"))
res=dobj.get(d) if dobj.get(d)!=None else "Number"
print("{} is {}".format(d,res))
-----
#greetMsgIfElseEx1.py
pname=input("Enter Ur Name:")
if(pname.lower() == "rossum"):
    print("Hi {}, Good Morning ".format(pname))
elif(pname.lower() == "travis"):
    print("Hi {}, Good Evening ".format(pname))
elif(pname.lower() == "kinney"):
    print("Hi {}, Good Night".format(pname))
elif(pname.lower() == "jim"):
    print("Hi {}, Good afternoon".format(pname))
else:
    print("Hello {}, Plz Join Python Classes".format(pname))
print("Program execution completed")
-----
#PosNegZeroElseEx1.py
n=int(input("Enter a number:"))#n=-34
if(n>0):
    print("{} is +VE".format(n))
else:
    if(n==0):
        print("{} is Zero".format(n))
    else:
        print("{} is -VE".format(n))
=====
```

d) match case statement.

=>Here "match case" is one the new feature in Python 3.10 Version onwards
=>match case statement is recommended to take in deciding Pre-designed Conditions in Menu Driven Applications.

=>Syntax:

```

match(Choice Expr):
    case Choice Label1:
```

```

        Block of Stements-1
case Choice Label2:
        Block of Stements-2
case Choice Label3:
        Block of Stements-3
-----
case Choice Label-n:
        Block of Stements-n
case _: # Default Case Block
        default Block of Statements
-----
```

Other Statements in Program

Explanation:

- =>here "match" and "case" are the keywords
- =>"Choice Expr" represents either int or str or bool
- =>If "Choice Expr" is matching with "case label1" then PVM executes Block of Statements-1 and later executes Other statements in program.
- =>If "Choice Expr" is matching with "case label2" then PVM executes Block of Statements-2 and later executes Other statements in program.
- =>In General "Choice Expr" is trying match with case label-1, case label-2,...case label-n then PVM executes corresponding block of statements and later executes Other statements in program.
- =>If "Choice Expr" is not matching with any of the specified case labels then PVM executes Default Block of Staements which are written under default case block(case _) and later executes Other statements in program.
- =>Writing default case block is optional and If we write then it must be written at last (Otherwise we get SyntaxError)
- =>When we represent multiple case labels under one case then those case labels must be combined with Bitwise OR Operator (|) .

```

#MatchCaseEx2.py
import sys
print("*50)
print("\tArea of Figures")
print("*50)
print("\tR. Rectangle")
print("\tS. Square")
print("\tC. Circle")
print("\tT. Triangle")
print("\tE. Exit")
print("*50)
ch=input("Enter Ur Choice:")
match(ch):
    case "R":
        l=float(input("Enter Length:"))
        b=float(input("Enter Breadth:"))
        ar=l*b
        print("Area of Rectangle={} .format(ar))
```

```

case "S":
    s=float(input("Enter Side:"))
    sa=s**2
    print("Area of Square:{}".format(sa))
case "C":
    r=float(input("Enter Radius:"))
    ca=3.14*r**2
    print("Area of Circle:{}".format(ca))
case "T":
    b=float(input("Enter Base:"))
    h=float(input("Enter Height:"))
    at=(1/2)*b*h
    print("Area of Triangle:{}".format(at))
case "E":
    print("Thx for using this Progrsm")
    sys.exit() # will terminate the program execution Physically
case _: # default case block
    print("Ur Selection of Operation is Wrong-try again")

```

```

#MatchCaseEx3.py
import sys
print("*50)
print("\tArea of Figures")
print("*50)
print("\tR. Rectangle")
print("\tS. Square")
print("\tC. Circle")
print("\tT. Triangle")
print("\tE. Exit")
print("*50)
ch=input("Enter Ur Choice:")
match(ch):
    case "R" | "r":
        l=float(input("Enter Length:"))
        b=float(input("Enter Breadth:"))
        ar=l*b
        print("Area of Rectangle={}".format(ar))
    case "S" | "s":
        s=float(input("Enter Side:"))
        sa=s**2
        print("Area of Square:{}".format(sa))
    case "C" | "c":
        r=float(input("Enter Radius:"))
        ca=3.14*r**2
        print("Area of Circle:{}".format(ca))
    case "T" | "t":
        b=float(input("Enter Base:"))
        h=float(input("Enter Height:"))
        at=(1/2)*b*h
        print("Area of Triangle:{}".format(at))

```

```

case "E"|"e":
    print("Thx for using this Progrsm")
    sys.exit() # will terminate the program execution Physically
case _: # default case block
    print("Ur Selection of Operation is Wrong-try again")
-----
```

```

#MatchcaseEx4.py
wkn=input("Enter the week Name:")
match(wkn.upper()):
    case "MONDAY":
        print("{} is working Day".format(wkn))
    case "TUESDAY":
        print("{} is working Day".format(wkn))
    case "WEDNESDAY":
        print("{} is working Day".format(wkn))
    case "THURSDAY":
        print("{} is working Day".format(wkn))
    case "FRIDAY":
        print("{} is working Day".format(wkn))
    case "SATURDAY":
        print("{} is weak end".format(wkn))
    case "SUNDAY":
        print("{} is Holiday".format(wkn))
    case _:
        print("{} not a week day:".format(wkn))
-----
```

```

#MatchcaseEx5.py
wkn=input("Enter the week Name:")
match(wkn.upper()):
    case "MONDAY" |"TUESDAY" | "WEDNESDAY" | "THURSDAY" |"FRIDAY":
        print("{} is working Day".format(wkn))
    case "SATURDAY":
        print("{} is weak end".format(wkn))
    case "SUNDAY":
        print("{} is Holiday".format(wkn))
    case _:
        print("{} not a week day:".format(wkn))
-----
```

```

#MatchcaseEx6.py
wkn=input("Enter the week Name:")
match(wkn.upper()):
    case "MON" |"TUE" | "WED" | "THU" |"FRI":
        print("{} is working Day".format(wkn))
    case "SAT":
        print("{} is weak end".format(wkn))
    case "SUN":
        print("{} is Holiday".format(wkn))
    case _:
        print("{} not a week day:".format(wkn))
-----
```

```
#MatchcaseEx7.py
wkn=input("Enter the week Name:")
if wkn.upper() in
["MONDAY","TUESDAY","WEDNESDAY","THURSDAY","FRIDAY","SATURDAY","S
UNDAY"]:
    match(wkn[:3].upper()):
        case "MON"|"TUE"|"WED"|"THU"|"FRI":
            print("{} is working Day".format(wkn))
        case "SAT":
            print("{} is weak end".format(wkn))
        case "SUN":
            print("{} is Holiday".format(wkn))
else:
    print("{} is not a week day".format(wkn))
```

===== Looping OR Iterative OR Repetitive Statements =====

=>The purpose of Looping OR Iterative OR Repetitive Statements " To Perform Certain Operation Repeatedly for finite number of Times until Test Condition Becomes False".

=>When we are dealing with Loop related Programs, we must ensure that there must exist 3 parts. They are

1. Initialization Part
2. Conditional Part
3. Updation Part

=>In Python Programming, we have 2 Types of Looping OR Iterative OR Repetitive Statements. They are

1. while loop OR while..else loop
2. for loop OR for .. else Loop

```
#Program for generating Even Numbers within n where n is +ve
#EvenNumGenEx1.py
n=int(input("Enter How Many Numbers u want to generate:"))
if(n<=0):
    print("{} is invalid".format(n))
else:
    print("=" * 50)
    print("Even Numebrs within:{}".format(n))
    print("=" * 50)
    i=2 # Init Part
    while(i<=n): # Cond Part
        print("{}".format(i),end="\t")
        i+=2 #Updation
    else:
        print()
    print("=*50)
```

```
#Program for Multable
#MulTableEx1,.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+=1
    else:
        print("=*50)
```

```
#Program for natural numbers Product
#NatNumsProductEx1.py
n=int(input("Enter How many natural Numbs Product u want:"))
if(n<=0):
    print("{} is invalid input".format(n))
else:
    print("=" * 50)
    print("Product of {} Nat Numbers".format(n))
    print("=" * 50)
    s=1 # Inital Product Value
    i=1 # Initialization Part
    while(i<=n):
        print("\t{}".format(i))
        s=s*i # Accumulaing Nat Numbs Product
        i=i+1
    else:
```

```

print("=" * 50)
print("Product={}".format(s))
print("=" * 50)
-----
```

```

#Program for generating numberes within n where n is +ve
#NumGenEx1.py
n=int(input("Enter How many numbers u want generate:"))
if(n<=0):
    print("{} is invalid input:".format(n))
else:
    print("*50")
    print("Numbers within :{}".format(n))
    print("=" * 50)
    i=1 # Init Part
    while(i<=n): # Cond Part
        print("\t{}".format(i))
        i=i+1 # Updation Part
    else:
        print("****50")
print("\nProgram execution completed")
```

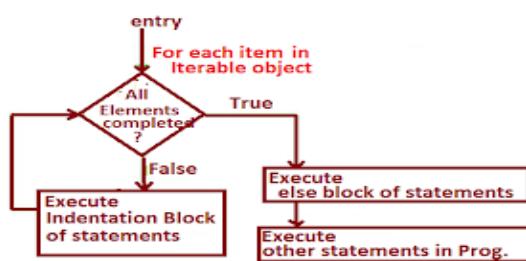
```

-----
```

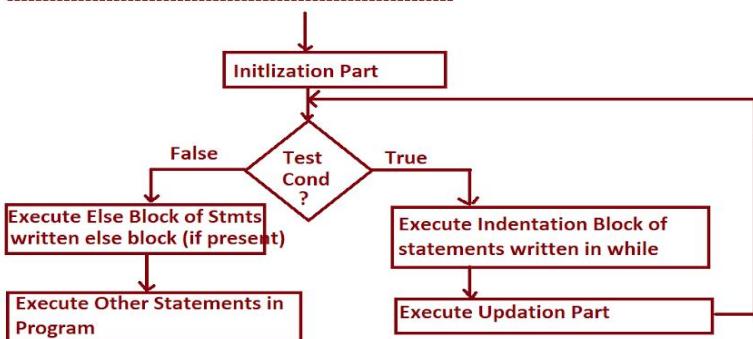
```

#Program for generating numberes within n where n is +ve from n to 1
#NumGenEx2.py
n=int(input("Enter How Many Numbers u want to generate:"))
if(n<=0):
    print("{} is invalid".format(n))
else:
    print("****50")
    print("Numbers within :{}".format(n))
    print("****50")
    i=n # Init Part
    while(i>0): # Cond Part
        print("\t\t{}".format(i))
        i-=1 #updation -- here -= is called Short hand Operat
    else:
        print("****50")
-----
```

Flow Chart of for loop



Flow Chart for while loop OR while ..else Loop



Syntax:

```
while( Test Cond ) :
    --->Statement-1
    --->Statement-2
    --->Statement-n
    ] Indentation
    Block of stmnts
    ; Other statements in Prog
```

Syntax:

```
while( Test Cond ) :
    --->Statement-1
    --->Statement-2
    --->Statement-n
    ] Indentation
    Block of stmnts
    ; OR
    else:
        --->Else Block of Stmts
    ; Other Stmts in Program
```

Explanation:

- =>here 'while' and 'else' are called Keywords
- =>Here Test Cond can be either True or False.
- =>If the Test Cond is True then PVM executes Indentation Block of statements and once again PVM Control goes to Test Cond. If once again Test Cond is True then PVM executes Indentation Block of statements. This Process repeated for finite number of times until Test Cond becomes false.
- =>Once Test Cond becomes false, PVM executes else block of statements which are written else block(if present) and later 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' , "in" 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.

=> when all the elements of Iterable Object completed then PVM comes out of for loop and executes else block of statements which are written under else block

=>Writing else block is optional.

#Program for generating Even Numbers within n where n is +ve

#EvenNumGenEx1.py

n=int(input("Enter How Many Numbers u want to generate:"))

if(n<=0):

print("{} is invalid".format(n))

else:

print("=" * 50)

print("Even Numebrs within:{}".format(n))

print("=" * 50)

```

i=2 # Init Part
while(i<=n): # Cond Part
    print("{}\t".format(i))
    i+=2 #Updation
else:
    print()
    print("=*50)
-----
#ForLoopWhileLoopEx2.py
lst=[100,"Rossum","Python",34.56,True]
print("=====for Loop=====")
for val in lst:
    print("\t{}\t".format(val))
else:
    print("=====")
-----
#Program for generating Mul table
#MulTableForEx1.py
n=int(input("Enter a Number for generating Mul table:"))
if(n<=0):
    print("{} is Invalid".format(n))
else:
    print("-----")
    print("Mul Table for :{}".format(n))
    print("-----")
    for i in range(1,11):
        print("\t{} x {} = {}".format(n,i,n*i))
    else:
        print("=====")
-----
#Program for natural numbers Product
#NatNumsProductEx1.py
n=int(input("Enter How many natural Numbs Product u want:"))
if(n<=0):
    print("{} is invalid input".format(n))
else:
    print("=* 50)
    print("Product of {} Nat Numbers".format(n))
    print("=* 50)
    s=1 # Initial Product Value
    i=1 # Initialization Part
    while(i<=n):
        print("\t{}\t".format(i))
        s=s*i # Accumulating Nat Numbs Product
        i=i+1
    else:
        print("=* 50)
        print("Product={}\t".format(s))
        print("=* 50)

```

```

#Program for natural numbers Product
#NatNumsProductEx1.py
n=int(input("Enter How many natural Numbs Product u want:"))
if(n<=0):
    print("{} is invalid input".format(n))
else:
    print("=" * 50)
    print("Product of {} Nat Numbers".format(n))
    print("=" * 50)
    s=1 # Inital Product Value
    i=1 # Initialization Part
    while(i<=n):
        print("\t{}".format(i))
        s=s*i # Accumutaing Nat Nums Product
        i=i+1
    else:
        print("=" * 50)
        print("Product={}".format(s))
        print("=" * 50)
#Program for generating Mul table
#NatSumsForEx1.py
n=int(input("Enter a Natural Numbers u want:"))
if(n<=0):
    print("{} is Invalid".format(n))
else:
    print("*50")
    print("Nat Numbs\tNat Num Squares\tNat Num Cubes")
    print("*50")
    s,ss,cs=0,0,0 # Multi Line assignment
    for i in range(1,n+1):
        print("\t{}\t{}\t{}".format(i,i**2,i**3))
        s,ss,cs=s+i,ss+i**2,cs+i**3 # Multi Line assignment
    else:
        print("*50")
        print("\t{}\t{}\t{}".format(s,ss,cs))
        print("*50")

```

```

#Program for generating 1 to n where n is +ve
#NumGenForEx1.py
n=int(input("Enter How Many Numbers u want to generate:"))
if(n<=0):
    print("{} is Invalid".format(n))
else:
    print("-----")
    print("Numbers within:{}".format(n))
    print("-----")
    for i in range(1,n+1):
        print("\t{}".format(i))
    else:
        print("***50")

```

```

#Program for generating n to 1 where n is +ve
#NumGenForEx2.py
n=int(input("Enter How Many Numbers u want to generate:"))
if(n<=0):
    print("{} is Invalid".format(n))
else:
    print("-----")
    print("Numbers within:{}".format(n))
    print("-----")
    for val in range(n,0,-1):
        print("\t{}".format(val))
    else:
        print("=====")
#program for accepting a word / line of Text and display Vowels
#VowelsCountEx1.py
line=input("Enter a Line of Text:") # line= apple Line=Python is an oop lang
print("-----")
print("Numbe of Vowels in {}".format(line))
print("-----")
nov=0
for ch in line:
    if ch in ['a','e','i','o','u']:
        print("\t{}".format(ch))
        nov+=1
    else:
        print("-----")
        print("Number of Vowels:{}".format(nov))
        print("-----")
-----
#Program for accepting a word or Line Text and find number of occurences of Every
Letter in text/word
#WordOccEx1.py
line=input("Enter a Line of Text / word:") # APPLE {A:1, P:2,L:1,E:1}---- {}
d={} # OR d=dict()---- Creating empty dict d={A:1,P:2,L:1,E:1}
for ch in line:
    if ch not in d:
        d[ch]=1
    elif(ch in d):
        d[ch]=d[ch]+1
else:
    print("-----")
    print("\tLetter\t\tOccurrences")
    print("-----")
    noc=0
    for let,occ in d.items():
        print("\t{}\t\t{}".format(let,occ))
        noc=noc+occ
    print("-----")
    print("Length of Str=",noc)

```

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 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
```

```
#program for demonstrating break statement  
#breakex2py  
s="PYTHON"  
i=0  
while(i<len(s)):  
    print(s[i])  
    i=i+1  
else:  
    print("=====")  
#while loop with break  
i=0  
while(i<len(s)):  
    if(s[i]=="O"):  
        break  
    else:  
        print(s[i])  
        i=i+1  
else:  
    print("while loop--else part")  
print("Other statements in program")
```

```
#program for demonstrating break statement
#breakex2py
s="PYTHON"
i=0
while(i<len(s)):
    print(s[i])
    i=i+1
else:
    print("====")
#while loop with break
i=0
while(i<len(s)):
    if(s[i]=="O"):
        break
    else:
        print(s[i])
        i=i+1
else:
    print("while loop--else part")
print("Other statements in program")
```

```
#primeex3.py
#Program for deciding whether the number prime or not
n=int(input("Enter a number:"))#n=5
if(n<=1):
    print("{} invalid input:".format(n))
else:
    s=False
    for i in range(2,n):
        if(n%i==0):
            s=True
            break
    if(not s):
        print("{} is Prime:".format(n))
    else:
        print("{} is not Prime".format(n))
```

```
#program for demonstrating continue statement
#continueex1.py
s="PYTHON" # Output PYTON
for ch in s:
    if(ch=="H"):
        continue
    else:
        print(ch)
else:
    print("i am from for loop--else part")
print("Other part of the program")
```

```
#program for demonstrating continue statement
#continueex2.py
s="PYTHON" # Ouput PYHN
for ch in s:
    if(ch=="T") or (ch=="O"):
        continue
    print(ch)
else:
    print("i am from for loop--else part")
print("Other part of the program")
```

```
#program for demonstrating continue statement
#continueex3.py
s="PYTHON" # Ouput PYHN
i=0
while(i<len(s)):
    if(s[i]=="T") or (s[i]=="O"):
        i=i+1
        continue
    else:
        print(s[i])
        i=i+1
print("Other part of the program")
```

```
#program accepting list of words and obtain palindrome words and their length
#continueex5.py
nameslist=[] # create an empty list
while(True):
    names=input("Enter the Name(press 'stop' to terminate the program):")
    if(names.lower() == "stop"):
        break
    else:
        nameslist.append(names)
print("Names List:{} ".format(nameslist)) # ['MOM', 'DAD', 'LIRIL', 'MADAM', 'PAVAN',
'RajesH', 'PYTHON', 'SAIRAM', 'ZAKI']
#code for obtaing Palindromes
pwords=[]
pdw={}
for val in nameslist:
    if(val!=val[::-1]):
        continue
    else:
        pwords.append(val)
        pdw[val]=len(val)
else:
    print("List of words=",pwords)
    print("List of words with length=",pdw)
    print("=====")
    for k,v in pdw.items():
```

```

        print("\t{}\t{}".format(k,v))
else:
    ml=max(pdw.values())
    for k,v in pdw.items():
        if(v==ml):
            hpwd=k
            break
print("Highest Palindrome Length=",hpwd)

```

===== Inner OR Nested Loops =====

=>The Process of Defining One Loop inside of another Loop is called Nested OR Inner Loop

=>The Execution Process of Nested Loops is that "For Every Value of Outer Loop, Inner Loop executes

repeatedly for Finite Number of Times".

=>Inner OR Nested Loops Can be Used with 4 Syntaxes.

Syntax-1 : for loop in for loop

```

for Varname1 in Iterbale_object1: # Outer For Loop
-----
-----
for Varname2 in Iterbale_object2: # Inner For Loop
-----
-----
else:
-----
-----
else:
-----
```

Example: Refer NestedLoopEx1.py

OUTPUT:

```
D:\KVR-PYTHON-9AM\LOOPS>py NestedLoopEx1.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

Syntax-2: while loop i in while loop

while(Test Cond1):

 while(Test Cond2):

 else:

 else:

=====

Examples: Refer NestedLoopEx2.py

D:\KVR-PYTHON-9AM\LOOPS>py NestedLoopEx2.py

Val of i-Outer while Loop:1
Val of j-inner while Loop:1
Val of j-inner while Loop:2
Val of j-inner while Loop:3
I am out of inner while loop

Val of i-Outer while Loop:2
Val of j-inner while Loop:1
Val of j-inner while Loop:2
Val of j-inner while Loop:3

I am out of inner while loop

Val of i-Outer while Loop:3

 Val of j-inner while Loop:1

 Val of j-inner while Loop:2

 Val of j-inner while Loop:3

I am out of inner while loop

Val of i-Outer while Loop:4

 Val of j-inner while Loop:1

 Val of j-inner while Loop:2

 Val of j-inner while Loop:3

I am out of inner while loop

I am out of outer while loop

Syntax-3 : while loop in for loop

```
for Varname1 in Iterbale_object1: # Outer For Loop
```

```
    while(Test Cond2): # Inner while loop
```

```
    else:
```

```
    else:
```

Examples: Refer NestedLoopEx3.py

D:\KVR-PYTHON-9AM\LOOPS>py NestedLoopEx3.py

Val of i-Outer for Loop:1

 Val of j-inner while Loop:3

 Val of j-inner while Loop:2

 Val of j-inner while Loop:1

I am out of inner while loop

Val of i-Outer for Loop:2

 Val of j-inner while Loop:3

 Val of j-inner while Loop:2

 Val of j-inner while Loop:1

I am out of inner while loop

Val of i-Outer for Loop:3
 Val of j-inner while Loop:3
 Val of j-inner while Loop:2
 Val of j-inner while Loop:1
 I am out of inner while loop

Val of i-Outer for Loop:4
 Val of j-inner while Loop:3
 Val of j-inner while Loop:2
 Val of j-inner while Loop:1
 I am out of inner while loop

I am out of outer for loop

Syntax-4: for loop in while loop

```
while(Test Cond1): # Outer while loop
    -----
    for Varname2 in Iterbale_object2: # Inner For Loop
        -----
        else:
            -----
            else:
                -----
```

Examples: Refer NestedLoopEx4.py

D:\KVR-PYTHON-9AM\LOOPS>py NestedLoopEx4.py

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

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: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: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
-----
```

```
I am out of outer while loop
-----
```

Program for demonstarting for loop in while loop in for loop

```
#NestedLoopEx3.py
for i in range(1,5):
    print("-"*50)
    print("Val of i-Outer for Loop:{}".format(i))
    j=3
    while(j>=1): # Inner Loop
        print("\tVal of j-inner while Loop:{}".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")
    print("-"*50)
"""
-----
```

```
D:\KVR-PYTHON-9AM\LOOPS>py NestedLoopEx3.py
-----
```

```

Val of i-Outer for Loop:1
  Val of j-inner while Loop:3
  Val of j-inner while Loop:2
  Val of j-inner while Loop:1
I am out of inner while loop
-----
```

```

-----
```

```

Val of i-Outer for Loop:2
  Val of j-inner while Loop:3
  Val of j-inner while Loop:2
  Val of j-inner while Loop:1
I am out of inner while loop
-----
```

```

-----
```

```

Val of i-Outer for Loop:3
  Val of j-inner while Loop:3
  Val of j-inner while Loop:2
```

Val of j-inner while Loop:1
 I am out of inner while loop

Val of i-Outer for Loop:4
 Val of j-inner while Loop:3
 Val of j-inner while Loop:2
 Val of j-inner while Loop:1
 I am out of inner while loop

I am out of outer for loop

```
#Program for generating 1 to n mul tables where n is +ve
#NestedLoopEx5.py
n=int(input("Enter How Many Mul Tables u want to generate:")) # n=5
if(n<=0):
    print("{} is invalid Input".format(n))
else:
    for num in range(1,n+1): # Outer for supply the numbers
        print("*"*50)
        print("Mul Table for :{}".format(num))
        print("*"*50)
        for i in range(1,11): # Inner for loop generates mul table
            print("\t{} x {}={}".format(num,i,num*i))
        else:
            print("*"*50)
```

```
#Program for generating different Mul tables for list of +Ve Values
#NestedLoopEx6.py
lst=[]
while(True):
    n=input("Enter a Number for generating mul table(press stop) to terminate
Prog:")
    if(n.lower() == "stop"):
        break
    else:
        lst.append(int(n))#converting str value into int type and append to list
#other statements in program
if(len(lst)==0):
    print("List is empty--can't generate Mul tables")
else:
    print("Given List of Numbers:{}".format(lst))
    print("-----")
    for num in lst: # Outer for loop supply values from list
        if(num<=0):
            print("{} is invalid Input".format(num))
            continue
        print("MUL table for :{}".format(num))
        print("-----")
```

```

for i in range(1,11):
    print("\t{} x {}={}".format(num,i,num*i))
else:
    print("-"*50)
-----
#Program for generating different Numebrs and get Primes and non-primes
#NestedLoopEx8.py
lst=list() # creating empty list
while(True):
    n=input("Enter a Numbers for getting primes ands non-primes(press @ ) to
terminate Prog:")
    if(n=="@"):
        break
    else:
        lst.append(int(n))#converting str value into int type and append to list
if(len(lst)==0):
    print("List is empty and nothing to find about primes and non-primes")
else:
    nonpslist=list() # For placing non-primes
    pslist=list() # For placing Primes
    for num in lst:
        if(num<=1):
            nonpslist.append(num)
        else:
            res=True
            for i in range(2,num):
                if(num%i==0):
                    res=False
                    break
            if(res):
                pslist.append(num)
            else:
                nonpslist.append(num)
    else:
        print("=====")
        print("Given List:{}".format(lst))
        print("Non-Primes List:{}".format(nonpslist))
        print("Primes List:{}".format(pslist))
        print("=====")

```

```

#VoterEx1.py
age=int(input("Enter age of the Citizen:"))
if(age>=18):
    print("{} Years Old Citizen Eligible to Vote".format(age))
else:
    print("{} Years Old Citizen not Eligible to Vote".format(age))
-----
```

```

#VoterEx2.py
while(True):
    age=int(input("Enter age of the Citizen:")) # 22
    if(age>=18) and (age<=100):
        break
    else:
        print("\t{} is invalid age-try again".format(age))
print("{} Years Old Citizen Eligible to Vote".format(age))

-----
#Program for generating Marks Report
#StudentmarksReport.py
#validation of student number(100-200)
while(True):
    sno=int(input("Enter Student Number:"))
    if(sno>=100) and (sno<=200):
        break
    else:
        print("\t{} is Invalid student number:".format(sno))

#validation of student name
sname=input("Enter Student Name:")
#validation of C Lang Marks
while(True):
    cm=int(input("Enter Marks in C Lang:"))
    if(cm>=0) and (cm<=100):
        break
    else:
        print("\t{} InValid Marks in C lang".format(cm))

#validation of CPP Lang Marks
while(True):
    cppm=int(input("Enter Marks in CPP Lang:"))
    if(cppm>=0) and (cppm<=100):
        break
    else:
        print("\t{} InValid Marks in CPP lang".format(cppm))

#validation of PYTHON Lang Marks
while(True):
    pym=int(input("Enter Marks in PYTHON Lang:"))
    if(pym>=0) and (pym<=100):
        break
    else:
        print("\t{} InValid Marks in PYTHON lang".format(pym))

#Calculate total marks and Percentage
totmarks=cm+cppm+pym
percent=(totmarks/300)*100
#give grade=Fail
if(cm<40) or (cppm<40) or (pym<40):
    grade="FAIL"
else:

```

```

if(totmarks>=250) and (totmarks<=300):
    grade="DISTINCTION"
elif (totmarks >= 200) and (totmarks <= 249):
    grade = "FIRST"
elif (totmarks >= 150) and (totmarks <= 199):
    grade = "SECOND"
elif (totmarks >= 120) and (totmarks <= 149):
    grade = "THIRD"
#Display the marks report
print("*"*50)
print("\tS T U D E N T M A R K S R E P O R T")
print("*"*50)
print("\tStudent Number:{}".format(sno))
print("\tStudent Name:{}".format(sname))
print("\tStudent Marks in C:{}".format(cm))
print("\tStudent Marks in C++:{}".format(cppm))
print("\tStudent Marks in PYTHON:{}".format(pym))
print("\tStudent Total Marks:{}".format(totmarks))
print("\tStudent Percentage of Marks:{}".format(percent))
print("\tStudent Grade:{}".format(grade))
print("*"*50)

```

===== 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

=>This Function is used for capitalizing the first letter First word of a given Sentence only.

1) **capitalize()**

=>**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 anu 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.isalnum()

=>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 delimiter 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'
>>>

-----
#Program for swapping the case by using swapcase()
#SwapCaseEx1.py
s=input("Enter a word / line of text:")
print("-----")
print("Given Data={}".format(s))
print("-----")
s1=s.swapcase()
print("Swaped case Data={}".format(s1))

-----
#Program for swapping the case without using swapcase()
#SwapCaseEx2.py
s=input("Enter a word / line of text:")
print("-----")
print("Given Data={}".format(s)) # PyThOn
print("-----")
sc=""
for ch in s:
    if(ch.isupper()):
        sc=sc+ch.lower()
    elif(ch.islower()):
        sc=sc+ch.upper()
    elif(ch.isspace()):
        sc=sc+ch
else:
    print("Swapped Case data:{}".format(sc))

-----
#Program for swapping the case by using swapcase()
#SwapCaseEx1.py
s=input("Enter a word / line of text:")
print("-----")
print("Given Data={}".format(s))
print("-----")
s1=s.swapcase()
print("Swaped case Data={}".format(s1))

-----
#Program for swapping the case without using swapcase()
#SwapCaseEx2.py
s=input("Enter a word / line of text:")
print("-----")

```

```

print("Given Data={}".format(s)) # PyThOn
print("-----")
sc=""
for ch in s:
    if(ch.isupper()):
        sc=sc+ch.lower()
    elif(ch.islower()):
        sc=sc+ch.upper()
    elif(ch.isspace()):
        sc=sc+ch
else:
    print("Swapped Case data:{}".format(sc))

```

Syntax for Defining a Function in Python

| | |
|---|---|
| <code>def functionname(list of formal Params if any):</code> | <---Function Heading |
| <code>"""doc string""" statement-1 statement-2 ----- statement-n</code> | <p><---Function Body</p> <p>Note: Function def=</p> <p>Function Heading +Function Body</p> |

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 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)
-

Functions in Python--Most Imp--10 days

Index

=>Purpose of Functions
 =>Advantages of Functions
 =>Definition of Function
 =>Parts of Functions
 =>Phases in Functions
 =>Syntax for Defining Functions
 =>Number of approaches to define Functions
 =>Programming Examples

=>Parameters and Arguments
 =>Types of Arguments

- 1) Positional Arguments
- 2) Default Arguments
- 3) Keyword Arguments
- 4) Variable Length Arguments
- 5) Keyword Variable Length Arguments

=>Programming Examples

=>Local and global Variables
 =>global keyword
 =>globals()
 =>Programming Examples
 =>Anonymous Functions OR Lambda Functions
 =>Programming Examples

=>Special Functions 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 IT

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

- 1. Un-Structured Programming Lang
 - 2. Structured Programming Lang
-

1. Un-Structured Programming Lang

=>In Un-Structured Programming Lang, we don't have the concept of Functions. So Un-Structured Programming Lang 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. Redundency (Duplication OR Replication) of the code is More

Example: GW-BASIC

2. Structured Programming Lang

=>In Structured Programming Lang, we have the concept of Functions. So Structured Programming Lang 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
5. Redundency (Duplication OR Replication) of the code is Minimized

Example: C,C++,Java,PYTHON,C#.net...etc

Functions in Python

=>The purpose of Functions is that " To Perform Certain Operation and Provides Code Re-Usability".

=>The Advantages of Functions in Python are

1. Application Development Time is Less
2. Application memory Space is Less
3. Application Execution Time is Less
4. Application Performance is Enhanced
5. Redundency (Duplication OR Replication) of the code is Minimized

Definition of Function

=>A Sub Program of Main Program is Called Function
(OR)

=>A Part of Main Program is called Function

Parts of Functions

When we are dealing with Functions, we must ensure that there must exist 2 Parts. They are

1. Function Definition
2. Function Calls

=>Function Definition Exist Only Once and We can Have One or More Function Calls

=>Function Definition will execute when we call the Function through the Function Call. In otherwords Function Definition will not execute by itself.

=>For Function Call there must exist Function Definition otherwise we get NameError

Phases in Functions

=>At the time defining the function definition, we must ensure that, there must exist 3 Phases. They are

1. Functions Takes INPUT
2. Functions PROCESS the Input
3. Function gives RESULT / OUTPUT

#FirstProg1.py

```
def greet(pname): # Function Definition
    print("{}", Good Morning".format(pname))
```

#main program

```
print("Line-6:-i am there after Function Definition")
greet("Rossum") # Function call
greet("Travis")# Function call
greet("Kinney")# Function call
```

#Program for calculating Addition of Two Numbers--Functions

#ApproachEx1.py

#INPUT----->Coming From Function Calls

#PROCESS----->Done in Function Body

#OUTPUT----->Giving Result / Output to the Function Call.

def sumop(k,v):

```
    r=k+v # Here r is called Local var
    return r
```

#main program

```
k=int(input("Enter value of a:"))
v=int(input("Enter value of b:"))
r=sumop(k,v) # Function call
print("Sum({},{})={}".format(k,v,r))
```

#Program for calculating Addition of Two Numbers--Functions

#ApproachEx11.py

def sumop(k,v):

```
    r=k+v # Here r is called Local var
    return r
```

#main program

```
a=int(input("Enter value of a:"))
b=int(input("Enter value of b:"))
c=sumop(a,b) # Function call
```

```

print("Sum({},{}]={})".format(a,b,c))
-----
#Program for calculating Addition of Two Numbers--Functions
#ApproachEx2.py
#INPUT----->taking Input in Function Body
#PROCESS----->process in Function Body
#OUTPUT----->display the Result in Function Body
def sumop():

    #taking Input in Function Body
    k=int(input("Enter value of a:"))
    v=int(input("Enter value of b:"))
    #process in Function Body
    r=k+v
    #display the Result in Function Body
    print("sum({},{}]={})".format(k,v,r))

#main program
sumop() # Function Call
-----
#Program for calculating Addition of Two Numbers--Functions
#ApproachEx3.py
#INPUT----->Coming From Function Calls
#PROCESS----->Done in Function Body
#OUTPUT----->display the Result in Function Body
def sumop(a,b):
    c=a+b
    print("sum({},{}]={})".format(a,b,c))

#main program
a=int(input("Enter value of a:"))
b=int(input("Enter value of b:"))
sumop(a,b) # Function call
-----
#Program for calculating Addition of Two Numbers--Functions
#ApproachEx4.py
#INPUT----->taking Input in Function Body
#PROCESS----->Done in Function Body
#OUTPUT----->Giving Result / Output to the Function Call.
def sumop():
    a=int(input("Enter value of a:"))
    b=int(input("Enter value of b:"))
    c=a+b
    return a,b,c # here return statement can return 1 or more number of values
#main program
a,b,c=sumop() # Function call with multi line assigment
print("sum({},{}]={})".format(a,b,c))
print("=====OR=====")
res=sumop() # Function call with single line assigment--here res is type of tuple

```

```

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

-----
#program for calculating Area and Perimeter of Circle
#CircleAreaPeri.py
def readrad( s ):
    r=float(input("Enter radius for cal {}:".format(s)))
    return r

def area():
    r=readrad("Area") # Function Call
    ac=3.14*r**2
    print("Area of circle={}".format(ac))

def peri():
    r=readrad("Perimeter") # Function Call
    pc=2*3.14*r
    print("Perimeter of circle={}".format(pc))
#main program
area() # Function call
peri()
#Note: One Function defintion can call another another Definition also.

-----
#IterableObjectsFunEx1.py
def displrobjects(obj):
    print("-"*50)
    print("Type of obj=",type(obj))
    print("-"*50)
    for val in obj:
        print("\t",val)
    print("-"*50)

def dispdictobj(d1):
    print("-"*50)
    print("Type of obj=",type(d1))
    print("-"*50)
    for key,val in d1.items():
        print("\t{}---->{}".format(key,val))
    print("-"*50)
#main program
lst=[10,"Rossum",33.33,True] # List Object
tpl=(100,"Travis",55.55,"NumPy","Python",2+3j) # tuple object
st1={100,"Rossum",43.33,True,"Python"} # List Object
displrobjects(lst) # Function call
displrobjects(tpl) # Function call
displrobjects(st1) # Function call
d1={10:"Python",20:"Java",30:"R",40:"Django"} # dictobj
dispdictobj(d1) # Function call
#program for cal simple interest by using Functions

```

```

#SimpleIntEx1.py
def simpleint(p,t,r):
    si=(p*t*r)/100
    totamt=p+si
    return si,totamt

#main program
p=float(input("Enter Principle Amount:"))
t=float(input("Enter Time:"))
r=float(input("Enter Rate of Interest:"))
si,tmt=simpleint(p,t,r) # Function call with Multi line assignment
print("SimpleInt(P={},R={},T={})={}".format(p,t,r,si))
print("Total Amount to pay:{}".format(tmt))

-----
#program for cal simple interest by using Functions
#SimpleIntEx2.py
def simpleint():
    #INPUT
    p=float(input("Enter Principle Amount:"))
    t=float(input("Enter Time:"))
    r=float(input("Enter Rate of Interest:"))
    #PROCESS
    si=(p*t*r)/100
    totamt=p+si
    #RESULT
    print("SimpleInt(P={},R={},T={})={}".format(p,t,r,si))
    print("Total Amount to pay:{}".format(totamt))

#main program
simpleint() # Function Call

-----
#program for cal simple interest by using Functions
#SimpleIntEx3.py
def simpleint(p,t,r):
    si=(p*t*r)/100
    totamt=p+si
    print("SimpleInt(P={},R={},T={})={}".format(p,t,r,si))
    print("Total Amount to pay:{}".format(totamt))

#main program
p=float(input("Enter Principle Amount:"))
t=float(input("Enter Time:"))
r=float(input("Enter Rate of Interest:"))
simpleint(p,t,r) # Function call with Multi line assignment

#
#program for cal simple interest by using Functions
#SimpleIntEx4.py
def simpleint():
    #INPUT
    p=float(input("Enter Principle Amount:"))
    t=float(input("Enter Time:"))

```

```

r=float(input("Enter Rate of Interest:"))
#PROCESS
si=(p*t*r)/100
totamt=p+si
return p,t,r,si,totamt

#main program
p,t,r,si,totamt=simpleint() # Function Call
print("SimpleInt(P={},R={},T={})={}".format(p,t,r,si))
print("Total Amount to pay:{}".format(totamt))
-----
```

Parameters and Arguments

Parameter:

=>The variables used in Function Heading are called Formal Parameters and They are used for Storing the inputs coming Function Calls.
=>The Variables Used in Function Body are called Local Variables / Parameters and They are used for Storing Temporary Results / Function Processing Logic Results.
=>The Values Formal Parameters and Local parameters can be accessed within corresponding Function Definition but not possible to access in Other Part of the Program
Examples: def sumop(a,b): # Here a, b are called Formal parameters
 c=a+b # Here c is local Parameter

Arguments:

=>Arguments are the variables which are used as Variables in Function Calls.
Examples: sumop(10,20) # Here 10 20 are called Argument Values
 (OR)
 a=10
 b=20
 sumop(a,b) # Here a,b are called Argument

=>The relationship between Arguments and Parameters is that all the Values of arguments are passing to Parameters. This Mechanism is called Arguments Passing.

Types of Parameters and Arguments

=>The relationship between Arguments and Parameters is that all the Values of arguments are passing to Parameters. This Mechanism is called Arguments Passing Mechanism.
=>We have 5 types of Arguments Passing Mechanism. They are

- a) Positional Arguments

- b) Default Arguments
 - c) Keyword Arguments
 - d) Variable Length Arguments
 - e) Keyword Variable Length Arguments
-

===== 1) Positional Arguments (or) Parameters =====

=>The Concept of Positional Parameters (or) arguments says that "The Number of Arguments of Function Call must be equal to the number of formal parameters in Function Heading".

=>This Parameter mechanism also recommends to follow 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 to Positional Arguments.

```
#Program for demonstrating Positional Args
#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(100,"RS",33.33) # Function Call
dispstudinfo(101,"DR",13.33) # Function Call
dispstudinfo(102,"TR",23.53) # Function Call
#dispstudinfo("KV",11.11,103) # Function Call--not recommended--order not
followed and meaning Lost
print("*50")
```

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

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

Syntax: for Function Definition with Default Parameters

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

Here param-n-1 and param-n are called "default Parameters".

and param1,param-2... are called "Positional parameters".

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

#Program for demonstrating Default Args

#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(100,"RS",33.33) # Function Call

dispstudinfo(101,"DR",13.33) # Function Call

dispstudinfo(102,"TR",23.53) # Function Call

dispstudinfo(103,"GR",43.53) # Function Call

print("*50)

#Program for demonstrating Default Args

#DefArgsEx2.py

```
def dispstudinfo(sno,sname,marks,crs="PYTHON",cnt="INDIA" ):  
    print("\t{}\t{}\t{}\t{}\t{}\t{}".format(sno,sname,marks,crs,cnt))
```

#main program

print("*50)

print("\tSNO\tNAME\tMARKS\tCourse\tCountry")

print("*50)

dispstudinfo(100,"RS",33.33) # Function Call

dispstudinfo(101,"DR",13.33) # Function Call

dispstudinfo(102,"TR",23.53) # Function Call

dispstudinfo(103,"GR",43.53) # Function Call

print("*50)

#Program for demonstrating Default Args

#DefArgsEx3.py

```
def dispstudinfo(sno,sname,marks,crs="PYTHON",cnt="INDIA" ):  
    print("\t{}\t{}\t{}\t{}\t{}\t{}".format(sno,sname,marks,crs,cnt))
```

#main program

print("*50)

print("\tSNO\tNAME\tMARKS\tCourse\tCountry")

```

print("*50)
dispstudinfo(100,"RS",33.33) # Function Call
dispstudinfo(101,"DR",13.33) # Function Call
dispstudinfo(102,"TR",23.53) # Function Call
dispstudinfo(103,"GR",43.53) # Function Call
dispstudinfo(104,"DT",10.00,"JAVA","USA") # Function Call
dispstudinfo(105,"JB",33.56,cnt="USA")

print("*50)
-----
#Program for demonstrating Default Args
#DefArgsEx4.py
def dispstudinfo1(sno,sname,marks,crs="PYTHON" ):
    print("\t{}\t{}\t{}\t{}".format(sno,sname,marks,crs))

def dispstudinfo2(sno,sname,marks,crs="JAVA" ):
    print("\t{}\t{}\t{}\t{}".format(sno,sname,marks,crs))

#main program
print("*50)
print("\tSNO\tNAME\tMARKS\tCourse")
print("*50)
dispstudinfo1(101,"DR",13.33) # Function Call
dispstudinfo1(102,"TR",23.53) # Function Call
dispstudinfo1(104,"TS",43.33) # Function Call
dispstudinfo1(102,"PT",13.53) # Function Call
print("*50)
print("\tSNO\tNAME\tMARKS\tCourse")
print("*50)
dispstudinfo2(101,"DS",53.33) # Function Call
dispstudinfo2(102,"TQ",13.53) # Function Call
dispstudinfo2(100,"RS",33.33) # Function Call
dispstudinfo2(103,"GR",43.53) # Function Call
print("*50)

```

===== 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

=>When we specify Keyword arguments before Possitional Arguments in Function Calls(s) then we get SyntaxError: positional argument follows keyword argument

#Program for demonstrating Keyword Args

#KeywordArgsEx1.py

def disp(a,b,c,d):

print("\t{}\t{}\t{}\t{}".format(a,b,c,d))

main program

print("*50)

print("\tA\tB\tC\tD")

print("*50)

disp(10,20,30,40) # Function Call with Possitional Args

disp(d=40,c=30,b=20,a=10) # Function Call with Keyword Args

disp(10,20,d=40,c=30) # Function Call with Possitional Args and Keyword Args

#disp(b=20,a=10,30,40)--Invalid Function Call

disp(10,d=40,b=20,c=30) # Function Call with Possitional Args and Keyword Args

print("*50)

#Program for demonstrating Keyword Args

#KeywordArgsEx2.py

def dispstudinfo(sno,sname,marks,crs="PYTHON",city="INDIA"):

print("\t{}\t{}\t{}\t{}\t{}".format(sno,sname,marks,crs,city))

#main program

print("*50)

print("\tSNO\tNAME\tMARKS\tCourse\tCity")

print("*50)

dispstudinfo(100,"SAIRAM",44.44) # Function call with Pos Args

dispstudinfo(sname="JANI",marks=34.44,sno=200) # Function call with Kwd Args

dispstudinfo(crs="JAVA",sno=300,sname="DINESH",marks=22.22) # Function call with Kwd Args

dispstudinfo(400,marks=34.33,city="USA",sname="TRUMP",crs="HTML") # Function call withPos and Kwd Args

#dispstudinfo(city="USA",sname="TRUMP",crs="HTML",400,marks=34.33)--

SyntaxError: positional argument follows keyword argument

print("*50)

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

=>When we have family of multiple function calls with Variable number of values / arguments then with normal python programming, we must define mutiple function

definitions. This process leads to more development time. To overcome this process, we must use the concept of Variable length Parameters .

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

Syntax for function definition with Variables Length Parameters:

```
def functionname(list of formal params, *param1,param2=value) :
```

=>Here *param1 is called Variable Length parameter and it can hold any number of argument values (or) variable number of argument values and *param1 type is <class,'tuple'>

=>Rule:- The *param1 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 demonstarting Variable length arguments-- This Program will not execute as it is.

```
#VarLenArgsEx1.py
def disp(a,b,c,d,e,f): # Function Def-1
    print("{}\t{}\t{}\t{}\t{}.".format(a,b,c,d,e,f))
def disp(a,b,c,d,e): # Function Def-2
    print("{}\t{}\t{}\t{}\t{}.".format(a,b,c,d,e))
def disp(a,b,c,d): # Function Def-3
    print("{}\t{}\t{}\t{}.".format(a,b,c,d))
def disp(a,b,c): # Function Def-4
    print("{}\t{}\t{}.".format(a,b,c))
def disp(a,b): # Function Def-5
    print("{}\t{}.".format(a,b))
def disp(a): # Function Def-6
    print("{}.".format(a))
def disp(): # Function Def-7
    print("Empty")
```

```
#main program
print("-----")
disp(10,20,30,40,50,60) # Function Call-1
disp(10,20,30,40,50) # Function Call-2
disp(10,20,30,40) # Function Call-3
disp(10,20,30) # Function Call-4
disp(10,20) # Function Call-5
```

```

disp(10) # Function Call-6
disp() # Function Call-7
print("-----")
-----
#Program for demonstarting Variable length arguments--This Program will not
execute as it is.
#VarLenArgsEx2.py
print("-----")
def disp(a,b,c,d,e,f): # Function Def-1
    print("{}\t{}\t{}\t{}\t{}\t{}".format(a,b,c,d,e,f))

disp(10,20,30,40,50,60) # Function Call-1
#-----
def disp(a,b,c,d,e): # Function Def-2
    print("{}\t{}\t{}\t{}\t{}".format(a,b,c,d,e))

disp(10,20,30,40,50) # Function Call-2
#-----
def disp(a,b,c,d): # Function Def-3
    print("{}\t{}\t{}\t{}".format(a,b,c,d))

disp(10,20,30,40) # Function Call-3
#-----
def disp(a,b,c): # Function Def-4
    print("{}\t{}\t{}".format(a,b,c))

disp(10,20,30) # Function Call-4
#-----
def disp(a,b): # Function Def-5
    print("{}\t{}".format(a,b))

disp(10,20) # Function Call-5
#-----
def disp(a): # Function Def-6
    print("{} ".format(a))

disp(10) # Function Call-6
#-----
def disp(): # Function Def-7
    print("Empty")

disp() # Function Call-7
print("-----")
#Program for demonstarting Variable length arguments
#PureVarLenArgsEx2.py
def disp(*k): # Here *k is called Variable Length Parameter and its type is tuple
    for val in k:
        print("{} ".format(val),end=" ")
    print()

```

```

#main program
print("-----")
disp(10,20,30,40,50,60) # Function Call-1
disp(10,20,30,40,50) # Function Call-2
disp(10,20,30,40) # Function Call-3
disp(10,20,30) # Function Call-4
disp(10,20) # Function Call-5
disp(10) # Function Call-6
disp("KVR") # Function Call-7
print("-----")

-----
```

```

#Program for demonstarting Variable length arguments
#PureVarLenArgsEx3.py
def findsum(sno,sname, *vals):
    print("-----")
    print("Student Number:{}".format(sno))
    print("Student Name:{}".format(sname))
    print("-----")
    s=0
    for v in vals:
        print("\t{}".format(v),end=" ")
        s=s+v
    print()
    print("\tsum=%0.2f" %s)
    print("-----")
```

```

#main program
findsum(10,"Jani",100,200,300,400)
findsum(20,"Sai",12.3,45.6,4.5)
findsum(30,"Teja",12,24,13)
findsum(40,"KVR",40,50)
```

```

#Program for demonstarting Variable length arguments
#PureVarLenArgsEx4.py
def findsum(sno,sname, *vals,city="HYD"):
    print("-----")
    print("Student Number:{}".format(sno))
    print("Student Name:{}".format(sname))
    print("Living City:{}".format(city))
    print("-----")
    s=0
    for v in vals:
        print("\t{}".format(v),end=" ")
        s=s+v
    print()
    print("\tsum=%0.2f" %s)
    print("-----")
```

```
#main program
findsum(10,"Jani",100,200,300,400)
findsum(20,"Sai",12.3,45.6,4.5)
findsum(30,"Teja",12,24,13)
findsum(40,"KVR",40,50)
#findsum(50,"TRUMP",city="USA",-10,-20,-30,-40,-50)---error
#findsum(50,"TRUMP",-10,-20,-30,-40,-50,"USA")---error
findsum(50,"TRUMP",-10,-20,-30,-40,-50,city="USA")
#findsum(-10,-20,-30,-40,-50,sno=60,sname="JoE",city="USA")----Error
```

===== 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 astrisk (** param) and the formal parameter with double astrisk symbol is called Keyword Variable length Parameters and whose purpose is to hold / store any number of (Key,Value) coming from similar function calls and whose type is <class,'dict'>.

Syntax for function definition with Keyword Variables Length Parameters:

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

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

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

Final Syntax for defining a Function

```
def funcname(PosFormal parms, *Varlenparams, default params,
**kwdvarlenparams):
```

#program for demonstrating Keyword Var length arguments--This Program will not execute as it is
#KwdVarLenArgsEx1.py

```
def dispvalues(sno,sname,cls,percent): # Function Def-1
    print("\t{}\t{}\t{}\t{}".format(sno,sname,cls,percent))
```

```

def dispvalues(eno,ename,dsg): # Function def-2
    print("\t{}\t{}\t{}".format(eno,ename,dsg))

def dispvalues(cid,cname,hobby1,hobby2, hobby3):# Function def-3
    print("\t{}\t{}\t{}\t{}\t{}".format(cid,cname,hobby1,hobby2,hobby3))

def dispvalues(tno,tname,subject): # Function Def-4
    print("\t{}\t{}\t{}".format(tno,tname,subject))

#main program
dispvalues(sno=10,sname="Rossum",cls="X",percent=9.8) # Function call-1
dispvalues(eno=111,ename="Travis",dsg="Scientist") # Function call-2
dispvalues(cid=222,cname="ABC",hobby1="Eating",hobby2="Sleeping",
hobby3="chatting") # # Function call-3
dispvalues(tno=444,tname="KVR",subject="PYTHON") # Function call-4
-----
```

#program for demonstrating Keyword Var length arguments--
#KwdVarLenArgsEx2.py

```

def dispvalues(sno,sname,cls,percent): # Function Def-1
    print("\t{}\t{}\t{}\t{}".format(sno,sname,cls,percent))

dispvalues(sno=10,sname="Rossum",cls="X",percent=9.8) # Function call-1
#-----
def dispvalues(eno,ename,dsg): # Function def-2
    print("\t{}\t{}\t{}".format(eno,ename,dsg))

dispvalues(eno=111,ename="Travis",dsg="Scientist") # Function call-2
#-----
def dispvalues(cid,cname,hobby1,hobby2, hobby3):# Function def-3
    print("\t{}\t{}\t{}\t{}\t{}".format(cid,cname,hobby1,hobby2,hobby3))
dispvalues(cid=222,cname="ABC",hobby1="Eating",hobby2="Sleeping",
hobby3="chatting") # # Function call-3
#-----
def dispvalues(tno,tname,subject): # Function Def-4
    print("\t{}\t{}\t{}".format(tno,tname,subject))

dispvalues(tno=444,tname="KVR",subject="PYTHON") # Function call-4
-----
```

#program for demonstrating Keyword Var length arguments
#PureKwdVarLenArgsEx1.py

```

def dispvalues(**k): # here **k is called Kwd Var Length args and its type is dict
    print(k,type(k))
    print("-----")
```

#main program
dispvalues(sno=10,sname="Rossum",cls="X",percent=9.8) # Function call-1

```

dispvalues(eno=111,ename="Travis",dsg="Scientist") # Function call-2
dispvalues(cid=222,cname="ABC",hobby1="Eating",hobby2="Sleeping",
hobby3="chatting") # # Function call-3
dispvalues(tno=444,tname="KVR",subject="PYTHON") # Function call-4
-----
#program for demonstrating Keyword Var length arguments
#PureKwdVarLenArgsEx2.py

def dispvalues(**k): # here **k is called Kwd Var Length args and its type is dict
    print("-----")
    for kn,kv in k.items():
        print("\t{}--->{}".format(kn,kv))
    print("-----")
#main program
dispvalues(sno=10,sname="Rossum",cls="X",percent=9.8) # Function call-1
dispvalues(eno=111,ename="Travis",dsg="Scientist") # Function call-2
dispvalues(cid=222,cname="ABC",hobby1="Eating",hobby2="Sleeping",
hobby3="chatting") # # Function call-3
dispvalues(tno=444,tname="KVR",subject="PYTHON") # Function call-4
-----
#program for demonstrating Keyword Var length arguments
#PureKwdVarLenArgsEx3.py
def findtotalmarks(sno,sname,cls,city="HYD",**subjects):

    print("-----")
    print("\tStudent Number:{}".format(sno))
    print("\tStudent Name:{}".format(sname))
    print("\tStudent Class:{}".format(cls))
    print("\tStudent City:{}".format(city))
    print("-----")
    tm=0

    print("\tSubjectName\tSubjectMarks")
    print("-----")
    for sn,sm in subjects.items():
        print("\t{}\t\t{}".format(sn,sm))
        tm=tm+sm
    print("-----")
    print("\tTotal marks={}".format(tm))
    print("-----")

#main program
findtotalmarks(10,"Umesh","X",Hindi=56,English=70,Maths=60,Science=66,Social=55)
findtotalmarks(20,"Ramesh","XII",Maths=70,Physics=59,Chemstry=55)
findtotalmarks(30,"Rakesh","B.Tech(CSE)",OS=55,C=50,DBMS=45,DAA=30,DS=35,CPP=45)
findtotalmarks(40,"Rossum","Research")

```

```
findtotalmarks(50,"TRUMP","POLITICS",city="USA",politics=50)
findtotalmarks(60,"PUTIN","WAR",politics=20,city="RSA")
```

```
#program for demonstrating Keyword Var length arguments
#PureKwdVarLenArgsEx4.py
def findtotalmarks(sno,sname,cls,*vals, city="HYD",**subjects):
    print("-----")
    print("\tStudent Number:{}".format(sno))
    print("\tStudent Name:{}".format(sname))
    print("\tStudent Class:{}".format(cls))
    print("\tStudent City:{}".format(city))
    print("-----")
    print("Var args values")
    for val in vals:
        print("\t{}".format(val))
    print("-----")
    tm=0
    print("\tSubjectName\tSubjectMarks")
    print("-----")
    for sn,sm in subjects.items():
        print("\t{}\t{}".format(sn,sm))
        tm=tm+sm
    print("-----")
    print("\tTotal marks={}".format(tm))
    print("-----")

#main program
findtotalmarks(10,"Umesh","X",10,20,30,40,Hindi=56,English=70,Maths=60,Science=66,Social=55)
findtotalmarks(20,"Ramesh","XII",1.2,2.3,3.4,Maths=70,Physics=59,Chemstry=55)
findtotalmarks(30,"Rakesh","B.Tech(CSE)",-10,-20,OS=55,C=50,DBMS=45,DAA=30,DS=35,CPP=45)
findtotalmarks(40,"Rossum","Research",100,200)
findtotalmarks(50,"TRUMP","POLITICS",1000,2000,3000,city="USA",politics=50)
findtotalmarks(60,"PUTIN","WAR",105,205,305, politics=20,city="RSA")
findtotalmarks(70,"KVR","Teaching",city="AP")
```

===== Local variables and Global Variables =====

Local Variables

=>The Variables used inside of Function Body are called Local Variables.

=>The Purpose of Local Variables is that "To Store the Temporary results".

Syntax:

```
def functionname(list of formal Params if any):
    -----
        var1=val1
        var2=Val2
    -----
        var-n=val-n
    -----
```

=>here var1,var2..var-n are called Local Variables.

Global Variables

=>Global variables are those which are common values for different function calls.

=>In Other words, if the Value is common for all Different Function Calls then such type

of values must be taken as Global Variables.

=>To access the values of Global Variables then They Must be defined Before Function

Calls only otherwise we get NameError.

Syntax:

```
var1=val1
var2=val2
-----
var-n=val-n

def fun1():
-----
def fun1():
-----
def fun-n():
```

Here Var1, Var2..var-n are called Global variables and we can access those values inside of Fun1(), fun2()....fun-n().

#Program demonstarting local and global variables

#GlobalLocalVarEx1.py

lang="PYTHON" # Global variables are common values for Different Function Calls

def learnML():

 sub1="ML" # Here sub1 is local Variable

 print("To develop '{}' Applications,we use '{}' Programming".format(sub1,lang))

 # print(sub2,sub3)--We can't accesss--error bcoz sub2 and sub3 are local

variables in other Functions

```

def learnAI():
    sub2="AI" # Here sub2 is local Variable
    print("To develop '{}' Applications,we use '{}' Programming
Lang".format(sub2,lang))
    #print(sub1,sub3)--We can't accesss--error bcoz sub1 and sub3 are local
variables in other Functions
def learnDL():
    sub3="DL" # Here sub3 is local Variable
    print("To develop '{}' Applications,we use '{}' Programming
Lang".format(sub3,lang))
    #print(sub1,sub2)--We can't accesss--error bcoz sub1 and sub2 are local
variables in other Functions
#main program
learnML()
learnAI()
learnDL()
-----
```

```

#Program demonstarting local and global variables
#GlobalLocalVarEx2.py
def learnML():
    sub1="ML"
    print("To develop '{}' Applications,we use '{}' Programming
Lang".format(sub1,lang))

def learnAI():
    sub2="AI"
    print("To develop '{}' Applications,we use '{}' Programming
Lang".format(sub2,lang))

def learnDL():
    sub3="DL"
    print("To develop '{}' Applications,we use '{}' Programming
Lang".format(sub3,lang))

#main program
lang="PYTHON" # Global variables are common values for Different Function Calls
learnML()
learnAI()
learnDL()
-----
```

```

#Program demonstarting local and global variables
#GlobalLocalVarEx3.py
def learnML():
    sub1="ML"
    print("To develop '{}' Applications,we use '{}' Programming
Lang".format(sub1,lang))

def learnAI():
```

```

    sub2="AI"
    print("To develop '{}' Applications,we use '{}' Programming
Lang".format(sub2,lang))

lang="PYTHON" # Global variables are common values for Different Function Calls

def learnDL():
    sub3="DL"
    print("To develop '{}' Applications,we use '{}' Programming
Lang".format(sub3,lang))

#main program
learnML()
learnAI()
learnDL()

```

```

#Program demonstarting local and global variables
#GlobalLocalVarEx4.py
def learnML():
    sub1="ML"
    print("To develop '{}' Applications,we use '{}' Programming
Lang".format(sub1,lang))

def learnAI():
    sub2="AI"
    print("To develop '{}' Applications,we use '{}' Programming
Lang".format(sub2,lang))

def learnDL():
    sub3="DL"
    print("To develop '{}' Applications,we use '{}' Programming
Lang".format(sub3,lang))

#main program
#learnML()--can't access 'lang' bcoz it was defined after its function call.
lang="PYTHON" # Global variables are common values for Different Function Calls
learnAI()
learnDL()

```

```

#Program demonstarting local and global variables
#GlobalLocalVarEx5.py
def learnML():
    sub1="ML"
    print("To develop '{}' Applications,we use '{}' Programming
Lang".format(sub1,lang))

```

```

def learnAI():
    sub2="AI"
    print("To develop '{}' Applications,we use '{}' Programming
Lang".format(sub2,lang))

def learnDL():
    sub3="DL"
    print("To develop '{}' Applications,we use '{}' Programming
Lang".format(sub3,lang))

#main program
#learnML()--can't access 'lang' bcoz it was defined after its function call.
#learnAI()----can't access 'lang' bcoz it was defined after its function call.
#learnDL()---can't access 'lang' bcoz it was defined after its function call.
lang="PYTHON" # Global variables are common values for Different Function Calls

```

===== global key word =====

=>When we want MODIFY the GLOBAL VARIABLE values in side of function defintion then global variable names must be preceded with 'global' keyword otherwise we get "UnboundLocalError: local variable names referenced before assignment"

Syntax:

```

-----  

var1=val1  

var2=val2  

var-n=val-n # var1,var2...var-n are called global variable names.  

-----  

def fun1():  

    -----  

        global var1,var2...var-n  

        # Modify var1,var2....var-n  

    -----  

def fun2():  

    -----  

        global var1,var2...var-n  

        # Modify var1,var2....var-n

```

```

#Program for demonstrating global variables
#GlobalKwdEx1.py
a=10 # global variable
def incr1():
    global a
    a=a+1
def incr2():
    global a

```

```

a=a*2

#main program
print("Val of a in main program before incr1={}".format(a)) # 10
incr1()
print("Val of a in main program after incr1={}".format(a)) # 11
incr2()
print("Val of a in main program after incr2={}".format(a)) # 22
-----

#Program for demonstrating global variables
#GlobalKwdEx3.py
a=1
b=2 # here a and b are called Global variables
def update1():
    global a,b
    a=a+1
    b=b+1

def update2():
    global a,b
    a=a*2
    b=b*2
#main program
print("main Program before update1()-->a={}\\tb={}".format(a,b)) # a=1 b=2
update1()
print("main Program after update1()-->a={}\\tb={}".format(a,b)) # a=2 b=3
update2()
print("main Program after update2()-->a={}\\tb={}".format(a,b)) # a=4 b= 6
-----

#Program for demonstrating globals()
#In This Program global Variable Names are Different from local Variable Names
and we get Result
#non-globalsfunex1.py
a=10
b=20
c=30
d=40 # Here a,b,c,d are called global variables
def operations():
    x=100
    y=200
    z=300
    k=400 # Here x,y,z,k are called Local var
    res=a+b+c+d+x+y+z+k
    print("result=",res)
#main program
operations()

#Program for demonstrating globals()
#In This Program global Variable Names and local Variable Names are Same
#globalsfunex1.py

```

```

a=10
b=20
c=30
d=40 # Here a,b,c,d are called global variables
def operations():
    a=100
    b=200
    c=300
    d=400 # Here a,b,c,d are called Local var
    res=a+b+c+d+globals()['a']+globals().get('b')+globals()['c']+globals().get('d')
    print("result=",res)
#main program
operations()

-----
#Program for demonstrating globals()
#globalsfunex2.py
a=10
b=20 # here a dna b are called global Varaibles
def operations():
    a=100
    b=200 # here a dn b are called local variables
    d=globals() # here 'd' is an object of type dict
    print("-----")
    print("Global var names and values")
    print("-----")
    for gvn,gvv in d.items():
        print("\t{}-->{}".format(gvn,gvv))
    print("-----")
    print("Way-1: Programmer-Defined Global variable names and values")
    print("-----")
    print("Val of a--global=",d['a'])
    print("Val of b--global=",d['b'])
    print("-----")
    print("Way-2: Programmer-Defined Global variable names and values")
    print("-----")
    print("Val of a--global=",d.get('a'))
    print("Val of b--global=",d.get('b'))
    print("-----")
    print("Way-3: Programmer-Defined Global variable names and values")
    print("-----")
    print("Val of a--global=",globals().get('a'))
    print("Val of b--global=",globals().get('b'))
    print("-----")
    print("Way-4: Programmer-Defined Global variable names and values")
    print("-----")
    print("Val of a--global=",globals()['a'])
    print("Val of b--global=",globals()['b'])
    print("-----")
#main program
operations()

```

Anonymous Functions OR Lambda Functions

=>Anonymous Functions which does not contain any name explicitly.
 =>The Purpose of Anonymous Functions is that " To Perform Instant Operations".
 =>Instant Operations are Operations those which we can perform at that point of time only and no longer interested to re-use in other part of the application".
 =>To Develop / Define Anonymous Functions, we use a keyword lambda and hence Anonymous Functions are also called Lambda Functions.
 =>Anonymous Functions contains Single Executable Statement only (But not Multiple statements)
 =>Anonymous Functions automatically / Implicitly returns the value(No Need to use return statement).

Syntax:

varname=lambda params-list : Expression

Explanation:

=>Varname represents an object of <class, 'function'> and we can Varname as Function Call and Indirect name of Anonymous Function
 =>lambda is keyword used for defining Anonymous Functions.
 =>params-list represents list formal parameter used for storing the values coming from function call.
 =>Expression represents Single Executable Statement and it performs Instant Operation and whose result returns automatically / Implicitly(No Need to use return statement).

Q) Define a function addition of two numbers

By using Normal Function

```
def addop(a,b):
    c=a+b
    return c
```

```
#main program
res=addop(10,20)
print(res) # 30
```

By using Anonymous Function

```
sumop=lambda a,b: a+b
```

```
#main program
res=sumop(10,20)
print("Result=",res)
```

```
#program for adding two numbers by using Functions
#AnonymousFunEx1.py
def addop(a,b): # Normal Function Def
    c=a+b
    return c
sumop=lambda a,b: a+b # Anonymous Function Def...
```

```

#main program
res=addop(10,20) # Normal Function call
print("type of addop=",type(addop))
print("Result=",res)
print("-----")
print("type of sumop=",type(sumop)) # type of sumop= <class 'function'>
result=sumop(100,200) # Anonymous Function Call
print("Result=",result)

-----
#program for cal different ares of Figures by using Anonymous Function with Menu
driven
#AnonymousFunEx2.py
def menu():
    print("*****")
    print("\tAreas of Figures")
    print("*****")
    print("\t1.Circle")
    print("\t2.Rectangle")
    print("\t3.Square")
    print("\t4.Exit")
    print("*****")

#anonymous Function for cal Area of Circle
circlearea=lambda r: 3.14*r**2
#anonymous Function for cal Area of Rectangle
rectarea=lambda l,b: l*b
#anonymous Function for cal Area of Sqaure
squarearea=lambda s: s**2
#main program
while(True):
    menu()
    ch=int(input("Enter Ur Choice:"))
    match(ch):
        case 1:
            r=float(input("Enter Radius:"))
            ca=circlearea(r) # anonymous Function call
            print("Area of Circle={}".format(ca))
        case 2:
            l=float(input("Enter Length:"))
            b=float(input("Enter Breadth:"))
            ra=rectarea(l,b) # anonymous Function call
            print("Area of Rectangle={}".format(ra))
        case 3:
            s=float(input("Enter Side:"))
            sa=squarearea(s) # anonymous Function call
            print("Area of Square={}".format(sa))
        case 4:
            print("Thx for using Program")
            break
        case _:

```

```

        print("Ur Selection of Operation is Wrong--try again")
-----
#program for biggest and smallet of Three numbers by using Anonymous Function
#AnonymousFunEx3.py
big=lambda a,b,c: 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 VALS ARE EQUAL"
small=lambda a,b,c: 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 VALS ARE EQUAL"
#main program
x1=float(input("Enter First Value:"))
x2=float(input("Enter Second Value:"))
x3=float(input("Enter Third Value:"))
bv=big(x1,x2,x3)
sv=small(x1,x2,x3)
print("Big({},{},{}]={}.".format(x1,x2,x3,bv))
print("Small({},{},{}]={}.".format(x1,x2,x3,sv))
-----
#Program for domstrating tuple comprehension
#DictComprehenEx1.py
tpl=[2,3,5,8,9,12,-25]
d=dict( [ (val, val**2)      for val in tpl ] )
print("-----")
print("\tNumbers\tSquares")
print("-----")
for n,sv in d.items():
    print("\t{}\t{}".format(n,sv))
print("-----")
-----
#Program for domstrating tuple comprehension
#DictComprehenEx2.py
tpl=[2,3,-5,6,5,8,9,-6,12,-25,-4,-23]
d=dict( [ (val, val**2)      for val in tpl if val<0] )
print("-----")
print("\tNumbers\tSquares")
print("-----")
for n,sv in d.items():
    print("\t{}\t{}".format(n,sv))
print("-----")
-----
#Program for donstrating List comprehension
#ListComprehenEx1.py
lst=[10,21,34,51,53,68,67,59,100,26]
evenlist=[ val for val in lst  if (val%2==0)  ]
oddlist=[val for val in lst  if (val%2!=0)]
print(type(lst),type(evenlist),type(oddlist))
print("-----")
print("Given List:{}.".format(lst))

```

```

print("Even List={}".format(evenlist))
print("Odd List:{}\n".format(oddlist))
print("-----")
-----  

#Program for demonstrating Set comprehension
#SetComprehenEx2.py
sts={10,21,34,51,53,68,67,59,100,26}
evenlist={ val for val in sts if (val%2==0) }
oddlist={val for val in sts if (val%2!=0)}
print(type(sts),type(evenlist),type(oddlist))
print("-----")
print("Given List:{}\n".format(sts))
print("Even List={}\n".format(evenlist))
print("Odd List:{}\n".format(oddlist))
print("-----")
-----  

#Program for demonstrating tuple comprehension
#TupleComprehenEx1.py
tpl=(10,21,34,51,53,68,67,59,100,26)
el=( val for val in tpl if (val%2==0) ) # here el is an object of <class 'generator'>
ol=(val for val in tpl if (val%2!=0) ) # here ol is an object of <class 'generator'>
print(type(tpl),type(el),type(ol))
print("-----")
print("Given List:{}\n".format(tpl))
print("Even List={}\n".format(tuple(el))) # Converting generator object into tuple object
print("Odd List:{}\n".format(tuple(ol)))
print("-----")
-----  

#NamesLengthEx1.py
print("Enter List of Names separated by # :")
nameslist=[str(names) for names in input().split("#")]
print("-----")
print("Names List=",nameslist)
print("-----")
print("Names of Student\tLengths")
print("-----")
dictobj=dict([(str(names),len(names)) for names in nameslist])
for names,lengths in dictobj.items():
    print("\t{}\t{}\n".format(names,lengths))
print("-----")
-----  

#NamesLengthEx2.py
print("Enter List of Names separated by # :")
dictobj=dict([(str(names),len(names)) for names in input().split("#")])
print("-----")
print("Names of Student\tLengths")
print("-----")
for names,lengths in dictobj.items():
    print("\t{}\t{}\n".format(names,lengths))
print("-----")
-----
```

```
#ReadValuesCompreEx1.py
print("Enter List of values separated by space:")
lst=[ int(val) for val in input().split()]
print("Content of lst=",lst)
print("-----")
print("Enter List of Names Separated by - ")
nameslist=[str(names) for names in input().split("-")]
print("names list=",nameslist)
```

```
#ReadvaluesComprehenTech.py
#Program reading list of values from KBD
print("Enter List of Values separated by space:")
lst=[float(val) for val in input().split() if int(val)>0]
print("Content of list:{}".format(lst))
```

===== Special Functions in Python =====

=>In Python programming, we have 3 Special Functions. They are

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

===== 1) filter(): =====

=>filter() is used for "Filtering out some elements from list of elements by applying to function".

=>Syntax:- varname=filter(functionName, Iterable_object)

Explanation:

=>here 'varname' is an object of type <class,'filter'> and we can convert into any 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 filter() is that " Each Value of Iterable object sends to Function Name. If the function return True then the element will be filtered. if the Function returns False then that element will be neglected/not filtered ". This process will be continued until all elements of Iterable object completed.

```
#Program for Demonstarating filter()
#FilterEx1.py
def posval(n):
    if(n>0):
        return True
    else:
        return False
```

```

def negval(n):
    if(n<0):
        return True
    else:
        return False

#main program
lst=[10,20,-30,-56,57,34,-67,0,23,-68]
filtobj1=filter(posval,lst) # Here filtobj is of type <class, filter>
filtobj2=filter(negval,lst)
pslist=list(filtobj1)
nslist=tuple(filtobj2)
print("-----")
print("Given Elements:",lst)
print("Positive Elements=",pslist)
print("Negative Elements=",nslist)
print("-----")

-----#Program for Demonstarating filter()
#FilterEx2.py
def posval(n):
    if(n>0):
        return True
    else:
        return False

def negval(n):
    if(n<0):
        return True
    else:
        return False

#main program
print("Enter List of Elements separated by Space")
lst=[int(val) for val in input().split()]
pslist=list(filter(posval,lst))
nslist=tuple(filter(negval,lst))
print("-----")
print("Given Elements:",lst)
print("Positive Elements=",pslist)
print("Negative Elements=",nslist)
print("-----")

-----#Program for Demonstarating filter()
#FilterEx3.py
posval=lambda n: n>0 # Anoymous Functions
negval=lambda n: n<0 # Anoymous Functions
print("Enter List of Elements separated by Space")
lst=[int(val) for val in input().split()]

```

```

pslist=list(filter(posval,lst))

nslist=tuple(filter(negval,lst))
print("-----")
print("Given Elements:",lst)
print("Positive Elements=",pslist)
print("Negative Elements=",nslist)
print("-----")

#Program for Demonstarating filter()
#FilterEx4.py
print("Enter List of Elements separated by Space")
lst=[int(val) for val in input().split()]
pslist=list(filter(lambda k: k>0 , lst))
nslist=tuple(filter(lambda k: k<0,lst))
print("-----")
print("Given Elements:",lst)
print("Positive Elements=",pslist)
print("Negative Elements=",nslist)
print("-----")

#FilterEx5.py
def decide(name):
    if(len(name)>=3) and (len(name)<=5):
        if ( ('a' in name.lower()) or ('e' in name.lower()) or ('i' in name.lower()) or
('o' in name.lower()) or ('u' in name.lower()) ):
            return True
        else:
            return False
    else:
        return False

#main program
print("Enter List of Names separated by space")
nameslist=[str(names) for names in input().split()] # [Rossum Tarvis Kinney
Rakesh apple Python ant]
resultnames=list(filter(decide,nameslist))
print(resultnames)

-----


#FilterEx6.py
def decide(name):
    if(3<=len(name)<=5) and ('a' in name.lower() or 'e' in name.lower() or 'i' in
name.lower() or 'o' in name.lower() or 'u' in name.lower() ) :
        return True
    else:
        return False

#main program

```

```

print("Enter List of Names separated by space")
nameslist=[str(names) for names in input().split()] # [Rossum Tarvis Kinney
Rakesh apple Pytn ant]
resultnames=list(filter(decide,nameslist))
print(resultnames)

```

```

#FilterEx7.py
print("Enter List of Names separated by space")
nameslist=[str(names) for names in input().split()] # [Rossum Tarvis Kinney
Rakesh apple Pytn ant]
resultnames=list(filter(lambda name: (3<=len(name)<=5) and ('a' in name.lower() or
'e' in name.lower() or 'i' in name.lower() or 'o' in name.lower() or 'u' in name.lower() ),nameslist))
print(resultnames)

```

=====

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 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 demonstrating map()

#MapEx1.py

```

def square(n): # Normal Function
    res=n**2
    return res
#main program
lst=[12,13,19,2,-4,6,16]
mapobj=map(square,lst)
print("Type of mapobj=",type(mapobj)) # <class 'map'>
sqlist=list(mapobj) # Converting mapobj into list type
print("Original List=",lst)
print("Square List=",sqlist)

```

#Program for demonstrating map()

#MapEx2.py

```

square=lambda k: k**2
cube=lambda k: k**3

```

```
#main program
print("Enter List of Values separated by space:")
lst=[int(val) for val in input().split()]
sqlist=list(map(square,lst))
cclist=list(map(cube,lst))
print("=====")
print("Given Number\tSquare Number\tCube Numbers")
print("=====")
for n,sn,cn in zip(lst,sqlist,cclist):
    print("\t{}\t{}\t{}".format(n,sn,cn))
print("=====")
```

#Program for demonstrating map()

```
#MapEx3.py
print("Enter List of Salaries of Employee separated by space:")
oldsallst=[int(val) for val in input().split()]
newsallist=list(map(lambda sal:sal+sal*10/100 , oldsallst))
print("=====")
print("Old Sal List\tNew Sal List")
print("=====")
for osl,nsl in zip(oldsallst,newsallist):
    print("\t{}\t{}".format(osl,nsl))
print("=====")
```

#Program for demonstrating map()

```
#MapEx4.py
lst1=[10,20,30,40,50]
lst2=[100,200,300,400,500]
addedlist=list(map(lambda x,y:x+y, lst1,lst2))
print("-----")
print("\tLst1\tLst2\tLst1+Lst2")
print("-----")
for x,y,z in zip(lst1,lst2,addedlist):
    print("\t{}\t{}\t{}".format(x,y,z))
print("-----")
```

#Program for demonstrating map()

```
#MapEx5.py
print("Enter List of Values separated by space for First List:")
lst1=[int(val) for val in input().split()]
print("Enter List of Values separated by space for Second List:")
lst2=[int(val) for val in input().split()]
addedlist=list(map(lambda x,y:x+y, lst1,lst2))
print("-----")
print("\tLst1\tLst2\tLst1+Lst2")
```

```

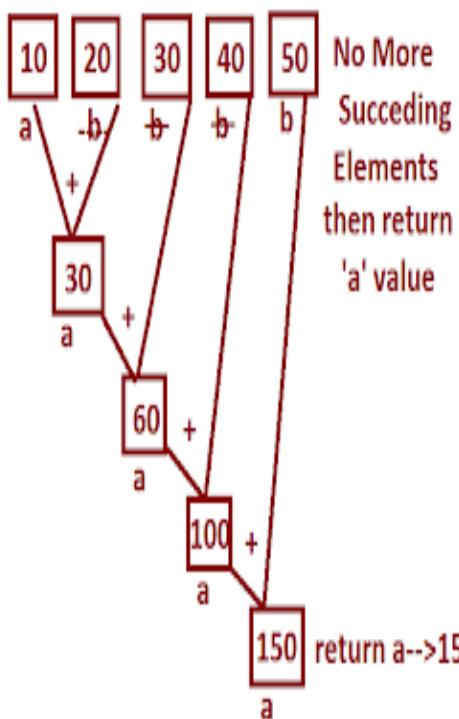
print("-----")
for x,y,z in zip(lst1,lst2,addedlist):
    print("\t{}\t{}\t{}".format(x,y,z))
print("-----")

```

Diagram--Internal flow of reduce()

consider $\text{lst}=[10,20,30,40,50]$

Q) Find sum of lst values by using reduce()



```

import functools
def addop(a,b):
    return(a+b)

```

```

lst=[10,20,30,40,50]
obj=functools.reduce(addop,lst)
print(obj)---150

```

=>Every Function belongs to MODULE.
=>The default module in Python is "builtins"
=>reduce() belongs to 'functools' module

```

lst=[10,20,30,40,50]
res=functools.reduce(lambda a,b:a+b , lst)
print(res)--->150

```

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(lambda 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 demonstrating reduce()

#reduceex1.py

```
import functools
print("\nEnter List of values separated by space for finding their sum:")
lst=[int(val) for val in input().split()]
print("-----")
print("List of Values:{}".format(lst))
sum=functools.reduce(lambda x,y:x+y,lst)
print("sum({})={}".format(lst,sum))
print("-----")
```

#Program for demonstrating reduce()

#reduceex2.py

```
import functools
def sumop(x,y):
    return(x+y)

#main program
print("\nEnter List of values separated by space for finding their sum:")
lst=[int(val) for val in input().split()]
print("-----")
print("List of Values:{}".format(lst))
sum=functools.reduce(sumop,lst)
print("sum({})={}".format(lst,sum))
print("-----")
```

#Program for demonstrating reduce()

#reduceex3.py

```
import functools
def kvrjoin(x,y):
    return(x+" "+y)

#main program
print("\nEnter List of Words separated by space :")
lst=[val for val in input().split()]
print("-----")
print("List of Words:{}".format(lst))
strresult=functools.reduce(kvrjoin,lst)
```

```

print("str cat result: {}".format(strresult))
print("-----")
-----#Program for demonstrating reduce()
#reduceex4.py
import functools
print("\nEnter List of Words separated by space :")
lst=[val for val in input().split()]
print("-----")
print("List of Words:{}".format(lst))
strresult=functools.reduce(lambda x,y: x+" "+y,lst)
print("str cat result: {}".format(strresult))
print("str cat result in reverse: {}".format(strresult[::-1]))
print("-----")
-----#Program for demonstrating reduce()
#reduceex5.py
import functools as kvr
print("\nEnter List of values separated by space for finding max :")
lst=[int(val) for val in input().split() # [10,20,1,13,4,,24]
maxv=kvr.reduce(lambda x,y: x if x>y else y, lst)
minv=kvr.reduce(lambda a,b: a if a<b else b, lst)
print("-----")
print("Max({})={}".format(lst,maxv))
print("Min({})={}".format(lst,minv))
print("-----")
-----#FilterMapreduceEx.py
import functools
print("Enter List of salaries separated by space:")
sallist=[int(sal) for sal in input().split() if 1000>=int(sal)>=0]
print("-----")
print("Given Salary List:{}".format(sallist))
print("-----")
sal_0_500=list(filter(lambda sal:sal>=0 and sal<=500, sallist))
sal_501_1000=list(filter(lambda sal:sal>=501 and sal<=1000, sallist))
hikesal_0_500=list(map(lambda sal:sal+sal*10/100,sal_0_500))
hikesal_501_1000=list(map(lambda sal:sal+sal*20/100,sal_501_1000))
totsal0_500=functools.reduce(lambda x,y:x+y,hikesal_0_500)
totsal501_1000=functools.reduce(lambda x,y:x+y,hikesal_501_1000)
print("-"*50)
print("\tOld Sal0-500\t\tNew Sal0-500")
print("-"*50)
for osl,nsl in zip(sal_0_500,hikesal_0_500):
    print("\t{}\t\t\t{}".format(osl,nsl))
print("\tTOAL SALARY-0-500=",totsal0_500)
print("=*50)
print("\tOld Sal501-1000\t\tNew Sal501-1000")
print("-"*50)
for osl,nsl in zip(sal_501_1000,hikesal_501_1000):

```

Modules in Python--2 days

Index

- =>Purpose of Modules
 - =>Definition of Module
 - =>Types of Modules
 - =>Steps for development of Programmer-Defined Modules
 - =>Number of approaches to Re-Use Modules
 - a) By using import statement
 - b) By using from ... import statement
 - =>Programming Examples

Modules in Python

- =>We know that Functions are used for "Performing Certain Operations and Provides Code Re-Usability within the same Program but not able to provide Code Re-Usability across the programs.".
 - =>The Purpose of Modules Concept is that "To Re-use the functions, global variables and Class Names" from One Program to another Program provided Both The Programs present in Same Folder.

=>Definition of Module

A Module is a collection of Functions, Global Variable Names and Class Names.

Types of Modules

=>In Python Programming, we have Two Types of Modules. They are

1. Pre-Defined Modules
2. Programmer OR User OR Custom Defined Module

1. Pre-Defined Modules

=>These Moudles are already defined by Python Lang Developers and Available in

Python Software and Used all Python Lang Programmers and for dealing with Universal Requirements.

Examples: functools,sys,calendar,re

=>Out-of Many Pre-defined Modules, By default One of the pre-defined module called "builtins" imported to all python programs and It is called Default imported python Module.

2. Programmer OR User OR Custom Defined Module

=>These Moudles are developed by Python Programmers and available in Python Project and Used by Other Members of Same Project for dealing with Common Requirements.

Examples: aop,MathsInfo,icici

===== 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.

=>Hence Every Python File 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-311.pyc ".

Examples:

| | |
|---------------------------|-------------------|
| __pycache__ | <-----Folder Name |
| aop.cpython-311.pyc | <-----Module Name |
| mathsinfo.cpython-311.pyc | <-----Module Name |
| icici.cpython-311.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 unique names

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

=>This syntax imports multiple modules and aliased with another unique names

Example: `import icici as i, aop as a , mathsinfo as m`

=>Hence after importing module name(s) by using "import statement", all the variable names, Function names and class names 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,addr, simpleint
```

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 Unique 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.

#aop.py--File Name and acts as Module Name

```
def addop(a,b):
    c=a+b
    print("sum({},{})={}".format(a,b,c))

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

def mulop(a,b):
    c=a*b
    print("mul({},{})={}".format(a,b,c))
```

#icici.py----File Name and Module Name

```
bname="ICICI"
addr="HYD" # Here bname and addr are called Global variables
def simpleint(): # Function Name
    p=float(input("Enter Principle Amount:"))
    t=float(input("Enter Time:"))
    r=float(input("Enter Rate of interest:"))
    #cal si and totamt
    si=(p*t*r)/100
    totamt=p+si
    #display the result
    print("****50)
    print("Principle Amount={}".format(p))
    print("Time={}".format(t))
    print("Rate of interest={}".format(r))
    print("-"*50)
    print("Simple Interest={}".format(si))
    print("TOTAL AMOUNT TO PAY={}".format(totamt))
    print("****50)
```

#MathsInfo.py---File Name and acts as module name

PI=3.14

E=2.71 # here PI and E are called Global Variables

```
#SE1.py---->Other Programmer
import aop
aop.addop(10,20) # Function Call
```

```
#SE2.py---Other team Mem using
import MathsInfo
print("Val of Pi=",MathsInfo.PI)
print("Val of E=",MathsInfo.E)
```

```
#SE3.py
import icici
print("Bank Name:{}".format(icici.bname))
print("Bank Address:{}".format(icici.addr))
icici.simpleint()
```

```
#importstmtsyntax1.py
import icici
import aop
print("Bank Name:",icici.bname)
print("Bank Address:",icici.addr)
icici.simpleint()
print("-----")
aop.addop(10,20)
```

```
#importstmtsyntax2.py
import icici,aop
print("Bank Name:",icici.bname)
print("Bank Address:",icici.addr)
icici.simpleint()
print("-----")
aop.addop(10,20)
aop.mulop(2,5)
```

```
#importstmtsyntax3.py
import icici as i
import aop as a
print("Bank Name:",i.bname)
print("Bank Address:",i.addr)
i.simpleint()
print("-----")
a.addop(10,20)
```

```
#importstmtsyntax4.py
import icici as i , aop as a
print("Bank Name:",i.bname)
print("Bank Address:",i.addr)
i.simpleint()
print("-----")
```

```
a.addop(10,20)
a.mulop(2,3)
a.subop(20,10)
```

```
#fromimportstmtsyntax1.py
from icici import bname,addr,simpleint
from aop import addop,subop
print("Bank Name:",bname)
print("Bank Address:",addr)
simpleint()
print("-----")
addop(10,20)
subop(3,4)
```

```
#fromimportstmtsyntax2.py
from icici import bname as bn,addr as ad,simpleint as si
from aop import addop as ap,subop as sp
print("Bank Name:",bn)
print("Bank Address:",ad)
si()
print("-----")
ap(10,20)
sp(3,4)
```

```
#fromimportstmtsyntax3.py
from icici import *
from aop import *
print("Bank Name:",bname)
print("Bank Address:",addr)
simpleint()
print("-----")
addop(10,20)
subop(3,4)
mulop(4,7)
```

```
#MulTable.py--File Name and Module name
def table(n):
    if(n<=0):
        print("{} 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))
        print("*50")
```

```
#MulDemoModule.py
from MulTable import table as t
t(int(input("Enter a Number for generating Mul table:")))
```

=====

realoding a modules in Python

=====

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

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

(OR)

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

=>Purpose / Situation:

=>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().

#shares.py---file and treated as module name

```
def sharesinfo():
    d={"Tech":19,"Pharma":11,"Auto":1,"Finance":00}
    return d
```

#main program

#sharesdemo.py

import shares

import time

import importlib

def disp(d):

print("-"*50)

print("\tShare Name\tValue")

print("-"*50)

for sn,sv in d.items():

print("\t{}\t\t{}".format(sn,sv))

else:

print("-"*50)

#main program

d=shares.sharesinfo()

disp(d)

time.sleep(15)

importlib.reload(shares) # relodaing previously imported module

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

disp(d)

#Program for demonstrating Re-Lodaing of Modules

```

#Shares.py---file name and module name
def sharesinfo():
    d={"IT":145,"Pharma":154,"Fin":614,"Auto":1130}
    return d

#OperMenu.py---File Name and Module Name
def menu():
    print("=*50)
    print("\tMiSc.Operations")
    print("=*50)
    print("\t1.Square of a Number:")
    print("\t2.Square Root of a Number:")
    print("\t3.Power of a Number:")
    print("\t4.Addition:")
    print("\t5.Substraction:")
    print("\t6.Multiplication:")
    print("\t7.Exit")
    print("=*50)

#Arithoper.py---File Name --Moudle
def addop(a,b):
    return (a+b)

def subop(a,b):
    return (a-b)

def mulop(a,b):
    return (a*b)

#mathsoper.py---File Name--Module
def square(n):
    return (n**2)

def sqrt(n):
    return(n**0.5)

def power(a,b):
    return (a**b)

#OperationsDemo.py---Main Program
from OperMenu import menu
import sys
from mathsoper import square,sqrt,power
from Arithoper import addop,subop,mulop
while(True):
    menu()
    ch=int(input("Enter Ur Choice:"))
    match(ch):
        case 1:
            n=float(input("Enter a number for calculating Square:"))

```

```

        res=square(n)
        print("Square({})={}".format(n,res))
    case 2:
        n=float(input("Enter a number for calculating Square Root:"))
        res=sqrt(n)
        print("Sqrt({})={}".format(n,res))

    case 3:
        m,n=float(input("Enter Base:")),float(input("Enter Power:"))
        res=power(m,n)
        print("power({},{})={}".format(m,n,res))
    case 4:
        a=float(input("Enter First Value for addition:"))
        b=float(input("Enter Second Value for addition:"))
        res=addop(a,b)
        print("sum({},{})={}".format(a,b,res))
    case 5:
        a=float(input("Enter First Value for Substraction:"))
        b=float(input("Enter Second Value for Substraction:"))
        res=subop(a,b)
        print("sub({},{})={}".format(a,b,res))
    case 6:
        a=float(input("Enter First Value for Multiplication:"))
        b=float(input("Enter Second Value for Multiplication:"))
        res=mulop(a,b)
        print("mul({},{})={}".format(a,b,res))
    case 7:
        print("Thx for using this Program")
        sys.exit()
    case _:
        print("Ur Selection of Operation is wrong-try again")

```

Case-1

```
=====
                    Arithemtic Operations
=====
```

- 1. Addition
- 2. Substration
- 3. Multiplications
- 4. Normal Division
- 5. Floor Division
- 6. Modulo Division
- 7. Expoentiation
- 8. Exit

Case-2

```
=====
                    Different Figures
=====
```

1. Circle Area
 2. Circle Perimter
 3. Square Area
 4. Square Perimeter
 5. Rectangle Area
 6. Rectange Perimter
 7. Triangle Area
 8. Triangle Perimeter($p=a+b+c$)
 9. Exit
-

Case-3

Temparature Conversions

1. Celcious to Fahrenheit
 2. Fahrenheit to Celcious
 3. Celcious to Kelvin ($\text{Kelvin} = \text{Celsius} + 273.15$)
 4. Kelvin to Celcious ($\text{Celsius}=\text{Kelvin}-273.15$)
 5. Fahrenheit to Kelvin($K = (F - 32) \times 5 / 9 + 273.15$)
 6. Kelvin to Fahrenheit ($F = (K - 273.15) \times 9 / 5 + 32$)
 7. Exit
-

File System

=>TempMenu.py----File Name and Module Name
 =>CelciousTOFahrenheit.py---File Name and Module Name
 =>FahrenheitToCelcious.py---File Name and Module Name
 =>CelciousToKelvin .py----File Name and Module Name
 =>KelvinToCelcious.py----File Name and Module Name
 =>FahrenheitToKelvin.py----File Name and Module Name
 =>KelvinToFahrenheit.py----File Name and Module Name
 =>TempCalculator.py---Main Program

Conversion Menu

1. Years to Months
2. Years to Days
3. Months to Years
4. Months to days
5. Days to Years
6. Days to Months
7. Exit

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 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 Enviromental Variables:

=>PYTHONPATH is one of the Enviromental Variable

=>Search for Enviromental 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

FAQ: What is Diff between Functions , Modules and Packages?

Function: To Perform Operation and Provides Code Re-Usability within the same Program-- But not Possible Re-Use across the Programs

Module: We Know that a Module is a collection of FUNCTIONS, GLOBAL VAR and CLASSES----- Advantage of Modules is that "To Provide Re-usability of FUNCTIONS, GLOBAL VAR and CLASSES one Program into another program / Program to program access provided Modules and Main program must present in Same Folder. But not Possible to Re-Use ACROSS the FOLDERS, ENVIRONMENTS, DRIVES, NETWORKS.

Package : A Package is a Collection of MODULES.

----- The Advtange of Packages is that To Provide Re-usability of FUNCTIONS, GLOBAL VAR and CLASSES of One Module of One Environment into another ENVIRONMENTS, DRIVES, NETWORKS.

=>Developing a package is nothing but Making Our Project as API

API: API is a collection of Packages.

---- Package is a Collection of Modules

A Module is a Collection FUNCTIONS, GLOBAL VAR and CLASSES

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

b) randint():

=>Syntax:- random.randint(start,stop)

=>This syntax generates any random value between start to stop. Here start and stop are inclusive.

Examples:

```
>>> print(random.randint(10,15))----10  
>>> print(random.randint(10,15))----13  
>>> print(random.randint(10,15))----14  
>>> print(random.randint(10,15))----11  
>>> print(random.randint(10,15))----15
```

```
#randintex.py  
import random  
for i in range(1,6):  
    print(random.randint(10,20))  
print("-----")
```

c) random()

=>Syntax:- random.random()

=>This syntax generates floating point random values between 0.0 and 1.0 (Exclusive)

Examples:

```
>>> import random  
>>> print(random.random())-----0.1623906138450063  
>>> print(random.random())-----0.15382209709271966  
>>> print(random.random())-----0.09542283007844476  
>>> print(random.random())-----0.6134301633766425
```

```
#randomex.py  
import random  
lst=[]  
for i in range(1,6):  
    lst.append("%0.2f" %random.random())  
print("-----")  
print("Content of lst={}".format(lst))
```

d) uniform()

Syntax:- random.uniform(start,stop)

=>This generates random floating point values from start to stop-1 values
=>The values of start and stop can both Integer or floating point values.

Examples:

```
>>> import random
>>> print(random.uniform(10,15))-----14.416746067678286
>>> print(random.uniform(10,15))---13.2420406264978
>>> print(random.uniform(10,15))----11.716110933506432
>>> print(random.uniform(10,15))-----10.703499588966528
>>> print(random.uniform(10,15))----11.306226559323017
>>> print(random.uniform(10.75,15.75))-----13.939787347170148
>>> print(random.uniform(10.75,15.75))---10.760428232717597
```

```
#uniformex.py
import random
lst=[]
for i in range(1,6):
    lst.append(float("%0.2f" %random.uniform(10,15.5)))
print("-----")
print("Content of lst={}".format(lst))
```

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))
```

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

```

```
I=[10,"Python","Rossum",34.56,True]
for i in range(1,6):
    r.shuffle(I)
    print("content of I=",I)
```

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))
```

```
#program for selecting random value from Iterable objects
#ChoiceEx2.py
import random as r
lst=[10,"Rossum","Python","Java",2+3j,True,20]
for i in range(1,6):
    print(r.choice(lst))
```

```
#program for selecting random value from Iterable objects
#ChoiceEx3.py
import random as r
alphas="ABCDEFGHIJKLMNPQRSTUVWXYZ"
digits="1234567890"
ss="~!@#$%^&*()_+{}|"
for i in range(1,6):
    print(r.choice(alphas),r.choice(digits),r.choice(ss),r.choice(alphas),r.choice(digits),r.choice(ss))
```

```
#program for selecting random value from Iterable objects
#ChoiceEx4.py
import random as r
alphas="ABCDEFGHIJKLMNPQRSTUVWXYZ"
sc=["09","08","07","12","13"]
```

```

stn="TS"
digits="0123456789"
for i in range(1,11):
    print(stn+r.choice(sc)+r.choice(alphas)+r.choice(alphas)+r.choice(digits)+r.choice(digits)+r.choice(digits)+r.choice(digits)+r.choice(digits))

-----
#program for generating random Integer numbers by using randint()
#RandIntEx1.py
import random as r
for i in range(5):
    print(r.randint(10,15))

-----
#program for generating random floating point values by using random()
#RandomEx1.py
import random as r
for i in range(1,5):
    print(r.random())
print("=====")
for i in range(1,5):
    print("%0.2f" %r.random())
print("=====")
for i in range(1,5):
    print(round(r.random(),3))
print("=====")

-----
#program for generating random numbers by using randrange()
#RandRangeEx1.py
import random as r
print("=====randrange(Value)=====")
for i in range(4):
    print(r.randrange(4))
print("=====randrange(start,stop)=====")
for i in range(4):
    print(r.randrange(10,20))
print("=====randrange(start,stop,step=====")
for i in range(4):
    print(r.randrange(10,20,2))

-----
#program for selecting random value from Iterable objects
#SampleEx1.py
import random as r
lst=[10,"Rossum","Python","Java",2+3j,True,20]
for i in range(1,6):
    print(r.sample(lst,2))

#
#program for selecting random value from Iterable objects
#SampleEx2.py
import random as r
alphas="ABCDEFGHIabcdefghijklmnopqrstuvwxyzJKLMNOPQRSTUVWXYZ"
digits="1234567890"

```

```

ss="~! @#$%^&*()_+{}|"
for i in range(1,6):
    for val in r.sample(alphas+digits+ss,6):
        print("{}".format(val),end="")
    print()
print("====")
for i in range(1,6):
    lst=r.sample(alphas+digits+ss,6)
    s=""
    s=s.join(lst)
    print("\t{}".format(s))
print("====")
-----  

#program for selecting random value from Iterable objects
#SampleEx3.py
import random as r
alphas="ABCDEFGHIJKLMONOPQRSTUVWXYZ"
sc=["09","08","07","12","13"]
stn="TS"
digits="0123456789"
for i in range(1,11):
    let=r.sample(alphas,k=2)
    ls=""
    ls=ls.join(let)
    #
    nums=r.sample(digits,k=4)
    ns=""
    ns=ns.join(nums)
    #
    scc=r.sample(sc,1)
    ss=""
    ss=ss.join(scc)
    print(stn+ss+ls+ns)
-----  

#program for shuffling the elements of Mutable-Iterabel object 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 shuffling the elements of Mutable-Iterabel object--shuffle
#ShuffleEx2.py
import random as r
s="MISSISSIPPI"
lst=list(s)
for i in range(0,len(s)):
    r.shuffle(lst)
    s1=""
    s1=s1.join(lst)
    print(s1)

```

```
#program for generating random floating point values by using uniform()
#UniformEx1.py
import random as r
for i in range(1,5):
    print(r.uniform(10,20))
print("====")
for i in range(1,5):
    print("%0.2f" %r.uniform(10,14.5))
print("====")
for i in range(1,5):
    print(round(r.uniform(10.3,12.34),3))
print("====")
```

=====

Exception Handling in Python-

=====

Index

=>Purpose of Exception Handling
 =>Definition of Exception
 =>Definition of Exception Handling
 =>Types of Errors
 a) Compile Time Errors
 b) Logical Errors
 c) Runtime Errors

=>Programming Examples

=>Types of Exceptions
 a) Pre-Defined Exceptions
 b) Programmer / User / Custom-Defined Exceptions

=>Keywords used in Handling the Exceptions

- 1) try
- 2) except
- 3) else
- 4) finally
- 5) raise

=>Syntax for handling the exceptions

=>Programming Examples

=>Various Forms of except block

=>Steps for Developing Programer-Defined Exceptions

=>Programming Examples

ATM Use Case Implementation with Exceptions

=====

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 need to choose a Programming Language. By using Programming Language, we Develop, Compile and Execute Various Programs.During This Process of Development, 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 (.py---->.pyc)

=>Compile Time Errors are occurring Due to Syntaxes are not Followed

=>Compile Time Errors are Solved by Programmers at Development time.

2. Logical Errors

=>Logical Errors are those which are occurring during Execution Process / Run Time.

=>Logical Errors are occurring Due to Wrong Representation of Logics.

=>Logical Errors always gives Wrong Results.

=>Logical Errors must be solved by Programmers during Development Time.

3. Runtime Errors (Implementation Errors)

=> Runtime Errors are those which are occurring during Execution Process / Run Time.

=>Runtime Errors are occurring Due to Wrong / Invalid Inputs entered by Application / End users.

=>Programmers must deals with Runtime Errors in project development for generating user-friendly error messages for making application Robust(Strong).

=>By Default Runtime Errors generates Technical Errors Messages which are understandable by Programmers but not by end-users. This Process is not recommended in Project Development. Hence Industry is Highly recommended to generate User-Friendly Error Messages by using Exception Handling.

Building Points for Dealing with Exception Handling

1. When Application User enters Invalid Input then we get Runtime Errors
(Invalid Input----->Runtime Errors)
2. By default Runtime Errors Generates Technical Error Messages
(Invalid Input----->Runtime Errors---->Tech Error Messages)
3. Definition of Exception : Every Runtime Error is called Exception.(Invalid Input---->Runtime Errors---->Exception)Hence Every Invalid Input gives Exception.

4. By default all exceptions generates Technical Errors Messages.
 5. Definition of Exception Handling:- The Process of Converting Technical Errors Messages into User-Friendly Error Messages is called Exception Handling.
 6. When an exception occurs in program, Internally PVM Performs 3 Steps.
 - a) PVM Terminates the Program Execution Abnormally.
 - b) PVM Flow Comes out-of Program Execution
 - c) By Default PVM Generates Technical Error Message.
 7. To Perform Step-(a), Step-(b) and Step-(c) , PVM creates an "object of appropriate exception class ".
 8. In General, If any exception occurs then PVM creates an object of appropriate exception class.
 9. Hence Every Exception in Python is Considered as Object of appropriate exception class".
-

NOTE:

Valid Inputs---->Program/App----->valid results

Invalid Inputs---->Program / App----->Runtime Error---->Exception--->TEM--->Exception handling--->UFEM

=====

Types of Exceptions in Python

=====

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

1. Pre-Defined OR Built-In Exceptions
 2. Programmer OR User OR Custom Defined Exceptions.
-

1. Pre-Defined OR Built-In Exceptions

=>These exceptions are already defined in Python Software and they are used by Python Programmers for dealing with Universal Problems.

=>Some of the Universal Problems are

1. Invalid Value Conversions (ValueError)
 2. Invalid Operations (TypeError)
 3. Invalid Indices (IndexError)
 4. Invalid Module names (ModuleNotFoundError)
 5. Invalid Attributes (AttributeError)
 6. Division by zero problems (ZeroDivisionError)..etc
-

2. Programmer OR User OR Custom Defined Exceptions.

=>These exceptions are Defined or Developed by Programmers and they are available in Python Project and they are used by other programmers of Same Project for dealing with Common Problems.

=>Some of the common problems

- a) Attempting to enter invalid pin in ATM Based Applications

- b) Attempting to enter Invalid User name and password
 - c) Attempting to withdraw Invalid amount from account..etc
-

===== Handling the exceptions in Python =====

=>Handling the exceptions in Python process is nothing but Converting Technical Error Messages into User-Friendly Error Messages.

=>To Handling the exceptions in Python, we have 5 keywords. They are

1. try
2. except
3. else
4. finally
5. raise

Syntax for handling the exceptions in python

```
try:
    Block of Statements
    generating exceptions
except exception-class-name-1:
    Block of Statements
    Generating User-Friendly
    Error Messages
except exception-class-name-2:
    Block of Statements
    Generating User-Friendly
    Error Messages
-----
except exception-class-name-n:
    Block of Statements
    Generating User-Friendly
    Error Messages
else:
    Block of Statements
    Generating results
finally:
    Block of Statements
    executes compulsorily
```

===== Explanation for the keywords used in Syntax of Handling Exception =====

1.try

=>It is the block in which we write block of statements generating exceptions. In otherwords what are all the statements generating exceptions, those statements must be written within try block and hence try block is called Exception monitoring block.

=>When an exception occurs in try block then PVM comes out of try block and executes appropriate except block.

=>After executing appropriate except block, PVM never goes to try block for executing rest of the statements in try block.

=>Every try block must be immediately followed by except block (Otherwise we get SyntaxError)

=>Every try block must contain atleast one except block . It is recommended to write multiple except blocks for generating User-Friendly error messages.

2.exception

=>It is the block in which we write block of statements generates User-Friendly Error Messages. In Otherwords except block suppreses Technical error messages and generates User-Freindly Error Messages and hence except block is called Exception Processing Block.

Note: Handling exception= try block + except block

=>except block will execute when there is an exception occurs in try block.

=>Even we write multiple except blocks , PVM executes Appropriate except block(Single Block)

=>The place for writing except block is that after try block and before else block.

3.else

=>It is the block in which we write block of statements will display results of the program and hence else block is called Result Generated Block.

=>else block will execute when there is no exception occurs in try block.

=>Writing else block is optional

=>The place of writing else block is that after except block and before finally block (if it present).

4.finally

=>It is the block, in which we write block of statements will relinquish (release / close / give-up/clean-up) the resources (Files, Database softwares) which are obtained in try block and finally block is called Resouces relinquishing Block.

=>finally block will execute compulsorily.

=>finally block is optional to write

=>The place of writing finally block is that after else block (if else block present)

Various forms of except blocks

=>we can use except block in Various Forms.

Form-1: With this form, we can handle one exception at a time.

```
try:  
-----  
-----  
except exception-class-name-1:  
-----  
-----  
except exception-class-name-2:  
-----  
-----  
-----  
except exception-class-name-n:  
-----  
-----
```

Example: Refer DivEx2.py Program

Form-2: With this form we can handle multiple specific exceptions with a single except block This type of facility is called Multi Exception handling block.

```
try:  
-----  
-----  
except ( exception-class-1,exception-class-2...exception-class-n ):  
-----  
-----  
Block of statements provides  
User-Friendly error messages  
for Multiple Specific Exceptions  
-----
```

Examples: Refer DivEx3.py

Form-3: With this form, we can handle one exception at a time with ALIAS NAME for capturing Technical error messages occurring due exception occurrence.

```
try:  
-----  
-----  
except exception-class-name-1 as alias name:  
-----  
-----  
print(alias name)  
-----  
except exception-class-name-2 as alias name:  
-----
```

```

-----
    print(alias name)
-----
except exception-class-name-n as alias name:
-----
    print(alias name)
-----

```

Examples: Refer DivEx4.py

Form-4: default except block--matches with all types of exceptions

```

try:
-----
-----
except :
-----
    print("OOOPs Some thing went wrong")
-----

```

=>Industry is not recommended to write single default except block bcoz it is giving common messages for all types of exceptions.

=>Hence Industry is highly recommended to write default except block at end of all types of specific except blocks.

Final Syntax:

```

try:
-----
-----
except exception-class-name-1:
-----
-----
except exception-class-name-2:
-----
-----
except exception-class-name-n:
-----
-----
except: # default except block
-----
-----
else:
-----
-----
finally:
-----

```

(OR)

try:

except (exception-class-1,exception-class-2...exception-class-n):

Block of statements provides
User-Friendly error messages
for Multiple Specific Exceptions

except: # default except block

else:

finally:

Examples: Refer DivEx5.py----Not recommended
 DivEx6.py----recommended

#program for cal division of two numbers by accepting input from Keyboard

#DivEx2.py

try:

```
s1=input("Enter First Value:")
s2=input("Enter Second Value:")
a=int(s1)
b=int(s2)
c=a/b
```

except ZeroDivisionError:

```
    print("Don't Enter Zero for Den...")
```

except ValueError:

```
    print("Don't enter alnums,strings and special symbols")
```

else:

```
    print("-----else--block-----")
    print("Div=",c)
    print("-----")
```

finally:

```
    print("I am from finally Block")
```

#program for cal division of two numbers by accepting input from Keyboard

#DivEx3.py

try:

```
s1=input("Enter First Value:")
s2=input("Enter Second Value:")
a=int(s1)
b=int(s2)
c=a/b
```

```

except (ZeroDivisionError,ValueError):
    print("Don't Enter Zero for Den...")
    print("Don't enter alnums,strings and special symbols")
else:
    print("-----else--block-----")
    print("Div=",c)
    print("-----")
finally:
    print("I am from finally Block")

```

#program for cal division of two numbers by accepting input from Keyboard

#DivEx4.py

```

try:
    s1=input("Enter First Value:")
    s2=input("Enter Second Value:")
    a=int(s1)
    b=int(s2)
    c=a/b
except ZeroDivisionError as z:
    print(z) # division by zero
except ValueError as kvr:
    print(kvr) # invalid literal for int() with base 10: '12x'
else:
    print("-----else--block-----")
    print("Div=",c)
    print("-----")
finally:
    print("I am from finally Block")

```

#program for cal division of two numbers by accepting input from Keyboard

#DivEx5.py--Not Recommeded

```

try:
    s1=input("Enter First Value:")
    s2=input("Enter Second Value:")
    a=int(s1)
    b=int(s2)
    c=a/b
except :
    print("oooops, some thing went wrong!!!!")
else:
    print("-----else--block-----")
    print("Div=",c)
    print("-----")
finally:
    print("I am from finally Block")

```

```
#program for cal division of two numbers by accepting input from Keyboard-
#DivEx6.py
try:
    s1=input("Enter First Value:")
    s2=input("Enter Second Value:")
    a=int(s1)
    b=int(s2)
    c=a/b
    s="PYTHON"
    print(s[10])
except ZeroDivisionError:
    print("Don't Enter Zero for Den...")
except ValueError:
    print("Don't enter alnums,strings and special symbols")
except IndexError:
    print("Plz check in the indexes")
except:
    print("Ooops some thing went wrong!!!")
else:
    print("-----else--block-----")
    print("Div=",c)
    print("-----")
finally:
    print("I am from finally Block")
```

===== Development of Programmer OR User OR Custom Defined Exceptions =====

=>These exceptions are Defined or Developed by Programmers and they are available in Python Project and they are used by other programmers of Same Project for dealing with Common Problems.

=>Some of the common problems are

- a) Attempting to enter invalid pin in ATM Based Applications
 - b) Attempting to enter Invalid User name and password
 - c) Attempting to withdraw Invalid amount from account..etc
-

Steps

- 1) Choose the Programmer-Defined Class
- 2) The Programmer-Defined Class Must Inherit From a Pre-defined Exception Super Class Called "Exception OR BaseException ". Hence Programmer-Defined Class acts Programmer-Defined Exception Class.

3) Save the above Development on Some file name with an extension.py and that file name acts as Module Name.

Examples: class PinError(Exception):pass

```
class LoginError(BaseException):pass
```

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

=>Here PinError, LoginError and InSuffFundError are called Programmer-Defined Exceptions.

#kvr.py-----File Name and Module Name

(1)

(2)

class KvrDivisionError(Exception):pass

#Phase-1----->Development of exceptions

#division.py---file name and module name

from kvr import KvrDivisionError

def divop(a,b):

if(b==0):

raise KvrDivisionError # Hit the exception when denominator is Zero

else:

return (a/b)

#Phase-2: Here we are Hitting the exception by using raise kwds

#DivDemo.py--main program

from division import divop

from kvr import KvrDivisionError

a=int(input("Enter value of a:"))

b=int(input("Enter value of b:"))

try:

res=divop(a,b) # Function call with Exception OR Result

except KvrDivisionError:

print("Don't enter zero for Den...")

else:

print("Result=",res)

finally:

print("I am frm finally Block")

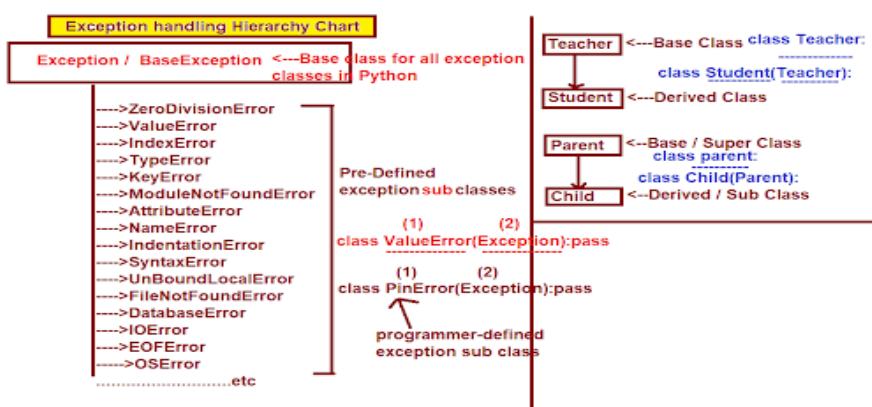
#Phase-3: Here are are handling the exceptions by using try and except kwds

FactExcept.py

```
class NegNumError(BaseException):pass
```

```
#FactcalHit.py
from FactExcept import NegNumError
def kvrfact(n):
    if(n<0):
        raise NegNumError
    else:
        f=1
        for i in range(1,n+1):
            f=f*i
        return f
```

```
#FactCalDemo.py
from FactcalHit import kvrfact
from FactExcept import NegNumError
try:
    n=int(input("Enter a Number for cal factorial:"))
    try:
        res=kvrfact(n) # Function call with result OR Exception
    except NegNumError:
        print("Don't enter -VE Number for cal Fact")
    else:
        print("Fact({})={}".format(n,res))
except ValueError:
    print("Don't enter alnums,strs and special symbols")
```



```
#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 Name and Module Name
from ATMExcept import DepositError,WithdrawError,InSuffFundError
bal=500.00 # here bal is Global Variable
def deposit():
    damt=float(input("Enter the Deposit Amt:")) # may raise ValueError in the case
    of damt is alnums/strs/symbols
    if(damt<=0):
        raise DepositError
    else:
        global bal
        bal=bal+damt
        print("Ur Account xxxxxxxx123 Credited with INR:{}".format(damt))
        print("Now Ur Account Bal INR:{}".format(bal))

def withdraw():
    global bal
    wamt=float(input("Enter the Withdraw Amt:")) # may raise ValueError in the
    case of wamt is alnums/strs/symbols
    if(wamt<=0):
        raise WithdrawError
    elif((wamt+500)>bal):
        raise InSuffFundError
    else:
        bal=bal-wamt
        print("Ur Account xxxxxxxx123 Debitted with INR:{}".format(wamt))
        print("Now Ur Account Bal INR:{}".format(bal))

def balenq():
    print("Ur Account Bal INR:{} ".format(bal))

#ATMCaseDemo.py---Main Program
from ATMMenu import menu
import sys
from ATMExcept import DepositError,WithdrawError,InSuffFundError
from ATMOperations import *
while(True):
    menu()
    try:
        ch=int(input("Enter Ur Choice:"))
        match(ch):
            case 1:
                try:
                    deposit()
                except ValueError:

```

```

        print("Don't enter Alnums,Strs and symbols for
Deposit Amt:")
    except DepositError:
        print("Don't try to Deposit -ve and 0 Amount")
    case 2:
        try:
            withdraw()
        except ValueError:
            print("Don't enter Alnums,Strs and symbols for
WithDarw Amt:")
        except WithdrawError:
            print("Don't try to Withdraw -ve and 0 Amount")
        except InSuffFundError:
            print("U Account don't have Sufficient Funds---"
Read Python Notes")
    case 3:
        balenq()
    case 4:
        print("Thx for this ATM--Visit Again")
        sys.exit()
    case _:
        print("Ur Selection of Operation is wrong--Try again")
except ValueError:
    print("Don't enter strs,alnums and symbols for Choice:")

```

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.

```
#program for demonstarting non-decorator
#non-decex1.py
```

```

def getval(): # Defined by KVR Programmer---Normal Function
    return int(input("Enter Ur Value:"))

def square(): # Sai Ram
    n=getval()
    print("Square({})={}".format(n,n**2))

def sqroot():                                # Deepak
    n=getval()
    print("SquareRoot({})={}".format(n,n**0.5))

def cube():                                    #Aparna
    n=getval()
    print("Cube({})={}".format(n,n**3))

def forth():                                   #Laxmi
    n=getval()
    print("ForthRoot({})={}".format(n,n**4))

#main program
square()
sqroot()
cube()
forth()

```

```

#program for demonstarting decorator
#decex1.py
def getval(): # Defined by KVR Programmer---Normal Function
    return int(input("Enter Ur Value:"))

def square(gv): # here square is called Decorator
    def operation(): # here operation is called Inner Function
        n=gv()
        res=n**2
        return res
    return operation

#main program
result=square(getval)
print("type result=",type(result))
r=result()
print("Square=",r)

```

```

#program for demonstarting decorator
#decex2.py
def square(kvr): # Here Square is a Decorator
    def operation(): # Inner Function
        n=kvr()
        r=n**2
        return r

```

```

        return operation
@square # here @Square is nothing calling Decorator
def getval(): # Defined by KVR Programmer---Normal Function
    return int(input("Enter Ur Value:"))
#main program
res=getval()
print("Square=",res)
-----
##program for demonstarting decorator
#decex3.py
def getval():
    return input("Enter a Word")
def decide(gv):
    def palin():
        word=gv()
        if word==word[::-1]:
            return "PALINDROME"
        else:
            return "NOT PALINDROME"
    return palin
#main program
result=decide(getval)
r=result()
print(r)
-----
##program for demonstarting decorator
#decex4.py
def Palindrome(hyd):
    def decide():
        w=hyd()
        res="PALINDROME" if w==w[::-1] else "NOT PALINDROME"
        return res
    return decide

@Palindrome
def getval():
    return input("Enter a Word:")

#main program
res=getval()
print("Given Word is :{}".format(res))
-----
##program for demonstarting decorator
#decex4.py

def Palindrome(hyd):
    def decide():
        w=hyd()
        res="PALINDROME" if w==w[::-1] else "NOT PALINDROME"

```

```

        return res
    return decide

@Palindrome
def getval():
    return input("Enter a Word:")

#main program
res=getval()
print("Given Word is :{}".format(res))

-----#Program accepting list of values and get Even and Odd Numbers
#EvenistEx.py
print("Enter List of Values Separated by Space:")
evenlist=[ int(val)  for val in input().split()  if int(val)%2==0]
print("Even List of Elements={}".format(evenlist))

-----#Program for demonstrating filter() ,map() and reduce()
#FilterMapReduceEx.py
import functools
print("Enter List of Employee Salaries ranges from 0 to 1000:")
empsallist=[int(sal) for sal in input().split() if ( 1000>=int(sal)>=0 ) ]
print("List Emp Salaries:{}".format(empsallist))
#-----Filter with 0 to 500-----
sal_0_500=list(filter(lambda sal: sal>=0 and sal<=500 , empsallist))
print("List of employee Salaries ranges from 0-500={}".format(sal_0_500))
#-----Filter with 501 to 1000-----
sal_501_1000=list(filter(lambda sal: 1000>=sal>=501 , empsallist))
print("List of employee Salaries ranges from 501-1000={}".format(sal_501_1000))
#-----Hike 10% for 0 to 500-----
hike0_500=list(map(lambda sal:sal+sal*10/100,sal_0_500))
#-----Hike 10% for 0 to 500-----
hike501_1000=list(map(lambda sal:sal+sal*20/100,sal_501_1000))

#-----Find Total Sal after hike of 10%-----
totalsal0_500=functools.reduce(lambda sal1,sal2 : sal1+sal2,hike0_500)
#-----Find Total Sal after hike of 10%-----
totalsal501_1000=functools.reduce(lambda sal1,sal2 : sal1+sal2,hike501_1000)
#-----Display Salaries of old 0--500 and hikes and total
print("*50")
print("Old Sal0-500\tHiked Sal 0-500")
print("*50")
for oldsal,newsal in zip(sal_0_500,hike0_500):
    print("\t{}\t{}".format(oldsal,newsal))
else:
    print("*50")
    print("Total Salary paid to 0-500 Hikes={}".format(totsal0_500))
    print("*50")
print("*50")
print("\nOld Sal501-1000\tHiked Sal 501-1000")

```

```

print("*50)
for oldsal,newsal in zip(sal_501_1000,hike501_1000):
    print("\t{}\t{}".format(oldsal,newsal))
else:
    print("*50)
    print("Total Salary paid to 501-1000 Hikes={}".format(totsal501_1000))
    print("*50)
print("TOTAL SALARY FOR ALL
EMPLOYEES={}".format(totsal0_500+totsal501_1000))
print("*50)

#ListCompreHenEx1.py
lst=[3,2,7,-4,5,12,19]
sqlist=[ val**2 for val in lst ]
print("-----")
print("Normal Value\tSquare Value")
print("-----")
for nv,sv in zip(lst,sqlist):
    print("\t{}\t{}".format(nv,sv))
print("-----")

#ListCompreHenEx2.py
lst=[3,-2,7,-4,5,-12,0,19]
poslist=[ val for val in lst if val>0      ]
neglist=[ val for val in lst if val<0      ]
print("Given List=",lst)
print("Positive Elements=",poslist)
print("Negative Elements=",neglist)

#Program accepting list of values and get Odd Numbers
#OddListEx.py
print("Enter List of Values Separated by Space:")
oddlist=[ int(val) for val in input().split() if int(val)%2!=0]
print("Odd List of Elements={}".format(oddlist))

#Program accepting list of words
#PalWordsEx1.py
import functools
print("Enter List of Words Separated by Comma:")
palwords=[word for word in input().split(",") if word==word[::-1] ]
print("List of Palindrome Words=",palwords)
#code obtaining Palindromes and their length
palwordslen=dict( [(word,len(word)) for word in palwords] )
print("-*50)
for word,wordlen in palwordslen.items():
    print("\t{}\t{}".format(word,wordlen))
print("-*50)
#Code for finding Max length Palindrome word
lengths=list(palwordslen.values())
biglen=functools.reduce(lambda x,y: x if x>y else y, lengths)

```

```

print("Highest Palindrome length=",biglen)
-----
#Program accepting list of words
#PalWordsEx2.py
import functools
print("Enter List of Words Separated by Comma:")
palwords=[word for word in input().split(",") if word==word[::-1] ]
print("List of Palindrome Words=",palwords)
#code obtaining Palindromes and their length
palwordslen=dict( [(word,len(word)) for word in palwords] )
print("-"*50)
for word,wordlen in palwordslen.items():
    print("\t{}\t{}".format(word,wordlen))
print("-"*50)
#Code for finding Max length Palindrome word
ml=max(palwordslen.values())
maxword=[ w for w,l in palwordslen.items() if ml==l]
if(len(maxword)==1):
    print("Max Length word:{} and Its Length={}".format(maxword[0],ml))
else:
    for word in maxword:
        print("Max Length word:{} and Its Length={}".format(word,ml))

```

```

#ReadingValues1.py
print("Enter List of values separed by space")
kdata=input()
x=kdata.split()
for val in x:
    print("\t{}".format(val))

```

```

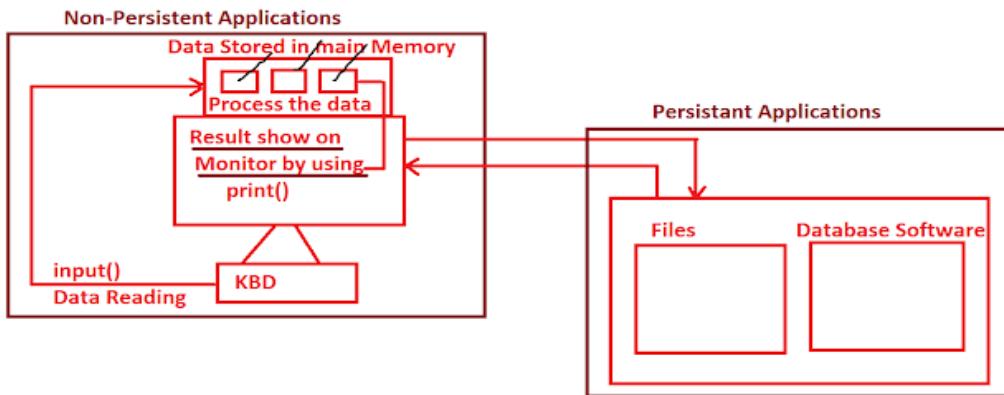
#ReadingValues2.py
print("Enter List of values separed by space")
lst=[ int(val) for val in input().split()] # [10, 20, 30, 40, 50, -23, -45, 45, 23, 45]
print("Given List=",lst)

```

```

#ReadingValues3.py
print("Enter List of values separed by comma")
lst={ int(val) for val in input().split(",")}
print("Given List=",lst)

```



Types of Application in Files

=>The purpose of Files in any programming language is that " To maintain Data Persistence".

=>The Process of storing the data permanently is called Data Persistence.

=>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(RAM) 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 RDBMS DataBase Softwares (Oracle, MySQL, PostgreSQL,

MongoDB, DB2, PostgreSQL,
SQL Server,SQLITE3..etc)

Data Persistence by Files of Python

Def. of File:

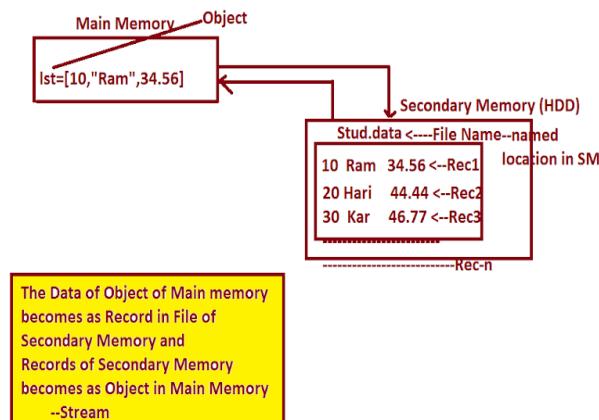
=>A File is a collection of Records.

=>Files Resides in Secondary Memory(HDD).

=>Technically, File Name is a named location in Secondary Memory.
 =>The purpose of Files is that "To get Data Persistency".
 =>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
-

Types of Files in Python

=>In Python Programming, we have Two Types of Files. They are

1. Text Files
 2. Binary Files
-

1. Text Files

=>Text Files are those which are containing Alphabets, Digits and Symbols.

=>Text Files are denoted by a Letter 't'

=>By Default Every File treated as Text File

=>Examples: .c, .cpp, .txt , .docx, .xlsx, .py...etc

2. Binary Files

=>Binary Files are those which are containing the data in the form of Binary

=>Binary Files are denoted by a Letter 'b'

=>Examples: All types of Image Files (.jpeg, .gif , .jpg)audio , video files

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 Exceptions where as Programmer makes the PVM to use raise keyword explicitly for Hitting or Generating Programmer-defined Exceptions.

=>Syntax-1:- if (Test Cond):
 raise <exception-class-name>

=>Syntax-2:- 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)
```

File Opening Modes in Python

=>The purpose of File Opening Modes in Python is that "In which 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 in Read Mode.

=>If we open the file name in "r" mode and if the file name does not exist then we get FileNotFoundError.

=>The default file mode in Python is "r" mode.

2. w

=>This Mode is used for Creating a New File and Open it in Write Mode.

=>If We Choose New File Then It will create a New File and Open it in Write Mode.

=>If We Open Existing File Then It will Open it in Write Mode and Existing data OVERLAPPED with NEW DATA.

3. a

=>This Mode is used for Creating a New File and Open it in Write Mode.

=>If We Choose New File Then It will create a New File and Open it in Write Mode.

=>If We Open Existing File Then It will Open it in Write Mode and Existing data APPENDED with NEW DATA.

4. r+

=>This Mode is Used for Opening the File in Read Mode.

=>If we open the file name in "r+" mode and if the file name does not exist then we get FileNotFoundError.

=>Once If we Open the File Name in "r+" Mode then we That file Opened in read Mode and First We can read the data from the file and later we can also Write the data to the file.

5. w+

=>This Mode is Used for Opening the Files in Write Mode

=>If We Choose New File Then It will create a New File and Open it in Write Mode.

=>If We Open Existing File Then It will Open it in Write Mode and Existing data OVERLAPPED with NEW DATA.

=>Once If we Open the File Name in "w+" Mode then we That file Opened in Write Mode and First We can Write the data to the file and later we can also Read the data from the file.

6. a+

=>This Mode is used for Creating a New File and Open it in Write Mode.

=>If We Choose New File Then It will create a New File and Open it in Write Mode.

=>If We Open Existing File Then It will Open it in Write Mode and Existing data APPENDED with NEW DATA.

=>Once If we Open the File Name in "a+" Mode then we That file Opened in Write Mode and First We can Write the data to the file and later we can also Read the data from the file.

7. x

=>This Mode is used for Creating a New File and Open it in Write Mode eXclusively and we can perform Write Operations.

=>If we Open existing file in "x" mode then we get FileNotFoundError.

8. x+

=>This Mode is used for Creating a New File and Open it in Write Mode eXclusively and we can perform Write Operations First and Later we can also Perform Read Operations.

=>If we Open existing file in "x+" mode then we get FileNotFoundError.

Number of approaches to Open the file

=>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 an Object and Whose Type is <class '_io.TextIOWrapper'>
=>open() is a pre-defined function present in builtins module and It is used for
Opening a File in Specified File Mode.

=>File Name Represents Name of the File

=>File Mode represents r,w,a,r+,w+,a+,x,x+

=>Hence Once Open the file name with open() then It Mandatory 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:

Block of Statements Performs File Operations

Other Statements in program

Explanation:

=>Here "with" and "as" are the Key words

=>open() is a pre-defined function present in builtins module and It is used for Opening a File in Specified File Mode.

=>File Name Represents Name of the File

=>File Mode represents r,w,a,r+,w+,a+,x,x+

=>Varname Represents an Object and Whose Type is <class 'io.TextIOWrapper'>

=>The execution Process of "with open(---) as " is that "As Long as PVM present in side of with open(---) as Indentation then File Name is actually Available and once

side of with open(...) as Indentation then File Name is actively Available and once PVM comes of with open(...) as Indentation then File name closed Automatically and This Facility is Called Auto-Closeability of File". No Need to use file by using close() manually.

#Program for Demonstrating Opening the file in Read Mode

#FileOpenEx1.py

try:

```
fp=open("stud.data")
```

```
    ip.open('stddata')
except FileNotFoundError:
```

```
print("File does not exist")
```

else:

```

print("File Opened in Read Mode Sucessfully")
print("Name of the file:",fp.name)
print("File Mode: ", fp.mode)
print("Is File Closed: ",fp.closed)
print("Is File Readable?: ",fp.readable())
print("Is File Writeable?: ", fp.writable())
finally:
    print("\nI am from finally Block:")
    fp.close()
    print("Is File Closed: ", fp.closed)
#Program for Demonstrating Opening the file in Write Mode
#FileOpenEx2.py
fp=open("stud.data","w")
print("File Opened in Write Mode Sucessfully")
print("Type of kvr=",type(fp))
print("Name of the file:",fp.name)
print("File Mode: ", fp.mode)
print("Is File Closed: ",fp.closed)
print("Is File Readable?: ",fp.readable())
print("Is File Writeable?: ", fp.writable())
-----
```

```

#Program for Demonstrating Opening the file in Write Mode
#FileOpenEx3.py
with open("stud.data","w") as fp:
    print("-----")
    print("File Opened in Write Mode Sucessfully")
    print("Type of kvr=",type(fp))
    print("Name of the file:",fp.name)
    print("File Mode: ", fp.mode)
    print("Is File Closed: ",fp.closed)
    print("Is File Readable?: ",fp.readable())
    print("Is File Writeable?: ", fp.writable())
    print("-----")
    print("\nI am out of with open() as Indentation")
    print("Is File Closed: ", fp.closed)
-----
```

```

#Program for Demonstrating Opening the file in Write Mode
#FileOpenEx4.py
with open("kvr.data","a+") as fp:
    print("-----")
    print("File Opened in Write Mode Sucessfully")
    print("Type of fpr=",type(fp))
    print("Name of the file:",fp.name)
    print("File Mode: ", fp.mode)
    print("Is File Closed: ",fp.closed)
    print("Is File Readable?: ",fp.readable())
    print("Is File Writeable?: ", fp.writable())
    print("-----")
    print("\nI am out of with open() as Indentation")
    print("Is File Closed: ", fp.closed)
```

#Program for Demonstrating Opening the file in Write Mode
#FileOpenEx5.py

```
try:
    with open("stud1.data","x") as fp:
        print("-----")
        print("File Opened in Write Mode Sucessfully")
        print("Type of kvr=",type(fp))
        print("Name of the file:",fp.name)
        print("File Mode: ", fp.mode)
        print("Is File Closed: ",fp.closed)
        print("Is File Readable?: ",fp.readable())
        print("Is File Writeable?: ", fp.writable())
        print("-----")
    print("\nI am out of with open() as Indentation")
    print("Is File Closed: ", fp.closed)
except FileExistsError:
    print("File Name alerady Exist")
```

#Program for Demonstrating Opening the file in Write Mode
#FileOpenEx6.py

```
try:
    with open("stud2.data","x+") as fp:
        print("-----")
        print("File Opened in Write Mode Sucessfully")
        print("Type of kvr=",type(fp))
        print("Name of the file:",fp.name)
        print("File Mode: ", fp.mode)
        print("Is File Closed: ",fp.closed)
        print("Is File Readable?: ",fp.readable())
        print("Is File Writeable?: ", fp.writable())
        print("-----")
    print("\nI am out of with open() as Indentation")
    print("Is File Closed: ", fp.closed)
except FileExistsError:
    print("File Name alerady Exist")-
```

===== Reading the data from the file =====

=>To read the data from file, we have TWO pre-defined Functions present in File Pointer. They are

1. `read()`
 2. `readlines()`
-

1. `read()`

=>**Syntax:** `varname=filepointer.read()`

=>This Function is used for Reading entire content of the file in the form of str and placed in varname.

2. **readlines()**

=>Syntax: `varname=filepointer.readlines()`

=>This Function is used for Reading entire content of the file in the form of list and placed in varname.

=>Here var name of type of list.

NOTE:=>`read()` and `readlines()` will read the data from the file in the form of Value by value.

===== Writing the data to the file =====

=>To write the data to the file, we have 2 pre-defined functions present in File Pointer object. They are

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

a) **write()**

=>Syntax: `filpointer.write(data)`

=>This Function is used for writing any type of data in the form `<class,'str'>`.

b) **writelines()**

=>Syntax: `filpointer.writelines(data)`

=>This Function is used for writing any Interable data directly to the file in the form `<class,'str'>`.

NOTE:

=>`write()` and `writelines()` will write the data to the file on the basis of Value by Value but not all at once..

===== Files OR Streams in Python =====

Index

=>Purpose of Files

=>Types of Applications

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

=>Definition of File

=>Definition of Stream

=>Types of Operations on Files

- a) Write Operation
- b) Read Operation

=>Types of Files

- a) Text Files
- b) Binary Files

=>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 "

=>Programming Examples

=>Pickling (Object Serialization) and Un-Pickling (Object De-Serialization)

=>Implementation of Pickling and Un-Pickling Operations

=>pickle module

=>Programming Examples

=>Working with CSV Files

=>csv module

=>Operations on CSV Files

- a) csv.Writer
- b) csv.Reader
- c) csv.DictWriter
- d) csv.DictReader

=>Programming Examples

=>Working with OS Based Oprations

=>os module

=>Programming Exampels

#Program for Reading the data from KBD and write to the file

#DynamicWriteFileOp.py

with open("hyd.info","a") as fp:

```
print("Enter the data from KBD(to terminate press '@'):")
while(True):
    kbddata=input()
    if(kbddata!=" @"):
        fp.write(kbddata+"\n")
    else:
        break
print("\nData written to tyhe file-verify")
```

#This Program copy the content of one file into another file

#FileCopyEx1.py

srcfile=input("Enter Source File:")

try:

```

with open(srcfile,"r") as rp:
    destfile=input("Enter Destination File:")
    with open(destfile,"w") as wp:
        srcfiledata=rp.read() # Reading the data from SRC File
        wp.write(srcfiledata) # Writing the data to the dest file
        print("File Copied--Verify")
except FileNotFoundError:
    print("File does not exist")

```

```
#program for counting number of lines, words and characters
#FileCountInfo.py
```

```
filename=input("Enter any file name:")
```

```
try:
```

```
    with open(filename,"r") as fp:
```

```
        filedata=fp.readlines()
```

```
        nl=0
```

```
        nw=0
```

```
        nc=0
```

```
        for line in filedata:
```

```
            nl=nl+1
```

```
            nw=nw+len(line.split())
```

```
            nc=nc+len(line)
```

```
        else:
```

```
            print("-----")
```

```
            print("Number of Lines:{} ".format(nl))
```

```
            print("Number of words={}".format(nw))
```

```
            print("Number of Chars={}".format(nc))
```

```
            print("-----")
```

```
except FileNotFoundError:
```

```
    print("File does not exist:")
```

```
#This Program demonstrates How to write the data to the file
```

```
#FileWriteEx1.py
```

```
with open("sample.data","w") as fp:
```

```
    fp.write(str(1000))
```

```
    fp.write("Travis")
```

```
    fp.write(str(44.44))
```

```
    fp.write(str(2+3))
```

```
    print("Data Written to the file")
```

```
#This Program demonstrates How to write the data to the file
```

```
#FileWriteEx2.py
```

```
x= {10:"Python",20:"Java",30:"R",40:"C++"}
```

```
with open("sampl1.data","a") as fp:
```

```
    fp.writelines(str(x)+"\n")
```

```
    print("Data Written to the file")
```

```
#FileWriteEx2.py
```

```
with open("addr1.data","a") as fp:
```

```

fp.write("James Gosling\n")
fp.write("FNO:13-14 HILL Side\n")
fp.write("Sun Micro System\n")
fp.write("USA-16\n")
print("Data Written to the file")

```

```

#program for Reading the data from File
#FileReadEx1.py
try:
    with open("addr1.data","r") as fp:
        filedata=fp.read()
        print("=====")
        print(filedata)
        print("=====")
except FileNotFoundError:
    print("File does not exist")

```

```

#ImageCopyEx1.py
try:
    with open("D:\\KVR-PYTHON-9AM\\FILES\\NOTES\\human.png","rb") as rp:
        with open("C:\\KVR\\abc.png","wb") as wp:
            imgdata=rp.read()
            wp.write(imgdata)
            print("Image Copied--verify")
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 an 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(bcoz of 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 of 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(bcoz of 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 transferring the entire 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.

#Program for Reading Student number,name and marks and save them in file by using Pickling Operation.

#StudPickEx1.py---Program-(A)

import pickle

def savestuddata():

 with open("studpick.data","ab") as fp:

 #accept the values from KBD

 sno=int(input("Enter Student Number:"))

 sname=input("Enter Student Name:")

 marks=float(input("Enter Student Marks:"))

 #create an empty list object and append student details

 l=list()

 l.append(sno)

 l.append(sname)

 l.append(marks)

 #save Iterable object to file by using dump()

 pickle.dump(l,fp)

 print("Student Record Saved Sucessfully in File:")

#main program

savestuddata()

#Program for Reading Student number,name and marks and save them in file by using Pickling Operation.

#StudPickEx2.py---Program-(A)--File Name and Module Name

import pickle

def savestuddata():

 with open("studpick.data","ab") as fp:

 while(True):

 #accept the values from KBD

 try:

 print("-----")

 sno=int(input("Enter Student Number:"))

 sname=input("Enter Student Name:")

 marks=float(input("Enter Student Marks:"))

 #create an empty list object and append student details

 l=list()

 l.append(sno)

 l.append(sname)

 l.append(marks)

 #save Iterable object to file by using dump()

 pickle.dump(l,fp)

 print("-----")

 print("Student Record Saved Sucessfully in File:")

 print("-----")

 ch=input("Do u want to insert another Record(yes/no):")

 if(ch.lower()=='no'):

 break

```

        except ValueError:
            print("Don't Enter alnums,strs and symbols for stno and
Marks")
#main program
savestuddata()
-----
```

```
#Program for Reading the records from file by using un-pickling concept
#StudUnPickEx1.py---Program--(B)
```

```

import pickle
def readrecords():
    try:
        with open("studpick.data","rb") as fp:
            print("****40)
            while(True):
                try:
                    obj=pickle.load(fp)
                    print(obj,type(obj))
                except EOFError:
                    print("****40)
                    break
    except FileNotFoundError:
        print("File does note exist")
```

```
#main program
readrecords()
```

```
#Program for Reading the records from file by using un-pickling concept
```

```
#StudUnPickEx2.py---Program--(B)
```

```

import pickle
def readrecords():
    try:
        with open("studpick.data","rb") as fp:
            print("****40)
            while(True):
                try:
                    obj=pickle.load(fp)
                    for val in obj:
                        print("{}".format(val),end="\t")
                    print()
                except EOFError:
                    print("****40)
                    break
    except FileNotFoundError:
        print("File does note exist")
```

```
#main program
readrecords()
```

```
#Program for Reading Student number,name and marks and save them in file by
using Pickling Operation.
#SPICK.py---Program-(A)--File Name and Module Name
import pickle
def savestuddata():
    with open("studpick.data","ab") as fp:
        while(True):
            #accept the values from KBD
            try:
                print("-----")
                sno=int(input("Enter Student Number:"))

                sname=input("Enter Student Name:")
                marks=float(input("Enter Student Marks:"))
                #create an empty list object and append student details
                l=list()
                l.append(sno)
                l.append(sname)
                l.append(marks)
                #save Iterable object to file by using dump()
                pickle.dump(l,fp)
                print("-----")
                print("Student Record Saved Sucessfully in File:")
                print("-----")
                ch=input("Do u want to insert another Record(yes/no):")
                if(ch.lower() == "no"):
                    break
            except ValueError:
                print("Don't Enter alnums,strs and symbols for stno and
Marks")
```

```
#Program for Reading the records from file by using un-pickling concept
#SUNPICK.py--File name and module name
import pickle
def readrecords():
    try:
        with open("studpick.data","rb") as fp:
            print("***40)
            while(True):
                try:
                    obj=pickle.load(fp)
                    for val in obj:
                        print("{}".format(val),end="\t")
                    print()
                except EOFError:
```

```

        print("****40)
        break
    except FileNotFoundError:
        print("File does note exist")

```

```

#PickUnPickMenu.py
import sys
from SPICK import savestuddata as sd
from SUNPICK import readrecords as rd
while(True):
    print("=====")
    print("\tPICKLE OPERATIONS")
    print("=====")
    print("\t1.Pickle")
    print("\t2.Un-Pickle")
    print("\t3.Exit")
    print("=====")
    ch=int(input("Enter Ur Choice:"))
    match (ch):
        case 1:
            sd()
        case 2:
            rd()
        case 3:
            print("Thx for using this Program")
            sys.exit()
        case _:
            print("Ur Selection of Operation is Wrong:")

```

===== 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 Using csv.writer class object
 - 2) By Using csv.DictWriter class object

=>There are 2 classes provided by this module for Reading the data from CSV File. They are

- 1) By using Using csv.reader class object
 - 2) By Using csv.DictReader class object
-

1) By using 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 method writes a single row at a time. Field row can be written using this method.

Syntax:- csvwriterobj.writerow(fields Row / Data Row)

2) writerows(): This method 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 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,fieldname="Hedare Names")**

| | | |
|-----|---------|-------|
| sno | sname | marks |
| 10 | Rossum | 33.33 |
| 20 | Travis | 44.44 |
| 30 | Ritche | 11.11 |
| 40 | Kinney | 55.44 |
| 50 | Gosling | 22.44 |

```
#Program for Reading the data from csv file
#csvreadex1.py
import csv
try:
    with open("stud.csv","r") as fp:
        csvr=csv.reader(fp) # Here csvr is an object of
<class, csv.reader>
        print("*50")
        for record in csvr:
            for val in record:
                print("{}".format(val),end="\t")
            print()
        print("*50")
except FileNotFoundError:
    print("File does not exist")
```

```
#Program for Creating and Writing the data to csv file
#csvwriteex1.py
import csv
hn=["empno","ename","sal","dsg"]
recs=[

[100,"Rossum",3.4,"Author"],
```

```

[200,"Travis",2.4,"Scientist"],
[300,"Kinney",6.4,"TL"],
[400,"KVR",0.0,"Teacher"],
[500,"Raj",1.4,"HRM"]
]
with open("emp.csv","a") as fp:
    csvwr=csv.writer(fp)
    print("Type of csvwr=",type(csvwr))
    csvwr.writerow(hn)
    csvwr.writerows(recs)
    print("\nCSV File Created and data written to the CSV File:")
-----
```

```

#csvwriteex2.py
import csv
record=[700,"Naveed",4.3,"SE"]
with open("emp.csv","a") as fp:
    cw=csv.writer(fp)
    cw.writerow(record)
    print("Employee Record Saved in CSV File--Verify")
-----
```

```

#csvreaderex2.py
import csv
with open("emp.csv","r") as fp:
    cr=csv.reader(fp)
    print("-----")
    for record in cr:
        for val in record:
            print("{}".format(val),end="\t")
        print()
    print("-----")
-----
```

```

#Program for Reading CSV file data in the for Dict
#CSVDictReaderEx1.py
import csv
with open("stud.csv","r") as fp:
    dr=csv.DictReader(fp) # Here dr is an object of <class 'csv.DictReader'>
    for record in dr:
        print("-----")
        for k,v in record.items():
            print("\t{}-->{}".format(k,v))
        print("-----")
-----
```

```

#Program for Reading CSV file data in the for Dict
#CSVDictReaderEx2.py
import csv
with open("emp.csv","r") as fp:
    dr=csv.DictReader(fp) # Here dr is an object of <class 'csv.DictReader'>
    for record in dr:
        print("-----")
        for k,v in record.items():
-----
```

```

        print("\t{}-->{}".format(k,v))
        print("-----")
#Program for Reading CSV file data in the for Dict
#CSVDictReaderEx3.py
import csv
with open("teacher.csv","r") as fp:
    dr=csv.DictReader(fp) # Here dr is an object of <class 'csv.DictReader'>
    for record in dr:
        print("-----")
        for k,v in record.items():
            print("\t{}-->{}".format(k,v))
        print("-----")
-----
#Program for Creating and Writing the dict data to csv file
#Dictcsvwriteex1.py
import csv
hn=["TNO","TNAME","SUB"]
recs=[
    {"TNO":100,"TNAME":"Rossum","SUB":"PYTHON"},
    {"TNO":200,"TNAME":"Travis","SUB":"NUMPY"},
    {"TNO":300,"TNAME":"RITCHE","SUB":"C"},
    {"TNO":400,"TNAME":"KINNEY","SUB":"Pandas"}
]
with open("teacher.csv","a") as fp:
    csvdw=csv.DictWriter(fp,fieldnames=hn)
    print("type of csvwr=",type(csvdw)) # <class 'csv.DictWriter'>
    csvdw.writeheader()
    csvdw.writerows(recs)
    print("Dict of Teacher data save in a CSV File--Verify")

```

| empno | ename | ``` | saldsg |
|-------|----------|-----|-----------|
| 100 | Rossum | 3.4 | Author |
| 200 | Travis | 2.4 | Scientist |
| 300 | Kinney | 6.4 | TL |
| 400 | KVR | 0 | Teacher |
| 500 | Raj | 1.4 | HRM |
| 600 | Sidartha | 6.3 | TL |
| 700 | Naveed | 4.3 | SE |

| sno | sname | marks |
|-----|--------|-------|
| 10 | Rossum | 33.33 |
| 20 | Travis | 44.44 |

| | | |
|----|---------|-------|
| 30 | Ritche | 11.11 |
| 40 | Kinney | 55.44 |
| 50 | Gosling | 22.44 |

| TNO | TNAME | SUB |
|-----|-------|-----|
|-----|-------|-----|

| | | |
|-----|--------|--------|
| 100 | Rossum | PYTHON |
| 200 | Travis | NUMPY |
| 300 | RITCHE | C |
| 400 | KINNEY | Pandas |

```
#Program for Reading the data from csv file
#non-csvread.py
```

```
try:
    with open("stud.csv","r") as fp:
        csvdata=fp.read()
        print(csvdata)
except FileNotFoundError:
    print("File does not exist")
```

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) Creating Folder / Directory. (`mkdir()`)
 - 2) Creating Folders Hierarchy. (`makedirs()`)
 - 3) Removing Folder / Directory. (`rmdir()`)
 - 4) Removing Folders Hierarchy. (`removedirs()`)
 - 5) Removing File Name from Folder(`remove()`)
 - 6) Renaming a Folder/File Name. (`rename()`)
 - 7) List the file names in folder (`listdir()`)
-

1) 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 `FileExistsError`

=>`mkdir()` can create only one folder at a time and if we try to create folderS 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")
```

2) 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")
```

3) 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
#rmdir.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 folder which are empty--check ur path")

```

4) Removing Folders Hierarchy. (removedirs())

=>For 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 \digi, \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 not exist")
except OSError:
    print("remove those folder which are empty-Check ur path of folder names")

```

5) 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")

```

6) 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 name
#RenameFolderEx.py
import os
try:
    os.rename("D:\KVR","D:\PYTHON")
    print("Folder Name renamed")
except FileNotFoundError:
    print("File does not exist")
```

7) 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 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")
```

```
#Program for list the file names in a folder
#ListFilesFilderEx.py
import os
try:
    filenames=os.listdir(".") # Listing the file of Current Working folder ( that
represents symbol . )
    print("-"*50)
    print("File Names:")
    print("-"*50)
    for filename in filenames:
        print("\t{}".format(filename))
    print("-"*50)
except FileNotFoundError:
```

```

print("Folder does not exist")
-----
#Program for list the file names in a folder
#ListFilesFilderEx1.py
import os
try:
    foldername=input("Enter Folder name for liosting the files:")
    filenames=os.listdir(foldername)
    print("-"*50)
    print("File Names in: {}".format(foldername))
    print("Number of Files={}".format(len(filenames)))
    print("-"*50)
    for filename in filenames:
        print("\t{}".format(filename))
    print("-"*50)
except FileNotFoundError:
    print("Folder does not exist")
-----
```

```

#Program for list the file names in a folder
#ListFilesFilderEx2.py
import os
try:
    foldername=input("Enter Folder name for liosting the files:")
    filenames=os.listdir(foldername)
    print("-"*50)
    print("File Names in: {}".format(foldername))
    print("Number of Files={}".format(len(filenames)))
    pyf=0
    print("-"*50)
    for filename in filenames:
        if(filename[-3:]==".py"):
            print("\t{}".format(filename))
            pyf=pyf+1
    print("-"*50)
    print("Number of Python Files={}".format(pyf))
except FileNotFoundError:
    print("Folder does not exist")
-----
```

```

#Program for creating a Filders Hierarchy--makedirs()
#mkdirsex.py
import os
try:
    os.makedirs("C:\APPLE\KIWI")
    print("Folders Hierarchy Created--Verify")
except FileExistsError:
    print("Folders Hierarchy already exist:")
-----
```

```

#Program for creating a Filder--mkdir()
#mkdirex.py
import os
```

```

try:
    os.mkdir("KVR1\PYTHON")
    print("Folder created--verify")
except FileExistsError:
    print("Folder already exist:")
except FileNotFoundError:
    print("Folder does not exist")
-----
#Program for deleting the folder name---rmdir()
#removedirsex.py
import os
try:
    os.removedirs("C:\INDIA\HYD\AMPT\PYTHON")
    print("Folder Removed Successfully")
except FileNotFoundError:
    print("Folder does not exist")
except OSError:
    print("Folder is not empty")
-----
#program for removing the file name
#RemoveFileNameEx.py
import os
try:
    os.remove("D:\\KVR-PYTHON-9AM\\FILES\\stud1.data")
    print("File removed sucessfully")
except FileNotFoundError:
    print("File does not exist")
-----
#Program for Renaming a File Or Folder Name--rename()
#renamefileex.py
import os
try:
    os.rename("D:\KVR-PYTHON-9AM\FILES\student2.data","D:\KVR-PYTHON-9AM\FILES\stud2.data")
    print("File name Renamed--verify")
except FileNotFoundError:
    print("File does not exist")
-----
#Program for Renaming a Folder Name--rename()
#renamefolderex.py
import os
try:
    os.rename("D:\KVR-PYTHON-9AM\TEST","D:\KVR-PYTHON-9AM\TESTING")
    print("Folder name Renamed--verify")
except FileNotFoundError:
    print("File does not exist")
-----
#Program for deleting the folder name---rmdir()
#rmdirrex.py

```

```

import os
try:
    os.rmdir("KVR")
    print("Folder Removed Successfully")
except FileNotFoundError:
    print("Folder does not exist")
except OSError:
    print("Folder is not empty")

```

Limitations of Files

- =>Files of any Lang Does no contain Security bcoz Files does not provide user name and password.(Files are un-Secured)
 - =>Files are unable to Store Large Vol of data and They can't Persist
 - =>Files Arch Differs from One Os to Another OS.(Files are OS Dependent)
 - =>Processing the data of Files is Very Complex bcoz Programmer must keep track of Indices
 - =>Extracting or Retrieving the data from Files is Complex bcoz Normal Files does not contains Col Names.
-

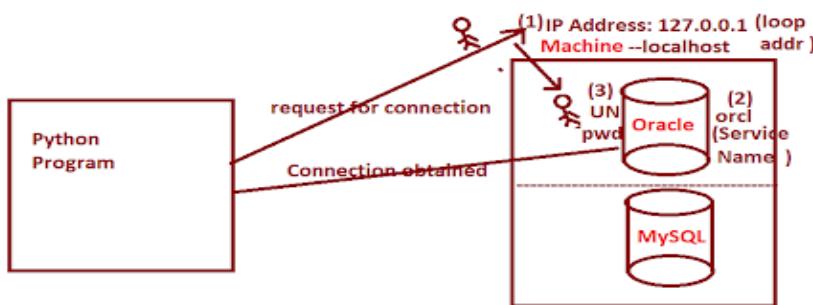
To Overcome these Limitations of Files, we can Achieve the Data Persistency by Using RDBMS Data base Softwares(Oracle, MySQL,DB2, MongoDB,Postgrey SQL, SQLITE3 ...etc)

```

#RandomAccessFileEx1.py
with open("addr1.data","r") as fp:
    fpindex=fp.tell()
    print("Initial Index of fp=",fpindex)
    filedata=fp.read(5)
    print("File data=",filedata)
    print("-----")
    fpindex = fp.tell()
    print("Now Index of fp=", fpindex)
    print("-----")
    filedata=fp.read(11)
    print("File data=", filedata)
    fpindex = fp.tell()
    print("Now Index of fp=", fpindex)
    print("=====")
    filedata = fp.read()
    print("File data=", filedata)
    fpindex = fp.tell()
    print("Now Index of fp=", fpindex)
    print("=====")
    filedata = fp.read()
    print("File data=", filedata)
    fp.seek(0)
    print("=====")
    filedata = fp.read(10)

```

```
print("File data=", filedata)
```



===== Python DataBase Communication (PDBC) =====

=>Even we achieved the Data Persistence by using Files, Files has the following Limitations.

1. Files of any language does not contain security bcoz Files are unable to provide security in the form of User Name and Password.
2. Files are unable to store large amount of data
3. File are differing from One OS to another OS (Files are OS depended)
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) and complex to Process data

=>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).

1. All RDBMS DataBase Softwares Provides Security bcoz RDBMS DataBase 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 records in the form of Tables with Column Names.
5. The Data Present in any RDBMS DataBase Softwares organized in the of Tables with Column Names makes the processing Easy.

=>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

Steps for Developing Python Program Communication with Oracle

Steps

1. import cx_Oracle module
 2. Every Python program must get the CONNECTION from Oracle Database
 3. Every Python program Must Create an object of Cursor
 4. Every Python Program must Design the Query, Place the Query in cursor object and execute
 5. Every Program must Process the Result of the Query which is coming from DatabaseSoftware.
 6. Every Python Program Closes the Connection.
-

Explanation

Step-1: 1. import cx_Oracle module

=>If a Python Program want to communicate with Oracle Database then we must import cx_Oracle module

=>Examples: import cx_Oracle

2. Every Python program must get the CONNECTION from Oracle Database

=>After importing cx_Oracle Module, Python Program must get the connection from Oracle Database.

=>If a Python program want to get the connection from Oracle database then we must use connect() of cx_Oracle module.

=>Syntax: varname=cx_Oracle.connect("Connection URL")

Here Connection URL Represents

"username/password@DNS/serviceid"

(OR)

Here Connection URL Represents
"username/password@IPAddress/serviceid"

Explanation

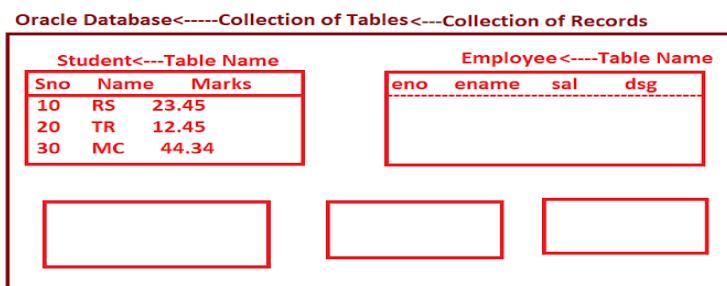
=>here varname is an object of Connection object
 =>Here username and password represents user name and password of Oracle Database
 =>DNS is nothing but name of the Physical Machine where Database Software Installed / resides The Default DNS of Every Computer is "localhost"
 =>IP Address is nothing but Address of the Physical Machine where Database Software Installed / resides. The Default IP ADDress of Every Computer is "127.0.0.1" (loop back address)
 =>ServiceID represents on which name Oracle database software is available in current machine.To Find Service-ID of Oracle Database, use the following Command at SQL Environment.

```
SQL> select * from global_name;
      GLOBAL_NAME
```

XE

Exmaple: con=cx_Oracle.connect("system/manager@localhost/XE")
 here con is an object of <class, cx_Oracle.Connection>

View about Oracle Database



Steps for Developing Python Program Communication with Oracle

Steps

1. import cx_Oracle module
2. Every Python program must get the CONNECTION from Oracle Database

3. Every Python program Must Create an object of Cursor
 4. Every Python Program must Design the Query, Place the Query in cursor object and execute
 5. Every Program must Process the Result of the Query which is coming from Database Software.
 6. Every Python Program Closes the Connection.
-

Explanation

Step-1: 1. import cx_Oracle module

=>If a Python Program want to communicate with Oracle Database then we must import cx_Oracle module

=>Examples: import cx_Oracle

2. Every Python program must get the CONNECTION from Oracle Database

=>After importing cx_Oracle Module, Python Program must get the connection from Oracle Database.

=>If a Python program want to get the connection from Oracle database then we must use connect() of cx_Oracle module.

=>Syntax: varname=cx_Oracle.connect("Connection URL")

Here Connection URL Represents "username/password@DNS/serviceid"

(OR)

Here Connection URL Represents "username/password@IPAddress/serviceid"

Explanation

=>here varname is an object of Connection object

=>Here username and password represents user name and password of Oracle Database

=>DNS is nothing but name of the Physical Machine where Database Software Installed / resides The Default DNS of Every Computer is "localhost"

=>IP Address is nothing but Address of the Physical Machine where Database Software Installed / resides. The Default IP ADDress of Every Computer is "127.0.0.1" (loop back address)

=>ServiceID represents on which name Oracle database software is available in current machine. To Find Service-ID of Oracle Database, use the following Command at SQL Environment.

```
SQL> select * from global_name;
      GLOBAL_NAME
```

XE

Exmaple: con=cx_Oracle.connect("system/manager@localhost/XE")
here con is an object of <class, cx_Oracle.Connection>

3. Every Python program Must Create an object of Cursor

=>after getting the connection from Oracle DB, the programmer must create an object of cursor.

=>The purpose of creating the cursor object is that "To carry the Query from Python Program, Handover to Database Software and Brings the result from database software and gives to Python Program".

=>To create an object of cursor, we have `curosr()` in connect object.

=>Syntax: `varname=conobj.cursor()`

4. Every Python Program must Design the Query, Place the Query in cursor object and execute

=>A Query a request / Question to the database from Program.

=>To execute a Query, we have `execute()` in cursor object

=>Syntax: `curobj.execute("Query")`

=>In SQL, We have Different Types of Queries. They are DDL , DML and DRL

===== Types of Queries in Database Softwares =====

=>In RDBMS Softwares, SQL Queries are classified into 3 Types. They are

1. DDL (Data Definition Language) Queries--create, alter, drop
 2. DML(Data Manipulation Language) Queries--insert,update,delete
 3. DRL (Data Retrieval Language) Queries ----select
-

#Program for obtaining the connection from Oracle

```
#TestOracleConEx1.py
import cx_Oracle
con=cx_Oracle.connect("system/manager@localhost/XE")
print("type con=",type(con))
print("Python Program got connection from Oracle")
```

#Program for obtaining the connection from Oracle

```
#TestOracleConEx2.py
import cx_Oracle
con=cx_Oracle.connect("system/manager@127.0.0.1/XE")
print("type con=",type(con))
print("Python Program got connection from Oracle")
```

#program for creating employee table with eno,ename and sal as col names in Oracle DB

```
#TableCreateEx1.py
import cx_Oracle # Step-1
try:
    con=cx_Oracle.connect("system/manager@localhost/xe") # Step-2
    cur=con.cursor() # Step-3
    #step-4
```

```

        ctq="create table employee (eno number(2),ename varchar2(10), sal
number(6,2)) "
        cur.execute(ctq) # Step-5
        print("Employee Table create sucessfully--verify")
except cx_Oracle.DatabaseError as db:
    print("Prob in DB:",db)
-----
#program for creating employee table with eno,ename and sal as col names in
Oracle DB
#TableCreateEx2.py
import cx_Oracle # Step-1
def tablecreate():
    try:
        con=cx_Oracle.connect("system/manager@localhost/xe") # Step-2
        cur=con.cursor() # Step-3
        #step-4
        ctq="create table teacher (tno number(2),tname varchar2(10), sub
varchar2(10)) "
        cur.execute(ctq) # Step-5
        print("Employee Table create sucessfully--verify")
    except cx_Oracle.DatabaseError as db:
        print("Prob in DB:",db)

#main program
tablecreate()

```

```

-----
#Program for Demonstarting adding new Column names to employee table
#AlterwithAdd.py
import cx_Oracle
def alteradd():
    try:
        con=cx_Oracle.connect("system/manager@localhost/xe")
        cur=con.cursor()
        aq="alter table employee add (dsg varchar2(10))"
        cur.execute(aq)
        print("Employee table altered--Verify")
    except cx_Oracle.DatabaseError as db:
        print("Prob in Oracle DB:",db)

#main program
alteradd()

```

```

#Program for Demonstarting altering Column sizez of employee table
#AlterwithModify.py
import cx_Oracle
def altermodify():
    try:
        con=cx_Oracle.connect("system/manager@localhost/xe")

```

```

        cur=con.cursor()
        aq="alter table employee modify (eno number(3),ename varchar2(15))"
        cur.execute(aq)
        print("Employee table altered--Verify")
    except cx_Oracle.DatabaseError as db:
        print("Prob in Oracle DB:",db)

#main program
altermodify()

-----
#Program for Demonstarting removing the table from Oracle DB
#removetable.py
import cx_Oracle
def removetab():

    try:
        con=cx_Oracle.connect("system/manager@localhost/xe")
        cur=con.cursor()
        cur.execute("drop table student")
        print("Student table removed--Verify")
    except cx_Oracle.DatabaseError as db:
        print("Prob in Oracle DB:",db)

#main program
removetab()

-----
#CursorObj.py
#Program for obtaining the cursor from Connection
import cx_Oracle
con=cx_Oracle.connect("system/manager@127.0.0.1/XE")
print("\ntype con=",type(con))
print("Python Program got connection from Oracle")
cur=con.cursor()
print("type of cur=",type(cur))
print("Python Program got cursor")
-----
```

===== 1. DDL (Data Definition Language) Queries =====

=>The purpose of DDL (Data Definition Language) Queries is that "To deal with Physical Level of Tables such as Table creation with column names, dropping tables and re-structuring columns of table".

=>DDL Queries are classified into 3 types. They are

1. create
2. alter
3. drop

1) create

=>It is used for creating Table in Database Software.

=>Syntax:

SQL>create table table-name(col1 DB Data Type, Col2 DB DataType,...Col-n DB Data Type)

Examples:

SQL> create table student (snonumber(2) primary key ,sname varchar2(10) not null ,marks number(5,2) not null);

2) alter----- add option modify option

=>This Query is used for altering table structure.

=>In Otherwords, alter is used for modifying the Column Sizes (modify) and adding new column names (add)

=>Syntax1:

SQL> alter table table-name modify(existing col-name1 DB Data Type,... existing col-name-n DB Data Type)

=>Syntax2:

SQL> alter table table-name add(new col-name1 DB Data Type,... new col-name-n DB Data Type)

Example1:

SQL> alter table teacher modify(tno number(3),tsal number(6,2));

Example2:

SQL> alter table teacher add(cname varchar2(10) not null);

3) drop

=>This query is used for removing or dropping the table from Database Software:

=>Syntax: SQL> drop table tablename

=>Examples: SQL > drop table employee

2. DML(Data Manipulation Language) Queries

=>The purpose of DML(Data Manipulation Language) Queries is that " To insert records, delete records and update records of any table".

=>DML(Data Manipulation Language) Queries are classified into 3 types. They are

1. insert
2. delete
3. update

=>After performing any DML Operation through Python Program, we must commit the database by using commit() and to undo the operation, we do roll back by using rollback().

=>commit() and rollback() are present in connection object.

1. insert

=>This Query is used for inserting Record in a table.

=>Syntax:-

SQL> insert into table-name values(val1 for col1,val2 for col2,...val-n for col-n)

Examples:

SQL> insert into employee values(20,'TR',1.9,'numpy');

2) delete

=>This Query is used for deleting a record from table.

=>Syntax1: SQL>delete from table-name ;
(OR)

=>Syntax2: SQL>delete from table-name where cond list ;

Examples:

SQL> delete from employee; #Deletes all records of employee table

SQL> delete from employee where eno=10; #Deletes Particular record of employee table

3)update

=>This Query is used for updating a record in a table.

=>Syntax1:- SQL>update table-name set col1=Val1,col2=val2....col-n=val-n;
(OR)

=>Syntax2:-

SQL>update table-name set col1=Val1,col2=val2....col-n=val-n where Cond List

#Program for inserting the employee records in employee table

#RecordInsertEx1.py

```
import cx_Oracle
def insertrecord():
    try:
```

```
        con=cx_Oracle.connect("system/manager@localhost/xe")
```

```
        cur=con.cursor()
```

```
        iq="insert into employee values(300,'Gopal',3,'infosys') "
```

```

        cur.execute(iq)
        con.commit()
        print("Employee Record Inserted Sucessfully:")
    except cx_Oracle.DatabaseError as db:
        print("Problem in Oracle DB:",db)

#main program
insertrecord()

-----
#Program for inserting the employee records in employee table
#RecordInsertEx2.py
import cx_Oracle,sys
def insertrecord():
    try:
        con=cx_Oracle.connect("system/manager@localhost/xe")
        cur=con.cursor()
        iq="insert into employee values(500,'Chendra',3.2,'HCL') "
        cur.execute(iq)
        ch=input("Do u want to Insert the record permanently(yes/no):")
        if(ch.lower()=='yes'):
            con.commit()
            print("Employee Record Inserted Sucessfully:")
        elif(ch.lower()=='no'):
            con.rollback()
            print("Employee Record Not Inserted:")
        else:
            print("Plz Learn Typing:")
            sys.exit()
    except cx_Oracle.DatabaseError as db:
        print("Problem in Oracle DB:",db)

#main program
insertrecord()

-----
#Program for inserting the employee records in employee table
#RecordInsertEx3.py
import cx_Oracle,sys
def insertrecord():
    try:
        con=cx_Oracle.connect("system/manager@localhost/xe")
        cur=con.cursor()
        #accept the employee Values from KBD
        eno=int(input("Enter Employee Number:"))
        ename=input("Enter Employee Name:")
        salary=float(input("Enter Employee Salary:"))
        dsg=input("Enter Employee Designation:")
        #prepare the query
        iq="insert into employee values(%d,'%s',%f,'%s' )"
        cur.execute(iq %(eno,ename,salary,dsg))
    
```

```

        con.commit()
        #OR cur.execute("insert into employee values(%d,'%s',%f,'%s' )"
%(eno,ename,salary,dsg) )
        print("{} Employee Record Inserted Sucessfully".format(cur.rowcount))
    except cx_Oracle.DatabaseError as db:
        print("Problem in Oracle DB:",db)

#main program
insertrecord()

-----
#Program for inserting the employee records in employee table
#RecordInsertEx4.py
import cx_Oracle,sys
def insertrecord():
    while(True):
        try:
            con=cx_Oracle.connect("system/manager@localhost/xe")
            cur=con.cursor()
            #accept the employee Values from KBD
            eno=int(input("Enter Employee Number:"))
            ename=input("Enter Employee Name:")
            salary=float(input("Enter Employee Salary:"))
            dsg=input("Enter Employee Designation:")
            #prepare the query
            iq="insert into employee values(%d,'%s',%f,'%s' )"
            cur.execute(iq %(eno,ename,salary,dsg))
            con.commit()
            #OR cur.execute("insert into employee values(%d,'%s',%f,'%s' )"
%(eno,ename,salary,dsg) )
            print("{} Employee Record Inserted
Sucessfully".format(cur.rowcount))
            print("-"*50)
            ch=input("Do u want to insert Another Record(yes/no):")
            if(ch.lower()!="no"):
                break
            elif(ch.lower()!="yes"):
                print("plz learn typing")
                break
        except cx_Oracle.DatabaseError as db:
            print("Problem in Oracle DB:",db)
        except ValueError:
            print("Don't enter alnums,strs and symbols for numbers and
Salaries")
    #main program
    insertrecord()

-----
#Program for deleting the employee records from employee table
#RecordDeleteEx1.py
import cx_Oracle,sys
def deleterecord():

```

```

try:
    con=cx_Oracle.connect("system/manager@localhost/xe")
    cur=con.cursor()
    #accept the employee Values from KBD
    eno=int(input("Enter Employee Number:"))
    #prepare the query
    iq="delete from employee where eno=%d"
    cur.execute(iq %eno) # OR cur.execute("delete from employee where
    eno=%d" %eno)
    con.commit()
    if(cur.rowcount>0):
        print("{} Record Deleted Sucessfully".format(cur.rowcount))
    else:
        print("Employee Record does not exist")
except cx_Oracle.DatabaseError as db:
    print("Problem in Oracle DB:",db)

#main program
deleterecord()

-----
#Program for deleting the employee records in employee table
#RecordDeleteEx2.py
import cx_Oracle,sys
def deleterecord():
    while(True):
        try:
            con=cx_Oracle.connect("system/manager@localhost/xe")
            cur=con.cursor()
            #accept the employee Values from KBD
            eno=int(input("Enter Employee Number:"))
            #prepare the query
            cur.execute("delete from employee where eno=%d" %eno)
            con.commit()
            if(cur.rowcount>0):
                print("{} Record Deleted
Sucessfully".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'):
                break
            elif(ch.lower()!="yes"):
                print("plz learn typing")
                break
        except cx_Oracle.DatabaseError as db:
            print("Problem in Oracle DB:",db)
        except ValueError:
            print("Don't enter alnums,strs and symbols for numbers")

#main program

```

deleterecord()

```
#Program for updating the employee records in employee table
#RecordUpdateEx1.py
import cx_Oracle,sys
def updaterecord():
    try:
        con=cx_Oracle.connect("system/manager@localhost/xe")
        cur=con.cursor()
        #accept the employee Values from KBD
        eno=int(input("Enter Employee Number for updating other details:"))
        sal=float(input("Enter New Salary of Employee:"))
        dsg=input("Enter New Desgination of Employee:")
        #prepare the query
        uq="update employee set sal=%f,dsg='%s' where eno=%d"
        cur.execute(uq %(sal,dsg,eno))
        con.commit()
        if(cur.rowcount>0):
            print("{} Record Updated Sucessfully".format(cur.rowcount))
        else:
            print("Employee Record does not exist")
    except cx_Oracle.DatabaseError as db:
        print("Problem in Oracle DB:",db)
    except ValueError:
        print("Don't enter alnums,strs and symbols for numbers and Salaries")
```

#main program
updaterecord()

Consider the following Query

cur.execute("select * from employee")

Once the above

Query is executed then cur object contains the following

| ENO | ENAME | SAL | DSG | <----Col Names--Meta Data |
|-----|---------|------|-------|---------------------------|
| 100 | SAIRAM | 5.6 | Sr.TL | |
| 200 | ZAKI | 55 | FL | |
| 300 | Gopal | 5.6 | PM | |
| 400 | Sujith | 3.5 | HCL | |
| 500 | Chendra | 12.3 | TL | |
| 550 | Rahul | 4.5 | TL | |

To get the records from cur object, we have 3 pre-defined functions. They are

1. fetchone()
2. fetchmany(no.of recs)
3. fetchall()

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 and it returns either tuple (if records exist) or None (if records does not exist)

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>0 and 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 in the form tuples of list.

Consider the following Query

cur.execute("select * from employee")

Once the above Query is executed then cur object contains the following

| ENO | ENAME | SAL | DSG |
|-----|---------|------|-------|
| 100 | SAIRAM | 5.6 | Sr.TL |
| 200 | ZAKI | 55 | FL |
| 300 | Gopal | 5.6 | PM |
| 400 | Sujith | 3.5 | HCL |
| 500 | Chendra | 12.3 | TL |
| 550 | Rahul | 4.5 | TL |

To get the records from cur object, we have 3 pre-defined functions. They are

1. fetchone()
2. fetchmany(no.of recs)
3. fetchall()

```
#program selecting the records from employee table
#SelectRecordsEx1.py
import cx_Oracle,time
def selectrecord():
    try:
        con=cx_Oracle.connect("system/manager@localhost/xe")
        cur=con.cursor()
        cur.execute("select * from employee")
        print("-----")
        while(True):
            record=cur.fetchone()
            if(record!=None):
                for val in record:
                    print("{}\t".format(val),end="")
                print()
            else:
                break
        print("-----")
    except cx_Oracle.DatabaseError as db:
        print("Problem in Oracle DB:",db)
```

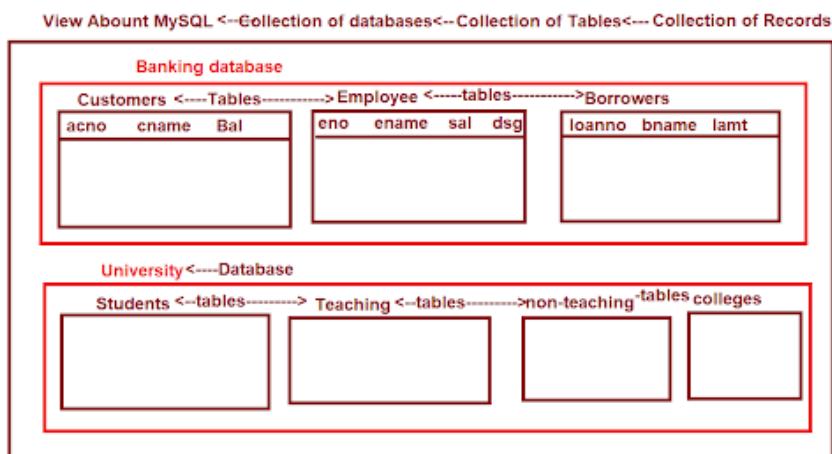
```
#main program
selectrecord()
```

```
#program selecting the records from employee table
#SelectRecordsEx2.py
import cx_Oracle,time
def selectrecord():
    try:
        con=cx_Oracle.connect("system/manager@localhost/xe")
        cur=con.cursor()
        cur.execute("select * from employee")
        print("-----")
        records=cur.fetchmany(3)
        for record in records:
            for val in record:
                print("{}\t".format(val),end="")
            print()
        print("-----")
    except cx_Oracle.DatabaseError as db:
        print("Problem in Oracle DB:",db)
```

```
#main program
selectrecord()
```

```
#program selecting the records from employee table
#SelectRecordsEx3.py
import cx_Oracle,time
def selectrecord():
    try:
        con=cx_Oracle.connect("system/manager@localhost/xe")
        cur=con.cursor()
        cur.execute("select * from employee")
        print("-----")
        records=cur.fetchall()
        for record in records:
            for val in record:
                print("{}\t".format(val))
            print()
        print("-----")
    except cx_Oracle.DatabaseError as db:
        print("Problem in Oracle DB:",db)

#main program
selectrecord()
```



Steps for Developing Python Program Communication with MySQL

Steps

1. import mysql.connector module
2. Every Python program must get the CONNECTION from MySQL Database
3. Every Python program Must Create an object of Cursor

4. Every Python Program must Design the Query, Place the Query in cursor object and execute
 5. Every Program must Process the Result of the Query which is coming from Database Software.
 6. Every Python Program Closes the Connection.
-

Explanation

Step-1: 1. import mysql.connector module

=>If a Python Program want to communicate with MySQL Database then we must import mysql.connector module

=>Examples: import mysql.connector

2. Every Python program must get the CONNECTION from MySQL Database

=>After importing mysql.connector Module, Python Program must get the connection from MySQL Database.

=>If a Python program want to get the connection from MySQL database then we must use connect() of mysql.connector module.

=>**Syntax:** varname=mysql.connector.connect("Connection URL")

Here Connection URL Represents the following

host=DNS, user="User name of MySQL",passwd="password of MySQL"
(OR)

Here Connection URL Represents

host=IP Address, user="User name of MySQL",passwd="password of MySQL"

Explanation

=>here varname is an object of Connection object

=>Here username and password represents user name and password of MySQL Database

=>DNS is nothing but name of the Physical Machine where Database Software Installed / resides. The Default DNS of Every Computer is "localhost"

=>IP Address is nothing but Address of the Physical Machine where Database Software Installed / resides. The Default IP ADDRESS of Every Computer is "127.0.0.1" (loop back address)

Example:

con=mysql.connector.connect(host="localhost",user="root",passwd="root")
here con is an object of <class, mysql.connector.MySQLConnection>

3. Every Python program Must Create an object of Cursor

=>after getting the connection from Oracle DB, the programmer must create an object of cursor.

=>The purpose of creating the cursor object is that "To carry the Query from Python Program, Handover to Database Software and Brings the result from database software and gives to Python Program".

=>To create an object of cursor, we have curosr() in connect object.

=>Syntax: varname=conobj.cursor()

4. Every Python Program must Design the Query, Place the Query in cursor object and execute

=>A Query a request / Question to the database from Program.

=>To execute a Query, we have execute() in cursor object

=>Syntax: curobj.execute("Query")

=>In SQL, We have Different Types of Queries. They are DDL , DML and DRL

```
#DisplayColNamesRecords.py
import mysql.connector, time
def selectrecord():
    try:
        con = mysql.connector.connect(host="127.0.0.1",
                                      user="root",
                                      passwd="root",
                                      database="batch9am")
        cur = con.cursor()
        cur.execute("select * from employee")
        print("-----")
        colinfo=cur.description
        for col in colinfo:
            print("{}".format(col[0]),end="\t")
        print()
        print("-----")
        #code for getting the records
        records=cur.fetchall()
        for record in records:
            for val in record:
                print("{}".format(val),end="\t")
            print()
        print("-----")
    except mysql.connector.DatabaseError as db:
        print("Problem in Oracle DB:", db)
    # main program
    selectrecord()
```

#Program adding new Col Name to the employee table

```
#MySQLColAddTableEx.py
import mysql.connector
def addcolumntable():
    try:
        con = mysql.connector.connect(host="127.0.0.1",
                                      user="root",
                                      passwd="root",
                                      database="batch9am")
        cur=con.cursor()
        ac="alter table employee add(dsg varchar(10) not null)"
        cur.execute(ac)
        print("Employee Table added a Column in MySQL")
    except mysql.connector.DatabaseError as db:
        print("Problem in MySQL DB:",db)
#main program
addcolumntable()

#Program for Demonstrating Connection from MySQL and creating cursor object
#MySQLConCurTest.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")
cur=con.cursor()
print("\ntype of cur=",type(cur))
print("Python Program Created Cursor")
#MySQLConCurTestwithIP.py
import mysql.connector
con=mysql.connector.connect(host="127.0.0.1",
                            user="root",
                            passwd="root")
print("type 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")
```

#Program for creating table in MYSQL

```
#MySQLCreateTableEx.py
import mysql.connector
def createtable():
    try:
        con = mysql.connector.connect(host="127.0.0.1",
                                      user="root",
                                      passwd="root",
                                      database="batch9am")
        cur=con.cursor()
```

```

#Query
tq="create table employee(eno int primary key,ename varchar(10) not null,sal
float not null)"
cur.execute(tq)
print("Employee Table Created in MySQL")
except mysql.connector.DatabaseError as db:
    print("Problem in MYSQL DB:",db)
#main program
createtable()
-----
```

#Program for creating the database name--batch9am

```

#MySQLDatabaseCreateEx.py
import mysql.connector
def cretedatabase():
    try:
        con = mysql.connector.connect(host="127.0.0.1",
                                      user="root",
                                      passwd="root")
        cur=con.cursor()
        #Query
        dq="create database batch9am"
        cur.execute(dq)
        print("Database Created in MYSQL Sucessfully")
    except mysql.connector.DatabaseError as db:
        print("Problem in MYSQL DB:",db)

#main program
cretedatabase()
-----
```

#Program for dropping the database name

```

#MySQLDatabaseDropEx.py
import mysql.connector
def dropdatabase():
    try:
        con = mysql.connector.connect(host="127.0.0.1",
                                      user="root",
                                      passwd="root")
        cur=con.cursor()
        #Query
        dq="drop database batch6pm"
        cur.execute(dq)
        print("Database removed from MYSQL Sucessfully")
    except mysql.connector.DatabaseError as db:
        print("Problem in MYSQL DB:",db)

#main program
dropdatabase()
-----
```

#Program inserting a record in a Employee Table

```
#MySQLRecordInsertEx1.py
```

```

import mysql.connector
def insertrecord():
    try:
        con = mysql.connector.connect(host="127.0.0.1",
                                      user="root",
                                      passwd="root",
                                      database="batch9am")
        cur=con.cursor()
        iq="insert into employee values(300,'Gopal',3,'SE') "
        cur.execute(iq)
        con.commit()
        print("Employee Record Inserted Sucessfully:")
    except mysql.connector.DatabaseError as db:
        print("Problem in MYSQL DB:",db)

#main program
insertrecord()
-----
#MySQLSample.py
import mysql.connector
print(mysql.connector.__version__)
#Program for deleting the employee records in employee table
#RecordDeleteEx2.py
import mysql.connector,sys
def deleterecord():
    while(True):
        try:
            con=mysql.connector.connect(host="127.0.0.1",
                                         user="root",
                                         passwd="root",
                                         database="batch9am")
            cur=con.cursor()
            #accept the employee Values from KBD
            eno=int(input("Enter Employee Number:"))
            #prepare the query
            cur.execute("delete from employee where eno=%d" %eno)
            con.commit()
            if(cur.rowcount>0):
                print("{} Record Deleted
Sucessfully".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"):
                break
            elif(ch.lower()!="yes"):
                print("plz learn typing")
                break
        except mysql.connector.DatabaseError as db:

```

```

        print("Problem in MySQLLe DB:",db)
    except ValueError:
        print("Don't enter alnums,strs and symbols for numbers ")
#main program
deleterecord()

-----



#Program for inserting the employee records in employee table
#RecordInsertEx4.py
import mysql.connector,sys
def insertrecord():
    while(True):
        try:
            con=mysql.connector.connect(host="127.0.0.1",
                                         user="root",
                                         passwd="root",
                                         database="batch9am")
            cur=con.cursor()
            #accept the employee Values from KBD
            eno=int(input("Enter Employee Number:"))
            ename=input("Enter Employee Name:")
            salary=float(input("Enter Employee Salary:"))
            dsg=input("Enter Employee Designation:")
            #prepare the query
            iq="insert into employee values(%d,'%s',%f,'%s' )"
            cur.execute(iq %(eno,ename,salary,dsg))
            con.commit()
            #OR cur.execute("insert into employee values(%d,'%s',%f,'%s' )"
            %(eno,ename,salary,dsg))
            print("{} Employee Record Inserted
Sucessfully".format(cur.rowcount))
            print("-"*50)
            ch=input("Do u want to insert Another Record(yes/no):")
            if(ch.lower()!="no"):
                break
            elif(ch.lower()!="yes"):
                print("plz learn typing")
                break
        except mysql.connector.DatabaseError as db:
            print("Problem in MySQL DB:",db)
        except ValueError:
            print("Don't enter alnums,strs and symbols for numbers and
Salaries")
#main program
insertrecord()

-----



#Program for updating the employee records in employee table
#RecordUpdateEx1.py
import mysql.connector,sys
def updaterecord():
    try:

```

```

con=mysql.connector.connect(host="127.0.0.1",
                            user="root",
                            passwd="root",
                            database="batch9am")
cur=con.cursor()
#accept the employee Values from KBD
eno=int(input("Enter Employee Number for updating other details:"))
sal=float(input("Enter New Salary of Employee:"))
dsg=input("Enter New Desgination of Employee:")
#prepare the query
uq="update employee set sal=%f,dsg='%s' where eno=%d"
cur.execute(uq %(sal,dsg,eno))
con.commit()
if(cur.rowcount>0):
    print("{} Record Updated Sucessfully".format(cur.rowcount))
else:
    print("Employee Record does not exist")
except mysql.connector.DatabaseError as db:
    print("Problem in Oracle DB:",db)
except ValueError:
    print("Don't enter alnums,strs and symbols for numbers and
Salaries")

#main program
updaterecord()

-----
#program selecting the records from employee table
#SelectRecordsEx2.py
import mysql.connector,time
def selectrecord():
    try:
        con=mysql.connector.connect(host="127.0.0.1",
                                    user="root",
                                    passwd="root",
                                    database="batch9am")
        cur=con.cursor()
        cur.execute("select * from employee")
        print("-----")
        records=cur.fetchmany()
        for record in records:
            for val in record:
                print("{}".format(val),end="\t")
            print()
        print("-----")
    except mysql.connector.DatabaseError as db:
        print("Problem in Oracle DB:",db)

#main program
selectrecord()

```

```
#program selecting the records from employee table
#SelectRecordsEx3.py
import mysql.connector,time
def selectrecord():
    try:
        con=mysql.connector.connect(host="127.0.0.1",
                                     user="root",
                                     passwd="root",
                                     database="batch9am")
        cur=con.cursor()
        cur.execute("select * from employee")
        print("-----")
        records=cur.fetchall()
        for record in records:
            for val in record:
                print("{}{}".format(val),end="\t")
            print()
        print("-----")
    except mysql.connector.DatabaseError as db:
        print("Problem in Oracle DB:",db)
```

```
#main program
selectrecord()
```

Object Oriented Principles or Features or Concepts (10 days)

Index:

=>What are the advantages of OOPs
 =>List of Object Oriented Principle

1. Classes
 2. Objects
 3. Data Encapsulation
 4. Data Abstraction
 5. Inheritance
 6. Polymorphism
 7. Message Passing(already discussed)
-

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
-

- =>Destructrors in OOPs with Garbage Collector
 - =>Importance and purpose of Detstructrors
 - =>Syntax for defining Detstructrors
 - =>Internal flow of Detstructrors
 - =>Relation between Detstructrors and Garbage Collector
 - =>gc module
 - =>Pgramming Examples
-

3. Data Encapsulation and 4. Data Abstraction

- =>Importance and purpose of Data Encapsulation
 - =>Importaance and purpose of Data Abstraction
 - =>Implementation of data encapsulation and Data Abstraction
 - =>Programming Examples
-

5. Inheritance

- =>Importaance 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
 - =>Constructor Overriding with Polymorphism
 - =>`super()` and class name approaches in Polymorphism
 - =>Programming Examples
-

=====

Introduction to Object Oriented Programming Principles (OOPs)

=====

=>In real Time, To develop any project / Application, we use a Language and It can Satisfy to either Procedure Oriented Features(Functional Programming) OR Object Oriented Features.

=>In Other Words, we have Two Types of Programming languages. They are

1. Procedure Oriented(Functional) Programming Languages.

 Examples: C,8086,upto Oracle7.3,Pascal,Cobol,PYTHON

2. Object Oriented Programming Languages.

 Examples: c++,java,c#.net,smalltalk,ruby,PYTHON,from Oracle8 onwards

=>Hence PYTHON Programming Lang Belongs to Both Functional and Object Oriented Programming lang.

=>Even Python Programming Lang Belongs to Functional Programming lang, Internally Every Thing is Treated as object.

"Every Thing in Python is Treated as Object"—Advantages (OR)
Advantages of Object Oriented Principles

1. The Concept of object allows us to store Large Volume of Data.
2. The Confidential data / Information is Transmitting between the Remote Machines in the form of Cipher text / Encrypted Format. So that we can get Security.
3. The large Volume of Data Transferred Between Multiple Remote machines all at once. So that we can Effective Communication.
4. The data is always available in the form of Objects. So that we can Effective Memory Management.
5. Provides Minimal Memory space for application development bcoz OOPs Provides Re-usableMechanisms.

Object Oriented Programming Principles (OOPs)

=>If any programming is said to be Object Oriented then that programming lang must satisfy the principles of Object oriented. They are

1. Classes
 2. Objects
 3. Data Encapsulations
 4. Data Abstraction
 5. Inheritance
 6. Polymorphism
 7. Message Passing (already discussed)
-
-

1. Classes

=>The purpose of Class is that "To develop Programmer-Defined Data Types and to develop any real Time Applications".

=>The purpose of Developing Programmer-Defined Data Type is that " To store Customized data(Data Members)and to Perform Customized Operations (Methods)".

=>To develop programmer-defined data type with classes concept , we use a Keyword called "class".

=>Programmatically, Every Class name in python is treated as Programmer-defined datatype.

=>Every Program in OOPs Must Starts with Classes Concept.

Definition of Class

=>A Class is a Collection of Data Members and Methods

=>When we define a class , There is no memory is created for data members and Methods but whose

memory space is created when we create an object.

=>What are all the Data Members and Methods we define inside of Class, They are Reflected / Available in the object with memory space.

=>Hence Classes always Considered as Logical Entities and Contains Logical Existence. where as an object is considered as Physical Entity and Contains Physical Existence.

Syntax for Defining a class in Python

=>To Define a class in Python, we use the following Syntax.

```
class <clsname>:
    Class Level Data Members
    def instancemethodname(self,list of formal params if any):
        -----
        Performs Specific Operations
        Specify Instance Data Members
        -----
        @classmethod
        def classmethodname(cls,list of formal params if any):
            -----
            Performs Common Operations
            Specify Class Level Data Members
            -----
            @staticmethod
            def staticmethodname(list of formal params if any):
                -----
                Performs Universal OR Utility Operations
```

Types of Data Members in Class of Python

=>In a Class of Python, we can define Two Types of data members. They are

1. Instance Data Memebers
 2. Class Level Data Members
-

1. Instance Data Memebers

=>Instance Data Memebers are those whose memory space created Every Time when an object is created.

=>Instance Data Members are Used for storing Specific values in object. Hence Instance data members are also called Object Level Data Members.

=>Instance Data Memebers can be specified in 3 ways. They are

- a) Through Object name
- b) Through Instance Method Name
- c) Through Constructors

=>Instance Data Memebers can be accessed by using the following Syntaxes.

Syntax1: objname.Instance Data Members name

Syntax2: self.Instance Data Members name

2. Class Level Data Members

=>Class Level Data Memebers are those whose memory space created One Time irrespective of Number of Objects are created.

=>Class Level Data Members are Used for storing Common values for all objects of same Class.

=>Class Level DataData Memebers can be specified in 3 ways. They are

- a) Inside of Class name
- b) Inside of Class Level Method Definition

=>Class Level Data Memebers can be accessed by using the following Syntaxes.

- Syntax1: Classname.Class Level Data Member name
 - Syntax2: cls.Class Level Data Member name
 - Syntax3: objname.Class Level Data Member name
 - Syntax4: self.Class Level Data Member name
-

#Program for storing Student Number, name and marks ----(Instance data members)

#StudEx1.py

```
class Student:pass
```

#main program

```
s1=Student()
print("Student Object Created")
print("Memory Address of s1=",id(s1))
```

#Program for storing Student Number, name and marks ----(Instance data members)

#StudEx2.py

```
class Student:pass
```

#main program

```
s1=Student()
print("Initial Content of s1={} and length={}".format(s1.__dict__, len(s1.__dict__)))
#Add Instance Data Members to object
s1.sno=100
s1.sname="RS"
s1.marks=22.22
print("Content of s1={} and length={}".format(s1.__dict__,len(s1.__dict__)))
s1.dob="13-04-2023"
print("Content of s1={} and length={}".format(s1.__dict__,len(s1.__dict__)))
```

#Program for storing Student Number, name and marks ----(Instance data members)

#StudEx3.py

```
class Student:pass
```

#main program

```
s1=Student()
s2=Student()
print("-"*50)
print("Intial Content of s1=",s1.__dict__)
print("Intial Content of s2=",s2.__dict__)
print("-"*50)
#Add Instance Data members to s1 object
s1.stno=1
s1.sname="RS"
```

```

#Add Instance Data members to s2 object
s2.sno=2
s2.name="TR"
print("Content of s1 after adding the data=",s1.__dict__)
print("Content of s2 after adding the data=",s2.__dict__)
print("-"*50)
-----
#Program for storing Student Number, name and marks ----(Instance data
members)
#StudEx4.py
class Student:pass

#main program
s1=Student()
s2=Student()
print("-"*50)
print("Intial Content of s1=",s1.__dict__)
print("Intial Content of s2=",s2.__dict__)
print("-"*50)
#Add Instance Data members to s1 object
s1.stno=1
s1.sname="RS"
#Add Instance Data members to s2 object
s2.sno=2
s2.name="TR"
print("-"*50)
print("Content of S1:")
print("-"*50)
print("\tStudent Number:{}".format(s1.stno))
print("\tStudent Name:{}".format(s1.sname))
print("-"*50)
print("Content of S2:")
print("-"*50)
print("\tStudent Number:{}".format(s2.sno))
print("\tStudent Name:{}".format(s2.name))
print("-"*50)
print("=====OR=====")
print("Content of S1:")
print("-"*50)
for dmn,dmv in s1.__dict__.items():
    print("\t{}-->{}".format(dmn,dmv))
print("-"*50)
print("Content of S2:")
print("-"*50)
for dmn,dmv in s2.__dict__.items():
    print("\t{}-->{}".format(dmn,dmv))
print("-"*50)
-----
#Program for storing Student Number, name and marks ----(Instance data
members)

```

```
#StudEx5.py
class Student:pass

#main program
s1=Student()
s2=Student()
print("-"*50)
print("Intial Content of s1=",s1.__dict__)
print("Intial Content of s2=",s2.__dict__)
print("-"*50)
#Add Instance Data members to s1 object by reading dynamic data
s1.stno=int(input("Enter First Student Number:"))
s1.sname=input("Enter First Student Name:")
#Add Instance Data members to s2 object by reading dynamic data
s2.sno=int(input("Enter Second Student Number:"))
s2.name=input("Enter Second Student Name:")
print("-"*50)
print("Content of S1:")
print("-"*50)
print("\tStudent Number:{}".format(s1.stno))
print("\tStudent Name:{}".format(s1.sname))
print("-"*50)
print("Content of S2:")
print("-"*50)
print("\tStudent Number:{}".format(s2.sno))
print("\tStudent Name:{}".format(s2.name))
print("-"*50)
print("=====OR=====")
print("Content of S1:")
print("-"*50)
for dmn,dmv in s1.__dict__.items():
    print("\t{}-->{}".format(dmn,dmv))
print("-"*50)
print("Content of S2:")
print("-"*50)
for dmn,dmv in s2.__dict__.items():
    print("\t{}-->{}".format(dmn,dmv))
print("-"*50)
```

#Program for storing Student Number, name , marks(Instance data members) and course which is common for all students (Class Level data members)

```
#StudEx6.py
class Student:
    crs="PYTHON" # Class Level data member

#main program
s1=Student()
s2=Student()
print("-"*50)
print("Intial Content of s1=",s1.__dict__)
```

```

print("Intial Content of s2=",s2.__dict__)
print("-"*50)
#Add Instance Data members to s1 object by reading dynamic data
s1.stno=int(input("Enter First Student Number:"))
s1.sname=input("Enter First Student Name:")
#Add Instance Data members to s2 object by reading dynamic data
s2.sno=int(input("Enter Second Student Number:"))
s2.name=input("Enter Second Student Name:")
print("-"*50)
print("Content of S1:")
print("-"*50)
print("\tStudent Number:{}".format(s1.stno))# Accessing Instance Data Memebrs w.r.t
object name
print("\tStudent Name:{}".format(s1.sname)) # Accessing Instance Data Memebrs
w.r.t object name
print("\tSTUDENT COURSE:{}".format(Student.crs)) #Accessing Class Level Data
Member w.r.t Class Name
print("-"*50)
print("Content of S2:")
print("-"*50)
print("\tStudent Number:{}".format(s2.sno))
print("\tStudent Name:{}".format(s2.name))
print("\tSTUDENT COURSE:{}".format(Student.crs)) #Accessing Class Level Data
Member w.r.t Class Name
print("-"*50)
print("=====OR====")
print("Content of S1:")
print("-"*50)
for dmn,dmv in s1.__dict__.items():
    print("\t{}-->{}".format(dmn,dmv))
print("\tSTUDENT COURSE:{}".format(Student.crs)) #Accessing Class Level Data
Member w.r.t Class Name
print("-"*50)
print("Content of S2:")
print("-"*50)
for dmn,dmv in s2.__dict__.items():
    print("\t{}-->{}".format(dmn,dmv))
print("\tSTUDENT COURSE:{}".format(Student.crs)) #Accessing Class Level Data
Member w.r.t Class Name
print("-"*50)

```

#Program for storing Student Number, name , marks(Instance data members) and course which is common for all students (Class Level data members)

```

#StudEx8.py
class Student:
    crs="PYTHON" # Class Level data member

```

```

#main program
s1=Student()
s2=Student()

```

```

print("-"*50)
print("Initial Content of s1=",s1.__dict__)
print("Initial Content of s2=",s2.__dict__)
print("-"*50)
#Add Instance Data members to s1 object by reading dynamic data
s1.stno=int(input("Enter First Student Number:"))
s1.sname=input("Enter First Student Name:")
#Add Instance Data members to s2 object by reading dynamic data
s2.sno=int(input("Enter Second Student Number:"))
s2.name=input("Enter Second Student Name:")
print("-"*50)
print("Content of S1:")
print("-"*50)
print("\tStudent Number:{}".format(s1.stno))# Accessing Instance Data Memebers w.r.t
object name
print("\tStudent Name:{}".format(s1.sname)) # Accessing Instance Data Memebers
w.r.t object name
print("\tSTUDENT COURSE:{}".format(s1.crs)) #Accessing Class Level Data
Member w.r.t object Name
print("-"*50)
print("Content of S2:")
print("-"*50)
print("\tStudent Number:{}".format(s2.sno))
print("\tStudent Name:{}".format(s2.name))
print("\tSTUDENT COURSE:{}".format(s2.crs)) #Accessing Class Level Data
Member w.r.t Class Name
print("-"*50)
print("=====OR=====")
print("Content of S1:")
print("-"*50)
for dm, dmv in s1.__dict__.items():
    print("\t{}-->{}".format(dm, dmv))
print("\tSTUDENT COURSE:{}".format(s1.crs)) #Accessing Class Level Data
Member w.r.t object Name
print("-"*50)
print("Content of S2:")
print("-"*50)
for dm, dmv in s2.__dict__.items():
    print("\t{}-->{}".format(dm, dmv))
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 class of Python, we 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 Operations on objects. In Otherwords, to perform the operations on objects data, we always use Instance Methods. Hence Instance Methods are called Object Level Methods.

=>Programtically, Instance Methods always takes "self" as First Possitional Formal Parameter.

=>Syntax: def instancemethod(self, list of formal Parameters if any):

Perform Specific Operations
Specify Instance Data Members

=>All types of Instance Methods Must accessed w.r.t Object Name or self
objectname.Instance Method
(OR)
self.Instance Method Name

What is self

=>"self" is of the Implicit Object which is always contains Ref /address of current object

=>"self" to be used always as First Possitonal Parameter in Instance Method

=>"self" to be accessed within Corresponding Instance Method body but no program bcoz It is a First Possitonal Parameter .

2. Class Level Methods

=>Class Level Methods are used for Performing Common Operations for the objects of Corresponding class and specifies Class Level Data Memebers.

=>To define Class Level Method, we must use a pre-defined Decorator called @classmethod

=>The Syntax for defining class level method is @classmethod

def classlevelmethod(cls,list of formal params if any):

Specify Class Level Data Members
Perform Common Operations for all objects of corresponding
objects.

=>Every Class Level Method must be accssed w.r.t to Class Name OR cls OR Object Name OR self

ClassName.Class Level Method

(OR)

cls.Class Level Method

(OR)
objectname.Class Level Method
(OR)
self.Class Level Method

What is cls:

=>"cls" is one of the implicit object and it contains Current Class Name
 =>"cls" always to be used as First Formal Parameter in Class Level Method
 =>Since "cls" is a Formal parameter, so that it can access inside of Corresponding Class Level Method Definition only but not possible to access in 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 OR cls OR self

ClassName.static method name()
 (OR)
 ObjectName.static method name()
 (OR)
 cls.static method name()
 (OR)
 self.static method name()

#Program for Demonstrating Instance Methods in a class

```
#InstanceMethodEx1.py
class Student:
    def readstudvalues(self):
        print("Ref of current object=",id(self))
        print("-----")
```

```
#main program
s1=Student() # Create student object--{}
s2=Student() # Create student object--{}
print("-----")
print("Ref s1 in main program=",id(s1))
s1.readstudvalues()
```

```

print("Ref s2 in main program=",id(s2))
s2.readstudvalues()
-----
#Program for Demonstrating Instance Methods in a class
#InstanceMethodEx2.py
class Student:
    def readstudvalues(self,sinfo):
        self.sno=int(input("\nEnter {} Student Number:".format(sinfo)))
        self.sname=input("Enter {} Student Name:".format(sinfo))
        self.marks=float(input("Enter {} Student Marks:".format(sinfo)))

    def dispstudinfo(self,sinfo):
        print("-----")
        print("{} Student Information:".format(sinfo))
        print("-----")
        print("Student Number:{}".format(self.sno))
        print("Student Name:{}".format(self.sname))
        print("Student Marks:{}".format(self.marks))
        print("-----")

#main program
s1=Student() # Create student object--{}
s2=Student() # Create student object--{}

s1.readstudvalues("First")
s2.readstudvalues("Second")
s1.dispstudinfo("First")
s2.dispstudinfo("Second")
-----
#Program for Demonstrating Instance Methods in a class
#InstanceMethodEx3.py
class Student:
    def readstudvalues(self,sinfo):
        self.sno=int(input("\nEnter {} Student Number:".format(sinfo)))
        self.sname=input("Enter {} Student Name:".format(sinfo))
        self.marks=float(input("Enter {} Student Marks:".format(sinfo)))
        self.dispstudinfo(sinfo) # calling One Instance Method from another
        Instance Method

    def dispstudinfo(self,sinfo):
        print("-----")
        print("{} Student Information:".format(sinfo))
        print("-----")
        print("Student Number:{}".format(self.sno))
        print("Student Name:{}".format(self.sname))
        print("Student Marks:{}".format(self.marks))
        print("-----")

#main program
s1=Student() # Create student object--{}

```

```
s2=Student() # Create student object--{}
s1.readstudvalues("First")
s2.readstudvalues("Second")
```

```
#Program for cal addition of Two Numbers by using Classes and objects
#InstanceMethodEx4.py
```

```
class Sum:
```

```
    def readvalues(kvrself):
        kvrself.a=float(input("Enter value of a:"))
        kvrself.b=float(input("Enter value of b:"))

    def addvalues(self):
        self.c=self.a+self.b

    def dispvalues(self):
        print("-----")
        print("sum({},{})={}".format(self.a,self.b,self.c))
        print("-----")
```

```
#main program
```

```
s=Sum()
s.readvalues()
s.addvalues()
s.dispvalues()
```

```
#Program for Demonstrating Instance and Class Level Methods in a class
```

```
#ClassInstanceMethodEx1.py
```

```
class Student:
```

```
    @classmethod
    def getuniv(cls):
        cls.un="OU" # OR Student.un="OU"

    def readstudvalues(self,sinfo):
        self.sno=int(input("\nEnter {} Student Number:".format(sinfo)))
        self.sname=input("Enter {} Student Name:".format(sinfo))
        self.marks=float(input("Enter {} Student Marks:".format(sinfo)))
        self.dispstudinfo(sinfo) # calling One Instance Method from another
```

Instance Method

```
def dispstudinfo(self,sinfo):
    print("-----")
    print("{} Student Information:".format(sinfo))
    print("-----")
    print("Student Number:{}".format(self.sno))
    print("Student Name:{}".format(self.sname))
    print("Student Marks:{}".format(self.marks))
    print("Student University:{}".format(Student.un))
    print("-----")
```

```

#main program
Student.getuniv() # calling Class Level Method By using Class Name
s1=Student() # Create student object--{}
s2=Student() # Create student object--{}
s1.readstudvalues("First")
s2.readstudvalues("Second")
-----
#Program for Demonstrating Instance and Class Level Methods in a class
#ClassInstanceMethodEx2.py
class Student:
    @classmethod
    def getuniv(cls):
        cls.un="OU" # OR Student.un="OU"
        Student.getunivaddr() # Calling One Class Level Method from another
class level

method
    @classmethod
    def getunivaddr(cls):
        cls.addr="HYD"
    def readstudvalues(self,sinfo):
        self.sno=int(input("\nEnter {} Student Number:".format(sinfo)))
        self.sname=input("Enter {} Student Name:".format(sinfo))
        self.marks=float(input("Enter {} Student Marks:".format(sinfo)))
        self.dispstudinfo(sinfo) # calling One Instance Method from another
        Instance Method

    def dispstudinfo(self,sinfo):
        print("-----")
        print("{} Student Information:".format(sinfo))
        print("-----")
        print("Student Number:{}".format(self.sno))
        print("Student Name:{}".format(self.sname))
        print("Student Marks:{} ".format(self.marks))
        print("Student University:{} ".format(Student.un))
        print("-----")

#main program
Student.getuniv() # calling Class Level Method By using Class Name
s1=Student() # Create student object--{}
s2=Student() # Create student object--{}
s1.readstudvalues("First")
s2.readstudvalues("Second")
-----
#Program for Demonstrating Instance and Class Level Methods in a class
#ClassInstanceMethodEx3.py
class Student:
    @classmethod
    def getuniv(cls):

```

```

cls.un="OU" # OR Student.un="OU"
cls.getunivaddr() # Calling One Class Level Method from another class
level method w.r.t cls
@classmethod
def getunivaddr(cls):
    cls.addr="HYD"
def readstudvalues(self,sinfo):
    self.sno=int(input("\nEnter {} Student Number:".format(sinfo)))
    self.sname=input("Enter {} Student Name:".format(sinfo))
    self.marks=float(input("Enter {} Student Marks:".format(sinfo)))
    self.dispstudinfo(sinfo) # calling One Instance Method from another
Instance Method

def dispstudinfo(self,sinfo):
    print("-----")
    print("{} Student Information:".format(sinfo))
    print("-----")
    print("Student Number:{}".format(self.sno))
    print("Student Name:{}".format(self.sname))
    print("Student Marks:{}".format(self.marks))
    print("Student University:{}".format(Student.un))
    print("-----")

#main program
Student.getuniv() # calling Class Level Method By using Class Name
s1=Student() # Create student object--{}
s2=Student() # Create student object--{}
s1.readstudvalues("First")
s2.readstudvalues("Second")
-----
#Program for Demonstrating Instance and Class Level Methods in a class
#ClassInstanceMethodEx4.py
class Student:
    @classmethod
    def getuniv(cls):
        cls.un="OU" # OR Student.un="OU"
    @classmethod
    def getunivaddr(cls):
        cls.addr="HYD"
    def readstudvalues(self,sinfo):
        self.sno=int(input("\nEnter {} Student Number:".format(sinfo)))
        self.sname=input("Enter {} Student Name:".format(sinfo))
        self.marks=float(input("Enter {} Student Marks:".format(sinfo)))
        self.dispstudinfo(sinfo) # calling One Instance Method from another
Instance Method
    def dispstudinfo(self,sinfo):
        print("-----")
        print("{} Student Information:".format(sinfo))
        print("-----")
        print("Student Number:{}".format(self.sno))

```

```

print("Student Name:{}".format(self.sname))
print("Student Marks:{}".format(self.marks))
#calling Class Level Method from instance Method by using Class
Name
Student.getuniv()
Student.getunivaddr()
print("Student University:{}".format(Student.un))
print("Student University Addr:{}".format(Student.addr))
print("-----")

#main program
s1=Student() # Create student object--{}
s2=Student() # Create student object--{}
s1.readstudvalues("First")
s2.readstudvalues("Second")
-----
#Program for Demonstrating Instance and Class Level Methods in a class
#ClassInstanceMethodEx5.py
class Student:
    @classmethod
    def getuniv(cls):
        cls.un="OU" # OR Student.un="OU"
    @classmethod
    def getunivaddr(cls):
        cls.addr="HYD"
    def readstudvalues(self,sinfo):
        self.sno=int(input("\nEnter {} Student Number:".format(sinfo)))
        self.sname=input("Enter {} Student Name:".format(sinfo))
        self.marks=float(input("Enter {} Student Marks:".format(sinfo)))
        self.dispstudinfo(sinfo) # calling One Instance Method from another
        Instance Method

    def dispstudinfo(self,sinfo):
        print("-----")
        print("{} Student Information:".format(sinfo))
        print("-----")
        print("Student Number:{}".format(self.sno))
        print("Student Name:{}".format(self.sname))
        print("Student Marks:{}".format(self.marks))
        #calling Class Level Method from instance Method by using self
        self.getuniv()
        self.getunivaddr()
        print("Student University:{}".format(Student.un))
        print("Student University Addr:{}".format(Student.addr))
        print("-----")

#main program
s1=Student() # Create student object--{}
s2=Student() # Create student object--{}
s1.readstudvalues("First")

```

```

s2.readstudvalues("Second")
-----
#Program for demonstrating Static Methods
#StaticMethodEx1.py
class Student:
    def readstudvalues(self):
        self.sno=int(input("Enter Student Number:"))
        self.sname=input("Enter Student Name:")
        self.marks=float(input("Enter Student Marks:"))

class Employee:
    def readempvalues(self):
        self.eno=int(input("\nEnter Employee Number:"))
        self.ename=input("Enter Employee Name:")
        self.sal=float(input("Enter Employee salary:"))
        self.dsg=input("Enter Employee Designation:")

class Teacher:
    def readteachervalues(self):
        self.tno=int(input("\nEnter teacher Number:"))
        self.tname=input("Enter teacher Name:")

class Hyd:
    @staticmethod
    def dispobjectdata( obj,infor ):
        print("-----")
        print("{} Information".format(infor))
        print("-----")
        for dmn,dmv in obj.__dict__.items():
            print("\t{}-->{}".format(dmn,dmv))
        print("-----")

#main program
s=Student()
e=Employee()
t=Teacher()
s.readstudvalues()
e.readempvalues()
t.readteachervalues()
#I want display the values of any values irrespective of Class Name--static method
Hyd.dispobjectdata(s,"Student")
Hyd.dispobjectdata(t,"teacher")
Hyd.dispobjectdata(e,"Employee")
-----
#Program for demonstrating Static Methods
#StaticMethodEx2.py
class Student:
    def readstudvalues(self):
        self.sno=int(input("Enter Student Number:"))
        self.sname=input("Enter Student Name:")

```

```

        self.marks=float(input("Enter Student Marks:"))

class Employee:
    def readempvalues(self):
        self.eno=int(input("\nEnter Employee Number:"))
        self.ename=input("Enter Employee Name:")
        self.sal=float(input("Enter Employee salary:"))
        self.dsg=input("Enter Employee Designation:")

class Teacher:
    def readteachervalues(self):
        self.tno=int(input("\nEnter teacher Number:"))
        self.tname=input("Enter teacher Name:")

class Hyd:
    @staticmethod
    def dispobjectdata( obj,infor ):
        print("-----")
        print("{} Information".format(infor))
        print("-----")
        for dmn,dmv in obj.__dict__.items():
            print("\t{}-->{}".format(dmn,dmv))
        print("-----")

#main program
s=Student()
e=Employee()
t=Teacher()
s.readstudvalues()
e.readempvalues()
t.readteachervalues()
#I want display the values of any values irrespective of Class Name--static method
h=Hyd()
h.dispobjectdata(s,"Student")
h.dispobjectdata(t,"teacher")
h.dispobjectdata(e,"Employee")

```

===== Constructors in Python =====

=>The purpose of Constructors in python is that " To initlize the object".
=>Initlizing the object is nothing but Placing our own data in object without leaving object empty.

Definition of Constructor

=>A Constructor is of the special method which is automatically / Implicitly called by PVM During Object Creation and whose purpose is to initialize the object..

Syntax for defining Constructor:

```
def __init__(self, list of formal params if any):
```

Block of Statements
Performs Initialization

Rules or Properties of Constructors

1. The Name of the constructor is always def __init__(self,...)
 2. Constructors will call automatically / implicitly by PVM during object creation
 3. Constructors will not return any value except None.
 4. In Python, Constructors can participate in Inheritance Process.
 5. In Python, Constructors can be Overridden
-

Constructors in Python

Index

- =>purpose of Constructor
 - =>Definition of Constructor
 - =>Syntax for defining Constructor
 - =>Rules / Properties of Constructors
 - =>Types of Constructors
 - =>Programming Examples
-

Types of 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 Default or Parameter Less Constructor is one, which never takes any Formal Parameters except self.

=>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 statements
Performs Initialization Process

Example:

```
#program for demonstrating Default Constructor
#DefaultConstEx1.py
class Test:
    def __init__(self):
        print("I am from default constructor:")
        self.a=10
        self.b=20
        print("\ta={}\tb={}".format(self.a,self.b))

#main program
t1=Test()# Object creation calls default constructor
t2=Test()# Object creation calls default constructor
t3=Test()# Object creation calls default constructor
```

2. Parameterized Constructor

=>A Parameterized Constructor is one, which always takes Formal Parameters after self.

=>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 statements
Performs Initialization Process

```
#program for demonstrating Parametrized Constructor
#ParamConstEx1.py
class Test:
    def __init__(self,k,v):
        print("I am from Parametrized constructor:")
        self.a=k
        self.b=v
        print("\ta={}\tb={}".format(self.a,self.b))

#main program
t1=Test(10,20)# Object creation calls Parametrized Constructor
t2=Test(100,200)# Object creation calls Parametrized Constructor
t3=Test(1000,2000)# Object creation calls Parametrized Constructor
```

Most Imp Point:

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.

```
#program for demonstrating Parametrized and DefaultConstructor
#ParamDefualtConstEx1.py
class Test:
    def __init__(self,k=1,v=2): # default and parameterized
        print("i am from default / Parametrized constructor:")
        self.a=k
        self.b=v
        print("\ta={}\tb={}".format(self.a,self.b))

#main program
t1=Test()# Object creation calls default Constructor
t2=Test(10,20)# Object creation calls Parametrized Constructor
#non-destex.py
class Employee:
    def readempvalues(self):
        self.eno=10
        self.ename="RS"

#main program
e=Employee()
print("content of e before method calling=",e.__dict__)
e.readempvalues() # explicitly, we defining a method in Employee class
print("content of e after method calling=",e.__dict__)

#constex1.py
class Employee:
    def __init__(self): # Default OR Parameter Less Constructor
        print("I am from Constructor")
        self.eno=10
        self.ename="RS"

#main program
e=Employee() # Object Creation--PVM ---calls Constructor
print("content of e =",e.__dict__)

#constex2.py
class Employee:
    def __init__(self,empno,empname): # Parametrized Constructor
        print("I am from Constructor")
        self.eno=empno
        self.ename=empname
```

```
#main program
e1=Employee(10,"RS") # Object Creation--PVM ---calls Constructor
print("content of e1=",e1.__dict__)
e2=Employee(20,"TR") # Object Creation--PVM ---calls Constructor
print("content of e2=",e2.__dict__)
```

```
#program for Demonstrating Default Constructor
```

```
#DefaultConstEx1.py
class Test:
    def __init__(self):
        print("i am from Default Constructor:")
        self.a=10
        self.b=20
```

```
#main program
```

```
t1=Test() # Object Creation--PVM Calls Default Constructor
print("Content of t1=",t1.__dict__)
t2=Test() # Object Creation--PVM Calls Default Constructor
print("Content of t2=",t2.__dict__)
t3=Test() # Object Creation--PVM Calls Default Constructor
print("Content of t3=",t3.__dict__)
```

```
#program for Demonstrating Parameterized Constructor
```

```
#ParameterizedConstEx1.py
class Test:
    def __init__(self,x,y):
        print("i am from Parameterized Constructor:")
        self.a=x
        self.b=y
```

```
#main program
```

```
t1=Test(10,20) # Object Creation--PVM Calls Parameterized Constructor
print("Content of t1=",t1.__dict__)
t2=Test(100,200) # Object Creation--PVM Calls Parameterized Constructor
print("Content of t2=",t2.__dict__)
t3=Test(1000,2000) # Object Creation--PVM Calls Parameterized Constructor
print("Content of t3=",t3.__dict__)
```

```
#program for Demonstrating _____Default Constructor
```

```
#DefaultParamConstEx1.py
class Test:
    def __init__(self,x=10,y=20):
        print("i am from Default Parameterized Constructor:")
        self.a=x
        self.b=y
```

```

print("Val of a={}".format(self.a))
print("Val of b={}".format(self.b))
print("-----")

#main program
print("Object t1 values")
t1=Test() # Object Creation--PVM Calls Default Constructor
print("Object t2 values")
t2=Test(100,200) # Object Creation--PVM Calls Paramtereized Constructor
print("Object t3 values")
t3=Test(1000)# Object Creation--PVM Calls Paramtereized Constructor
print("Object t4 values")
t4=Test(y=1000)# Object Creation--PVM Calls Paramtereized Constructor
print("Object t5 values")
t5=Test(y=500,x=100)# Object Creation--PVM Calls Paramtereized Constructor

```

===== 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.
=>The Purpose of creating an object is that "To store Data".
=>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
o=Student()
Example:- create an object Employee
e=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

```
#StudentDBWithOOPs.py
import cx_Oracle
class Student:
    def __init__(self):
        self.sno=int(input("Enter Student Number:"))
        self.sname=input("Enter Student Name:")
        self.marks=float(input("Enter Student marks:"))
    def savestuddata(self):
        try:
            con=cx_Oracle.connect("system/manager@localhost/xe")
            cur=con.cursor()
            cur.execute("insert into student
values(%d,'%s',%f)"%(self.sno,self.sname,self.marks))
            con.commit()
            print("{} Student Record Inserted Successfully".format(cur.rowcount))
        except cx_Oracle.DatabaseError as db:
            print("Problem in DB",db)

#main program
s=Student() # Object Creation
s.savestuddata()
```

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 initializing the object.

=>When the program execution is completed, GC calls its own destructor to de-allocate the memory space of objects present in program 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 the control of Programmer but it always maintained and managed 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.

#Program for demonstrating Destructor Concept

#non-destex1.py

class Emp:

```
def __init__(self,eno,ename):
    print("-"*50)
    self.eno=eno
    self.ename=ename
    print("Emp No:{}\nEmp Name:{}".format(self.eno,self.ename))
    print("-"*50)
```

```

#main program
print("Program Execution Started:")
e1=Emp(10,"RS") # Object creation--PVM--calls--Constructor
e2=Emp(20,"TR") # Object creation--PVM--calls--Constructor
e3=Emp(30,"KV") # Object creation--PVM--calls--Constructor
print("Program Execution Ended:")
-----
#Program for demonstrating Destructor Concept
#destex1.py
import time
class Emp:
    def __init__(self,eno,ename): # Parameterized Constructor
        print("-"*50)
        self.eno=eno
        self.ename=ename
        print("Emp No:{}\nEmp Name:{}".format(self.eno,self.ename))
        print("-"*50)
    def __del__(self):
        print("GC calls __del__() for de-allocatting Current Object Memory
space")

#main program
print("Program Execution Started:")
e1=Emp(10,"RS") # Object creation--PVM--calls--Constructor
e2=Emp(20,"TR") # Object creation--PVM--calls--Constructor
e3=Emp(30,"KV") # Object creation--PVM--calls--Constructor
print("Program Execution Ended:")
time.sleep(5)
-----
#Program for demonstrating Destructor Concept
#destex2.py
import sys,time
class Emp:
    def __init__(self,eno,ename): # Parameterized Constructor
        print("-"*50)
        self.eno=eno
        self.ename=ename
        print("Emp No:{}\nEmp Name:{}".format(self.eno,self.ename))
        print("-"*50)
    def __del__(self):
        print("-"*50)
        print("GC calls __del__() for de-allocatting Current Object Memory
space")
        global totmemspace
        print("{} Memory Space Deleted:".format(sys.getsizeof(self)))
        totmemspace=totmemspace-sys.getsizeof(self)
        print("Now Available Memory Space:{}".format(totmemspace))
        print("-"*50)

#main program

```

```

print("Program Execution Started:")
e1=Emp(10,"RS") # Object creation--PVM--calls--Constructor
print("Object e1 memory space=",sys.getsizeof(e1))
e2=Emp(20,"TR") # Object creation--PVM--calls--Constructor
print("Object e2 memory space=",sys.getsizeof(e2))
e3=Emp(30,"KV") # Object creation--PVM--calls--Constructor
print("Object e3 memory space=",sys.getsizeof(e3))
totmemspace=sys.getsizeof(e1)+sys.getsizeof(e2)+sys.getsizeof(e3)
print("TOTAL MEMORY SPACE:{}".format(totmemspace))
print("Program Execution Ended:")
time.sleep(5)
#When the program execution is completed, GC removes the memory space of all
the objects of current program automatically BY CALLING IT DESTRUCTOR and
This type of Garbage Collection is Called AUTOMATIC GARBAGE COLLECTION.
-----
```

```

#Program for demonstrating Destructor Concept
#destex3.py
import time
class Emp:
    def __init__(self,eno,ename): # Parameterized Constructor
        print("-"*50)
        self.eno=eno
        self.ename=ename
        print("Emp No:{}\nEmp Name:{}".format(self.eno,self.ename))
        print("-"*50)
    def __del__(self):
        print("GC calls __del__() for de-allocating Current Object Memory
space")
#main program
print("Program Execution Started:")
e1=Emp(10,"RS") # Object creation--PVM--calls--Constructor
print("No longer Interested to use e1 object")
time.sleep(2)
del e1 # Here Forcefully GC calls __del__() -- This type garbage Collection is called
Forceful Garbage Collection
e2=Emp(20,"TR") # Object creation--PVM--calls--Constructor
e3=Emp(30,"KV") # Object creation--PVM--calls--Constructor
print("Program Execution Ended:")
time.sleep(3)
#here e2 and e3 objects collected by GC automatically-
-----
```

```

#Program for demonstrating Destructor Concept
#destex4.py
import time
class Emp:
    def __init__(self,eno,ename): # Parameterized Constructor
        print("-"*50)
        self.eno=eno
        self.ename=ename
        print("Emp No:{}\nEmp Name:{}".format(self.eno,self.ename))
```

```

        print("-"*50)
    def __del__(self):
        print("GC calls __del__() for de-allocatting Current Object Memory
space")
#main program
print("Program Execution Started:")
e1=Emp(10,"RS") # Object creation--PVM--calls--Constructor
print("No longer Interested to use e1 object")
time.sleep(3)
del e1 # Here Forcefully GC calls __del__() -- This type garbage Collection is called
Forceful Garbage Collection
e2=Emp(20,"TR") # Object creation--PVM--calls--Constructor
print("No longer Interested to use e2 object")
time.sleep(3)
del e2 # Here Forcefully GC calls __del__() -- This type garbage Collection is called
Forceful Garbage Collection
e3=Emp(30,"KV") # Object creation--PVM--calls--Constructor
print("No longer Interested to use e3 object")
time.sleep(3)
del e3 # Here Forcefully GC calls __del__() -- This type garbage Collection is called
Forceful Garbage Collection
print("Program Execution Ended:")
time.sleep(3)
-----
```

```

#Program for demonstrating Destructor Concept
#destex5.py
import time
class Emp:
    def __init__(self,eno,ename): # Parameterized Constructor
        print("-"*50)
        self.eno=eno
        self.ename=ename
        print("Emp No:{}\nEmp Name:{}".format(self.eno,self.ename))
        print("-"*50)
    def __del__(self):
        print("GC calls __del__() for de-allocatting Current Object Memory
space")

#main program
print("Program Execution Started:")
e1=Emp(10,"RS") # Object creation--PVM--calls--Constructor
print("No longer Interested to use e1 object")
time.sleep(3)
e1=None # Here Forcefully GC calls __del__() -- This type garbage Collection is
called Forceful Garbage Collection
e2=Emp(20,"TR") # Object creation--PVM--calls--Constructor
print("No longer Interested to use e2 object")
time.sleep(3)
```

```

e2=None # Here Forcefully GC calls __del__() -- This type garbage Collection is
called Forceful Garbage Collection
e3=Emp(30,"KV") # Object creation--PVM--calls--Constructor
print("No longer Interested to use e3 object")
time.sleep(3)
e3=None # Here Forcefully GC calls __del__() -- This type garbage Collection is
called Forceful Garbage Collection
print("Program Execution Ended:")
time.sleep(3)

-----
#Program for demonstrating Destructor Concept
#destex6.py
import time
class Emp:
    def __init__(self,eno,ename): # Parameterized Constructor
        print("-"*50)
        self.eno=eno
        self.ename=ename
        print("Emp No:{}\nEmp Name:{}".format(self.eno,self.ename))
        print("-"*50)
    def __del__(self):
        print("GC calls __del__() for de-allocatting Current Object Memory
space")

#main program
print("Program Execution Started:")
e1=Emp(10,"RS") # Object creation--PVM--calls--Constructor
e2=e1 #Deep Copy
e3=e1 #Deep Copy
print("Program Execution ended:")

-----
#Program for demonstrating Destructor Concept
#destex7.py
import time
class Emp:
    def __init__(self,eno,ename): # Parameterized Constructor
        print("-"*50)
        self.eno=eno
        self.ename=ename
        print("Emp No:{}\nEmp Name:{}".format(self.eno,self.ename))
        print("-"*50)
    def __del__(self):
        print("GC calls __del__() for de-allocatting Current Object Memory
space")

#main program
print("Program Execution Started:")
e1=Emp(10,"RS") # Object creation--PVM--calls--Constructor
e2=e1 # Deep Copy
e3=e1 # Deep Copy

```

```

print("No longer Interested to use e1 object")
time.sleep(3)
e1=None # Here GC will not call __del__() bcoz e2 and e3 still pointing to the
object
print("No longer Interested to use e2 object")
time.sleep(3)
del e2 # Here GC will not call __del__() bcoz e3 still pointing to the object
print("No longer Interested to use e3 object")
time.sleep(3)
e3=None # Here Forcefully GC calls __del__() bcoz no objects points to the created
object.
print("Program Execution Ended:")
time.sleep(3)
-----
```

```

#Program for demonstrating Garbage Collection execution
#gce1.py
import gc
print("Program execution Started:")
print("Line:4-->is GC running-->",gc.isenabled()) # True
a=10
b=20
print("Val of a={}".format(a))
print("Val of b={}".format(b))
gc.disable()
print("Line:11-->is GC running-->",gc.isenabled()) # False
c=a+b
d=a-b
e=a*b
print("Sum={}".format(c))
print("Sub={}".format(d))
gc.enable()
print("Line:18-->is GC running-->",gc.isenabled()) # True
print("Mul={}".format(e))
gc.disable()
print("Line:21-->is GC running-->",gc.isenabled()) # False
print("Program execution Ended:")
-----
```

```

#Program for demonstrating Destructor Concept
#gce2.py
import sys,time,gc
class Emp:
    def __init__(self,eno,ename): # Parameterized Constructor
        print("-"*50)
        self.eno=eno
        self.ename=ename
        print("Emp No:{}\nEmp Name:{}".format(self.eno,self.ename))
        print("-"*50)
    def __del__(self):
        print("-"*50)
-----
```

```

print("GC calls __del__() for de-allocatting Current Object Memory
space")
global totmemspace
print("{} Memory Space Deleted:".format(sys.getsizeof(self)))
totmemspace=totmemspace-sys.getsizeof(self)
print("Now Available Memory Space:{}".format(totmemspace))
print("-"*50)

#main program
print("Program Execution Started:")
print("Line:22-->Is GC running:{}".format(gc.isenabled())) # True
print("-----")
e1=Emp(10,"RS") # Object creation--PVM--calls--Constructor
print("Object e1 memory space=",sys.getsizeof(e1))
e2=Emp(20,"TR") # Object creation--PVM--calls--Constructor
print("Object e2 memory space=",sys.getsizeof(e2))
e3=Emp(30,"KV") # Object creation--PVM--calls--Constructor
gc.disable()
print("Line:30-->Is GC running:{}".format(gc.isenabled())) # False
print("-----")
print("Object e3 memory space=",sys.getsizeof(e3))
totmemspace=sys.getsizeof(e1)+sys.getsizeof(e2)+sys.getsizeof(e3)
print("TOTAL MEMORY SPACE:{}".format(totmemspace))
print("Program Execution Ended:")
time.sleep(10)
-----
```

#When the program execution is completed, GC removes the memory space of all the objects of current program automatically BY CALLING IT DESTRUCTOR and This type of Garbage Collection is Called AUTOMATIC GARBAGE COLLECTION.

===== 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
```

Data Abstraction:

=>The Process of retrieving / extracting Essential Details without considering Hidden Details is called Data Abstraction.

Note:- We can't apply Data Encapsulation on Constructors in Python but whose Initialized Data Members can be encapsulated.

#Program for Demonstrating Data Encapsulation

#Account1.py--File Name and Module name

class Account:

```
def __init__(self):
    self.__acno=100
    self.cname="Rossum"
    self.__bal=4.5
    self.__pin=1234
    self.bname="SBI"
def dispvalues(self):
    print("Account Number:{}".format(self.__acno))
```

#Program for Demonstrating Data Encapsulation

#Account2.py--File Name and Module name

class Account:

```
def getaccdetails(self):
```

```

    self.__acno=100
    self.cname="Rossum"
    self.__bal=4.5
    self.__pin=1234
    self.bname="SBI"

```

```

#Program for Demonstrating Data Encapsulation
#Account3.py--File Name and Module name
class Account:
    def __getaccdetails(self):
        self.acno=100
        self.cname="Rossum"
        self.bal=4.5
        self.pin=1234
        self.bname="SBI"

```

```

#Program for Demonstrating Data Encapsulation
#Account4.py--File Name and Module name
class Account:
    def _____init__(self):
        self.acno=100
        self.cname="Rossum"
        self.bal=4.5
        self.pin=1234
        self.bname="SBI"

```

```

#program for demonstrating Data Abstraction
#OtherProg1.py
from Account1 import Account
ac=Account()
#print("Account Number:{}" .format(ac.acno)) can't access
print("Account Holder Name:{}" .format(ac.cname))
#print("Account Balance:{}" .format(ac.bal)) can't access
#print("Account PIN:{}" .format(ac.pin)) can't access
print("Account Branch Name:{}" .format(ac.bname))

```

```

#program for demonstrating Data Abstraction
#OtherProg2.py
from Account2 import Account
ac=Account()
ac.getaccdetails()

```

```

#print("Account Number:{}" .format(ac.acno)) can't access
print("Account Holder Name:{}" .format(ac.cname))
#print("Account Balance:{}" .format(ac.bal))
#print("Account PIN:{}" .format(ac.pin))
print("Account Branch Name:{}" .format(ac.bname))

```

```
#program for demonstrating Data Abstraction
```

```

#OtherProg3.py
from Account3 import Account
ac=Account()
print("Content of ac=",ac.__dict__)
#ac.getaccdetails() can't access
#print("Account Number:{}".format(ac.acno)) can't access
#print("Account Holder Name:{}".format(ac.cname)) can't access
#print("Account Balance:{}".format(ac.bal)) can't access
#print("Account PIN:{}".format(ac.pin)) can't access
#print("Account Branch Name:{}".format(ac.bname)) can't access
-----
#program for demonstrating Data Abstraction
#OtherProg4.py
from Account4 import Account
ac=Account()
ac.___init__()
print("Account Number:{}".format(ac.acno))
print("Account Holder Name:{}".format(ac.cname))
print("Account Balance:{}".format(ac.bal))
print("Account PIN:{}".format(ac.pin))
print("Account Branch Name:{}".format(ac.bname))
-----
#Employee.py---File name and Module name
class Employee:
    def __init__(self,eno,ename,sal):
        self.eno=eno
        self.ename=ename
        self.sal=sal
    def dispempdata(self):
        print("\t{}\t{}\t{}".format(self.eno,self.ename,self.sal))
-----
#EmpPick.py----File Name----Program-(A)
from Employee import Employee
import pickle
class EmpPick:
    def saveempdata(self):
        with open("emppick.data","ab") as fp:
            while(True):
                print("-"*50)
                #Read Emp Data
                empno=int(input("Enter Employee Number:"))
                empname=input("Enter Employee Name:")
                empsal=float(input("Enter Employee Salary:"))
                print("-"*50)
                #create an Employee Class object
                eo=Employee(empno,empname,empsal) # Object
                                                creation calls Parameterized Const
                #save the eo object into file
                pickle.dump(eo,fp)
                print("Employee Data Saved Successfully as a Record in")

```

```

        a file:")
print("-"*50)
ch=input("Do u want to insert another emp
         record(yes/no):")
if(ch.lower() == "no"):
    break

#main program
ep= EmpPick()
ep. saveempdata()

-----#EmpUnPick.py---File Name --Program-(B)
import pickle
class EmpUnPick:
    def reademprecords(self):
        try:
            with open("emppick.data","rb") as fp:
                print("-----")
                while(True):
                    try:
                        obj=pickle.load(fp)
                        obj.dispempdata()
                    except EOFError:
                        print("-----")
                        break
        except FileNotFoundError:
            print("File does not exist")

#main program
eup=EmpUnPick()
eup.reademprecords()

-----#program for cal factorial of a number by using classes and objects
#FactEx1.py
class Math:
    def __init__(self):
        self.n=int(input("Enter a Number for cal Factorial:"))
    def factorial(self):
        if(self.n<0):
            print("{} is Invalid Input and Can't cal Factorial for -ve
Values".format(self.n))
        else:
            f=1
            for i in range(1,self.n+1):
                f*=i # f=f*i
            else:
                print("Factorial({})={}".format(self.n,f))

#main program

```

```
m=Math()
m.factorial()
```

```
#program for cal factorial of a number by using classes and objects
#FactEx2.py
class Math:
    def __init__(self):
        self.n=int(input("Enter a Number for cal Factorial:"))
    def factorial(self):
        if(self.n<0):
            print("{} is Invalid Input and Can't cal Factorial for -ve
Values".format(self.n))
        else:
            f=1
            for i in range(1,self.n+1):
                f*=i # f=f*i
            else:
                print("Factorial({})={}".format(self.n,f))

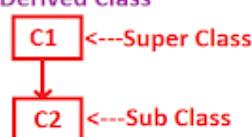
#main program
Math().factorial() # calling Instance Method with NameLess object
```

1) Single Inheritance

Definition:

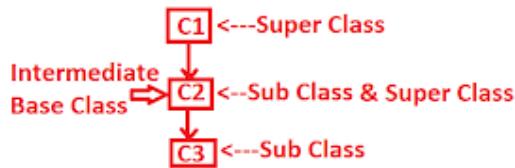
This Inheritance contains Single Base class and Single Derived Class

Diagram:



2. Multi Level Inheritance

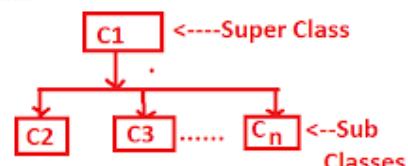
Definition: This Inheritance contains single base class , single derived class and Intermediate base class(es).



3. Hierarchical Inheritance

Definition: This Inheritance Contains Single Super Class and Multiple Sub Classes

Diagram:



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. Redundency (Duplication) of the code is minimized.
-
-

Types of Inheritances

=>A Type of Inheritance is a model / Paradigm , which makes us to understand how the features are Inherited from Base Class into Derived Class.

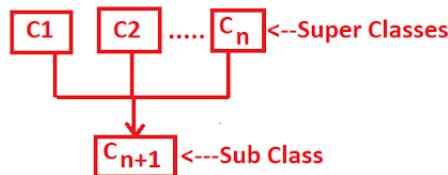
=>In Python Programming, we have 5 types of Inheritances. They are

1. Single Inheritance
2. Multi Level Inheritance
3. Hierarchical Inheritance
4. Multiple Inheritance
5. Hybrid Inheritance

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:

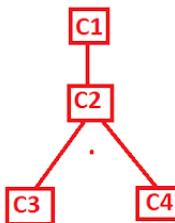
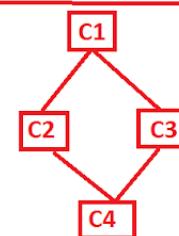


Diagram2:



Inheriting the features of Base Class into Derived Class

=>To Inherit the features of Base or Super Class into Derived or Sub class, we use the following syntax:

Syntax:

```
class <clsname-1>:
```

```
class <clsname-2>:
```

```
class <clsname-n>:
```

```
class <clsname-n+1>(clsname-1,clsname-2,...clsname-n):
```

Explanation:

=>Here <clsname-1,clsname-2,...clsname-n> are called Base Classes.

=><clsname-n+1> is called Derived Class.

=>The Data Members, Methods and Constructors (Features) of Base Classes (

clsname-1,clsname-2,...clsname-n) are Inherited logically into Derived Class clsname-n+1 and we can access them either by using Object Name (Instance Features) or by using class name (Class Level and Static Features).

=>When we develop any Inheritance Based Application, It is always recommended to create an object of Bottom Most Derived Class bcoz It contains Features of Intermediate Base Class(es) and Top Most Base Class.

=>For Every Class in Python, there exist an implicit Pre-defined Super class called "object" and It provides Garbage Collection Facility to its sub classes.

#Program for Demonstrating class and objects without Inheritance

#non-inhex.py

class C1:

```
    def disp(self):
        print("C1--disp())")
```

class C2:

```
    def show(self):
        print("C2--show())")
```

#main program

o1=C1()

o2=C2()

o1.disp()

o2.show()

#Program for Demonstrating Inheritance Concept

#InhProg1.py

class C1:

```
    def disp(self):
        print("C1--disp())")
```

class C2(C1):

```
    def show(self):
        print("C2--show())")
```

#main program

o2=C2()

o2.disp()

o2.show()

#Program for Demonstrating Inheritance Concept

#InhProg2.py

class C1:

```
    def disp(self):
        print("C1--disp())")
```

class C2(C1):

```
    def show(self):
        self.disp() # calling base class method from derived class
        print("C2--show())")
```

#main program

o2=C2()

o2.show()

```
#Program for Demonstrating Inheritance Concept
#InhProg3.py
class C1:
    def disp(self):
        print("C1--disp()")

class C2(C1):
    def show(self):
        print("C2--show()")
class C3(C2):
    def hello(self):
        print("welcome to inheritance Concept-C3")
#main program
o3=C3()
o3.hello()
o3.disp()
o3.show()
```

```
#Program for Demonstrating Inheritance Concept
#InhProg4.py
class C1:
    def disp(self):
        print("C1--disp()")
class C2(C1):
    def show(self):
        print("C2--show()")
class C3(C2):
    def hello(self):
        print("welcome to inheritance Concept-C3")
        self.show()
        self.disp()
#main program
o3=C3()
o3.hello()
```

```
#Program for Demonstrating Inheritance Concept
#InhProg5.py
class C1:
    def disp(self):
        print("C1--disp()")

class C2(C1):
    def show(self):
        print("C2--show()")
        self.disp() # Calling disp() of C1 from show() of C2
class C3(C2):
    def hello(self):
```

```
print("welcome to inheritance Concept-C3")
self.show() # Calling show() of C2 from hello() of C3
```

```
#main program
o3=C3()
o3.hello()
```

```
#InhProg6.py
class Parent:
    def getParentProperty(self):
        self.pp=3.4

class Child(Parent):
    def getChildProperty(self):
        self.cp=1.0
    def childTotProperty(self):
        self.totprop=self.pp+self.cp

    def dispproperty(self):
        print("-----")
        print("Child Peroperty:{}".format(self.cp))
        print("Parent Peroperty:{}".format(self.pp))
        print("Total Peroperty:{}".format(self.totprop))
        print("-----")
```

```
#main program
c=Child()
c.getChildProperty()
c.getParentProperty()
c.childTotProperty()
c.dispproperty()
```

```
#InhProg7.py
class Parent:
    def getParentProperty(self):
        self.pp=float(input("Enter Parent Property:"))

class Child(Parent):
    def getChildProperty(self):
        self.cp=float(input("Enter Child Property:"))
    def childTotProperty(self):
        self.totprop=self.pp+self.cp

    def dispproperty(self):
        print("-----")
```

```

print("Child Peroperty:{}".format(self.cp))
print("Parent Peroperty:{}".format(self.pp))
print("Total Peroperty:{}".format(self.totprop))
print("-----")
#main program
c=Child()
c.getChildProperty()
c.getParentProperty()
c.childTotProperty()
c.dispproperty()

-----
#InhProg8.py
class Parent:
    def getParentProperty(self):
        self.pp=float(input("Enter Parent Property:"))

class Child(Parent):
    def getChildProperty(self):
        self.cp=float(input("Enter Child Property:"))
    def childTotProperty(self):
        self.totprop=self.pp+self.cp

    def dispproperty(self):
        self.getParentProperty()
        self.getChildProperty()
        self.childTotProperty()
        print("-----")
        print("Child Peroperty:{}".format(self.cp))
        print("Parent Peroperty:{}".format(self.pp))
        print("Total Peroperty:{}".format(self.totprop))
        print("-----")

#main program
c=Child()
c.dispproperty()

-----
#UniversityCollegeStudentEx1.py
class Univ:
    def getunivdet(self):
        self.uname=input("Enter University Name:")
        self.uloc=input("Enter University Location:")
    def dispunivdet(self):
        print("-"*50)
        print("University Name:{}".format(self.uname))
        print("University Location:{}".format(self.uloc))
        print("-"*50)

class College(Univ):
    def getcolldet(self):
        self.cname=input("Enter College Name:")

```

```

        self.cloc=input("Enter College Location:")
def dispcolldet(self):
    print("-"*50)
    print("College Name:{}".format(self.cname))
    print("College Location:{}".format(self.cloc))
    print("-"*50)
class Student(College):
    def getstuddet(self):
        self.sno=int(input("Enter Student Number:"))
        self.sname=input("Enter Student Name:")
        self.crs=input("Enter Student Course:")
    def dispstuddet(self):
        print("-"*50)
        print("Student Number:{}".format(self.sno))
        print("Student Name:{}".format(self.sname))
        print("Student Course:{}".format(self.crs))
        print("-"*50)

#main program
s=Student()
s.getstuddet()
s.getcolldet()
s.getunivdet()
s.dispunivdet()
s.dispcolldet()
s.dispstuddet()

#UniversityCollegeStudentEx2.py
from student import Student
s=Student()
s.getstuddet()
s.dispstuddet()
s.savestuddata()

```

```

#UniversityCollegeStudentEx3.py
class Univ:
    def getunivdet(self):
        self.uname=input("Enter University Name:")
        self.uloc=input("Enter University Location:")
    def dispunivdet(self):
        print("-"*50)
        print("University Name:{}".format(self.uname))
        print("University Location:{}".format(self.uloc))
        print("-"*50)
class College:
    def getcolldet(self):
        self.cname=input("Enter College Name:")

```

```

        self.cloc=input("Enter College Location:")
def dispcolldet(self):
    print("-"*50)
    print("College Name:{}".format(self cname))
    print("College Location:{}".format(self.cloc))
    print("-"*50)
class Student(College,Univ): # Multiple Inheritance
    def getstuddet(self):
        self.sno=int(input("Enter Student Number:"))
        self.sname=input("Enter Student Name:")
        self.crs=input("Enter Student Course:")
        self.getcolldet()
        self.getunivdet()
    def dispstuddet(self):
        self.dispunivdet()
        self.dispcolldet()
        print("-"*50)
        print("Student Number:{}".format(self.sno))
        print("Student Name:{}".format(self.sname))
        print("Student Course:{}".format(self.crs))
        print("-"*50)

#main program
s=Student()
s.getstuddet()
s.dispstuddet()
#College.py--File Name and Module name
from University import Univ
class College(Univ):
    def getcolldet(self):
        self.cname=input("Enter College Name:")
        self.cloc=input("Enter College Location:")
    def dispcolldet(self):
        print("-"*50)
        print("College Name:{}".format(self cname))
        print("College Location:{}".format(self.cloc))
        print("-"*50)
#student.py--File Name and Module Name
from College import College
import mysql.connector
class Student(College):
    def getstuddet(self):
        self.sno=int(input("Enter Student Number:"))
        self.sname=input("Enter Student Name:")
        self.crs=input("Enter Student Course:")
        self.getcolldet()
        self.getunivdet()
    def dispstuddet(self):
        self.dispunivdet()
        self.dispcolldet()

```

```

print("-"*50)
print("Student Number:{}".format(self.sno))
print("Student Name:{}".format(self.sname))
print("Student Course:{}".format(self.crs))
print("-"*50)
def savestuddata(self):
    try:

        con=mysql.connector.connect(host="localhost",user="root",passwd="root",
database="batch9am")
        cur=con.cursor()
        cur.execute("insert into studcolluniv
values(%d,'%s','%s','%s','%s','%s')"% (self.sno,self.sname,self.crs,self.cname,sel
f.cloc,self.uname,self.uloc))
        con.commit()
        print("\nStudent Data Saved in Table--Verify")
    except mysql.connector.DatabaseError as db:
        print("Problem in MySQL:",db)
-

```

#University.py--File Name and Module Name
class Univ:

```

def getunivdet(self):
    self.uname=input("Enter University Name:")
    self.uloc=input("Enter University Location:")
def dispunivdet(self):
    print("-"*50)
    print("University Name:{}".format(self.uname))
    print("University Location:{}".format(self.uloc))
    print("-"*50)

```

===== Polymorphism in Python =====

=>Polymorphism is one of the distinct features of OOPs
=>The purpose of Polymorphism is that "Efficient Utilization of 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)

#Non-InhEx.py (Without Inheritance Ex)

```
class Circle:
    def draw1(self):
        print("Drawing Circle")

class Rect:
    def draw2(self):
        print("Drawing Rect")

# main program
R=Rect()
R.draw2()
C=Circle()
C.draw1()
```

#Non-PolyEx.py (Inheritance Ex)

```
class Circle:
    def draw1(self):
        print("Drawing Circle")
class Rect(Circle):
    def draw2(self):
        print("Drawing Rect")
# main program
R=Rect()
R.draw2()
R.draw1()
```

```
#PloyEx1.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
print("w.r.t Square Class")
S=Square()
S.draw()

#PloyEx2.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")
        #super().super().draw()----Error
        Circle.draw(self)
        super().draw() # OR Rect.draw(self)

#main program
print("w.r.t Square Class")
S=Square()
S.draw()
```

===== Numpy Module =====

Index

- ====
- =>Purpose of Numpy
- =>History of Numpy
- =>Setup of numpy Module
- =>Advantages of Numpy Module
- =>ndarray of numpy module
- =>Similarities and Differences between List and ndarray of numpy

=>Number of approaches to create an object of ndarray of numpy
 =>Basic Indexing and Slicing Operations on ndarray
 =>Advanced Indexing and Slicing Operations on ndarray
 =>Random Elements Selection of ndarray
 =>Filtering Elements Selection of ndarray

=>Matrix Operations OR Arithmetic Operations of ndarray
 =>Statistical Operations on ndarray
 =>Adding an Element to ndarray
 =>deleting an Element from ndarray
 =>Updating the Elements of ndarray

===== 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 where as an object of ndarray of numpy can store only similar type of values(even we store different values, internally they are treated as similar type by treating all values of type "object").
- =>On the object of list, we can't perform Vector Based Operations. where as on the object of ndarray, we can perform Vector based operations.
- =>In large sampling of data, List based applications takes more memory space where as ndarray object takes less memory space.
- =>List based applications are not effiecient bcoz list object values takes more time to extract or retrive (they are available at different Address) where as numpy based applications are efficient bcoz of ndarray object values takes less to time to extract or retrive(they are available at same Address / clustered).
- =>List object can't perform complex mathematical operations where as an object of ndarray can perform complex mathematical operations.

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 ndrray object.
=>object represents any Traditional Python Objects
=>dtype represents any numpy data type such as
int8,int16,int32,float16, float 32, float64,...etc (Internal data 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(2,2,2,2)-----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],
         [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]]])

-----
```

```

        [[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 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 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]])

```

```

[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'>
```

7. numpy.hstack()

=>numpy().hstack stacks arrays horizontally.

=>All the input arrays must have same number of dimensions, but the nested arrays of different input arrays can have different number of columns. This is because the horizontal stack is not restricted to the vertical alignments.

```
varname=numpy.hstack(ndarrayobj1,(ndarrayobj2))
```

Examples:

```
import numpy as np
a = np.array([1, 2, 3])
b = np.array([4, 5, 6])
np.hstack((a,b))-----# [1, 2, 3, 4, 5, 6]
```

```
import numpy as np
a = np.array([[1, 2], [3, 4]])
b = np.array([[4, 5, 6], [7, 8, 9]])
np.hstack((a,b)) # [[1, 2, 4, 5, 6],
                  [3, 4, 7, 8, 9]]
```

numpy.vstack() stacks arrays vertically, and the number of columns of input arrays must be the same. This is because NumPy array requires all the nested arrays to have the same size. If you try to vertically stack 2 arrays with different number of columns we get ValueError.

Syntax:

```
varname=numpy.vstack(ndarrayobj1,(ndarrayobj2))
```

Examples:

```
import numpy as np
a = np.array([1, 2, 3])
b = np.array([4, 5, 6])
np.vstack((a,b))-----# [[1 2 3],
                      [4 5 6]]
```

```
import numpy as np
a = np.array([[1, 2], [3,4]])
b = np.array([[4, 5], [5,6]])
np.vstack((a,b)) # [[1, 2],
                  [3, 4],
                  [4, 5],
                  [5, 6]]
```

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:- 1ndndrrayobj[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:- 2ndndrrayobj[Row Index, Column Index]

Syntax:- 2ndndrrayobj[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]]])
>>> 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])
-----
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])
-----
```

**Numpy--Selecting the elements based on condition
(OR)
Creating Filter Directly From ndarray**

=>To select any element from ndarray object, we have two approaches. They are

Approach-1:

=>Prepare Boolean Array (It contains True or False. True represents Condition satisfied and False represents Condition not satisfied)

Syntax:- varname=ndarrayobject with condition

varname is called boolean array.

=>Pass the Boolean Array to the ndarray object. so that we can get those elements from ndarray which satisfies with the entry True(or) we can get those elements from ndarray corresponding True entries of Boolean array.

Syntax: ndarray [Boolean Array]

Approach-2:

=>In this approach, we directly pass Boolean array values to the ndarray for getting required elements based on condition.

Syntax: ndarray[ndarrayobject with condition]

Examples:

Q1) Select the Positive Elements from ndarray

```
>>> import numpy as np
>>> l=[10,21,-34,23,-45,30,-40]
>>> print(l)-----[10, 21, -34, 23, -45, 30, -40]
>>> a=np.array(l)
>>> a-----array([ 10, 21, -34, 23, -45, 30, -40])
>>> b=a>0 # Boolean Array
>>> print(b)---[ True  True False  True False  True False]
>>> a[b]-----array([10, 21, 23, 30])
```

```
>>> a[ a>0 ]-----array([10, 21, 23, 30])
```

Q2) Select the Negative Elements from ndarray

```
>>> l=[10,21,-34,23,-45,30,-40]
>>> a=np.array(l)
>>> a-----array([ 10, 21, -34, 23, -45, 30, -40])
>>> b=a<0 # Boolean Array
>>> b---- array([False, False, True, False, True, False, True])
>>> a[b]----- array([-34, -45, -40])
=====OR=====
>>> a[a<0]----- array([-34, -45, -40])
```

Q3) Select the Even and Odd Elements from ndarray

```
>>> a=np.array([11,20,33,31,41,47,46,12,13])
>>> a-----array([11, 20, 33, 31, 41, 47, 46, 12, 13])
>>> a[a%2==0]-----array([20, 46, 12])
>>> a[a%2!=0]-----array([11, 33, 31, 41, 47, 13])
```

```
>>> a=np.array([10,20,30,40,50,60,70,80,90])
>>> b=a.reshape(3,3)
>>> b
array([[10, 20, 30],
       [40, 50, 60],
       [70, 80, 90]])
```

#Get Multiples of 3

```

>>> m3=(b%3==0)
>>> m3 #-----Boolean array
      array([[False, False, True],
             [False, False, True],
             [False, False, True]])
>>> b[m3]-----array([30, 60, 90])
=====OR=====
>>> b[b%3==0]-----array([30, 60, 90])

```

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)
-----
```

===== Numpy—Statistical Operations =====

=>On the object of ndarray, we can perform the following Statistical Operations .

- a) amax()
- b) amin()
- c) mean()
- d) median()
- e) var()
- f) std()

=>These operations we can perform on the entire matrix and we can also perform on columnwise (axis=0) and Rowwise (axis=1)

a) amax():

=>This function 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=np.amax(A)
>>> cmax=np.amax(A, axis=0)
>>> rmax=np.amax(A, axis=1)
>>> print("Max element=",max)-----Max element= 4
>>> print("Column Max elements=",cmax)---Column Max elements= [4 4 3]
>>> print("Row Max elements=",rmax)---Row Max elements= [3 4 4]
```

b) amin():

=>This functions obtains minimum element of the entire matrix.
=>Syntax1:- varname=numpy.amin(ndarrayobject)
=>Syntax2:- varname=numpy.amin(ndarrayobject, axis=0)-->obtains min elements on the basis of columns.
=>Syntax3:- varname=numpy.amin(ndarrayobject, axis=1)-->obtains min 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]]
>>> 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 calculate 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= 2.444444444444446
>>> 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= $\text{sqrt}(\text{mean}-\bar{x}) / \text{total number of elements}$
here ' \bar{x} ' represents each element of matrix.

Syntax1:- varname=numpy.var(ndarrayobject)

Syntax2:- varname=numpy.var(ndarrayobject, axis=0)

Syntax3:- varname=numpy.var(ndarrayobject, axis=1)

Examples:

```
>>> l1=[[1,2,3],[4,2,1],[3,4,2]]
>>> A=np.array(l1)
>>> print(A)
[[1 2 3]
 [4 2 1]
 [3 4 2]]
>>> vr=np.var(A)
>>> cvr=np.var(A, axis=0)
>>> rvr=np.var(A, axis=1)
>>> print("Variance=", vr)---Variance= 1.1358024691358024
>>> print("Column Variance=", cvr)---Column Variance= [1.55555556 0.88888889
0.66666667]
>>> print("Row Variance=", rvr)---Row Variance=[0.66666667 1.55555556
0.66666667]
```

f) std()

standard deviation=sqrt(var)

Syntax1:- varname=numpy.std(ndarrayobject)

Syntax2:- varname=numpy.std(ndarrayobject, axis=0)

Syntax3:- varname=numpy.std(ndarrayobject, axis=1)

Examples:

```
>>> l1=[[1,2,3],[4,2,1],[3,4,2]]
>>> A=np.array(l1)
>>> print(A)
[[1 2 3]
 [4 2 1]
 [3 4 2]]
>>> vr=np.var(A)
>>> cvr=np.var(A, axis=0)
>>> rvr=np.var(A, axis=1)
>>> print("Variance=", vr)---Variance= 1.1358024691358024
>>> print("Column Variance=", cvr)---Column Variance= [1.55555556 0.88888889
0.66666667]
>>> print("Row Variance=", rvr)---Row Variance=[0.66666667 1.55555556
0.66666667]
```

```
>>> sd=np.std(A)
>>> csd=np.std(A, axis=0)
>>> rsd=np.std(A, axis=1)
```

```
>>> print("std=",sd)---std= 1.0657403385139377
>>> print(" column std=",csd)--- column std= [1.24721913 0.94280904 0.81649658]
>>> print("Row std=",rsd)--Row std= [0.81649658 1.24721913 0.81649658]
```

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----append() =====

Numpy module in python, provides a function `numpy.append()` to add an element in a numpy array.

Syntax: `Varname=numpy.append(ndarrayobj, value)`

Example.

```
import numpy as np
#Create a Numpy Array of integers
arr = np.array([11, 2, 6, 7, 2])
# Add / Append an element at the end of a numpy array
new_arr = np.append(arr, 10)
print('New Array: ', new_arr)
print('Original Array: ', arr)
```

===== numpy---insert() =====

Numpy module in python, provides a function `numpy.insert()` to insert the element in ndarray at particular Index.

Syntax: `varname=np.insert(ndarray , index, value)`

Example.

```
import numpy as np
a=np.array([10,20,30,40])
```

```
a=np.array([10,20,30,40])
b=np.insert(a,1,35)
print(a)# array([10, 35, 20, 30, 40])
```

numpy--delete()

Python's Numpy library provides a method to delete elements from a numpy array based on index position i.e.

```
numpy.delete(ndarrayobj, index(es), axis=None)
```

ndarrayobj : Is an object of ndarray class

index(es): Represents either single Index position or list of index positions of items to be deleted from ndarrayobject.

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 2
arr = np.delete(arr, 2)
print('Modified Numpy Array by deleting element at index 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 indexes , pass the numpy array and list of indexes 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 indexes 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 .

=>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 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 and 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 row 1 and 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]

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]])

```

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

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.

=>DataFrame is a Part of Panel

===== 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,tuple,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
```

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
```

Ist=[[10,20,30,40],["RS","JS","MCK","TRV"]]

```
df=pd.DataFrame(Ist)
print(df)
      0  1  2  3
0 10 20 30 40
1 RS JS MCK TRV
```

```
Ist=[[10,'RS'],[20,'JG'],[30,'MCK'],[40,'TRA']]
df=pd.DataFrame(Ist)
print(df)
```

| | 0 1 |
|---|--------|
| 0 | 10 RS |
| 1 | 20 JG |
| 2 | 30 MCK |
| 3 | 40 TRA |

```
Ist=[[10,'RS'],[20,'JG'],[30,'MCK'],[40,'TRA']]
df=pd.DataFrame(Ist, index=[1,2,3,4],columns=['Rno','Name'])
print(df)
```

| Rno | Name |
|------|------|
| 1 10 | RS |
| 2 20 | JG |
| 3 30 | MCK |
| 4 40 | TRA |

```
tpl=( ("Rossum",75), ("Gosling",85), ("Travis",65), ("Ritche",95),("McKinney",60) )
df=pd.DataFrame(tpl, index=[1,2,3,4,5],columns=['Name','Age'])
print(df)
```

| | Name | Age |
|---|----------|-----|
| 1 | Rossum | 75 |
| 2 | Gosling | 85 |
| 3 | Travis | 65 |
| 4 | Ritche | 95 |
| 5 | McKinney | 60 |

Creating an object DataFrame by Using dict object

=>When we create an object of DataFrame by using Dict , all the keys are taken as Column Names and Values of Value are taken as Data.

Examples:

```
>>> import pandas as pd
>>>dictdata={"Names":["Rossum","Gosling","Ritche","McKinney"],"Subjects":["Python","Java","C","Pandas"],"Ages":[65,80,85,55] }
>>> df=pd.DataFrame(dictdata)
>>> print(df)
      Names Subjects   Ages
0  Rossum    Python     65
1  Gosling     Java     80
2  Ritche       C     85
3 McKinney    Pandas     55
>>> df=pd.DataFrame(dictdata,index=[1,2,3,4])
>>> print(df)
      Names Subjects   Ages
```

```

1 Rossum Python      65
2 Gosling Java       80
3 Ritche C           85
4 McKinney Pandas   55

```

Creating an object DataFrame by Using Series object

```

>>> import pandas as pd
>>> sdata=pd.Series([10,20,30,40])
>>> df=pd.DataFrame(sdata)
>>> print(df)
   0
0 10
1 20
2 30
3 40
>>> sdata=pd.Series({"IntMarks": [10,20,30,40], "ExtMarks": [80,75,65,50]})
>>> print(sdata)
IntMarks [10, 20, 30, 40]
ExtMarks [80, 75, 65, 50]
dtype: object
>>> df=pd.DataFrame(sdata)
>>> print(df)
   0
IntMarks [10, 20, 30, 40]
ExtMarks [80, 75, 65, 50]
>>> ddata={"IntMarks": [10,20,30,40], "ExtMarks": [80,75,65,50]}
>>> df=pd.DataFrame(ddata)
>>> print(df)
   IntMarks ExtMarks
0        10      80
1        20      75
2        30      65
3        40      50

```

Creating an object DataFrame by Using ndarray object

```

>>> import numpy as np
>>> l1=[[10,60],[20,70],[40,50]]
>>> a=np.array(l1)
>>> df=pd.DataFrame(a)
>>> print(df)
   0  1
0 10 60
1 20 70
2 40 50
>>> df=pd.DataFrame(a,columns=["IntMarks", "ExtMarks"])
>>> print(df)
   IntMarks ExtMarks
0        10      60

```

| | | |
|---|----|----|
| 1 | 20 | 70 |
| 2 | 40 | 50 |

e) By using CSV File(Comma Separated Values)

```
import pandas as pd1
df=pd1.read_csv("D:\KVR-JAVA\stud.csv")
print("type of df=",type(df)) #type of df= <class 'pandas.core.frame.DataFrame'>
print(df)
```

----- OUTPUT -----

| | stno | name | marks |
|---|------|---------|-------|
| 0 | 10 | Rossum | 45.67 |
| 1 | 20 | Gosling | 55.55 |
| 2 | 30 | Ritche | 66.66 |
| 3 | 40 | Travis | 77.77 |
| 4 | 50 | KVR | 11.11 |

===== Accesssing the Data of DataFrame =====

- 1) DataFrameobj.head(no.of rows) OR DataFrameobj.head()
 - 2) DataFrameobj.tail(no.of rows) (OR) DataFrameobj.tail()
 - 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 to get the data from data frame
 (but not column numbers used)

- 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"]
 - 7) DataFrameobj.loc[start:stop:step,"Col Name1" : Col Name-n":step]
-

Understabding iloc[] ---- here start index included and stop index excluded and Col
 Numbers must be used to get the data from data frame (but not column names
 used)

- 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

Examples:

```
df["total"] = df["english"]+df["telugu"]+df["hindi"]+df["maths"]+df["science"]+df["social"]
df["percent"] = round((df["total"]/600)*100,2)
```

Removing Column Name from Data Frame

1) dataframe.drop(columns="col name")

2) dataframe.drop(columns="col name",inplace=True)

sorting the dataframe data

1) dataframeobj.sort_values(["colname"])

2) dataframeobj.sort_values(["colname"],ascending=False)

knowing duplicates in dataframe data

1) dataframeobj.duplicated()-----gives boolean result

Removing duplicates from dataframe data

1) dataframeobj.drop_duplicates()

2) dataframeobj.drop_duplicates(inplace=True)

Data Filtering and Conditional Change / updatons

1) dataframeobj.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)
```

For adding the record in data frame object

`df.loc[index]=[List of values]`

| Names | Subjects | Ages |
|--------------|----------|------|
| Rossum | Python | 85 |
| Ritche C | | 90 |
| Stup | C++ | 75 |
| Travis Numpy | | 65 |
| Kinney | Pandas | 80 |
| Kernigan | B | 99 |

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 DataFrameGroupBy object also provides a function nth() 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:

| city | Name | Age | Experience |
|--------|------|-----|------------|
| 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.

```
import pandas as pd
recs=[[100,"Ram","Hyd"],
[200,"Rajesh","Hyd"],
[300,"Raj","Delhi"],
[400,"Rakesh","AP"],
[500,"Jani","AP"],
[600,"Sai","AP"],
```

```

[700,"Kvr","Hyd"],
[800,"Uday","Delhi"] ]
df=pd.DataFrame(recs,columns=["SID","NAME","CITY"])
print("-----")
print(df)
print("-----")
grp=df.groupby("CITY")
print("-----")
for grpname,grpinfo in grp:
    print("-----")
    print("City:",grpname)
    print("-----")
    print(grpinfo)
    print("-----")
print("-----")
print("First records in each group")
print("-----")
print(grp.first())
print("-----")
print("Last records in each group")
print("-----")
print(grp.last())
print("-----")
print("First records in each group")
print("-----")
print(grp.nth(1))
print("-----")
print("Number of records in each group")
print("-----")
print(grp.size())
print("-----")

-----
import pandas as pd
df=pd.read_csv("D:\\KVR-PYTHON-9AM\\PANDAS\\notes\\peoples.csv")
print("-----")
print(df)
print("-----")
print("-----")
grp=df.groupby("city")
print("-----")
for grpname,grpinfo in grp:
    print("-----")
    print("City:",grpname)
    print("-----")
    print(grpinfo)
    print("-----")
print("-----")
print("First records in each group")
print("-----")
print(grp.first())

```

```

print("-----")
print("Last records in each group")
print("-----")
print(grp.last())
print("-----")
print("First records in each group")
print("-----")
print(grp.nth(1))
print("-----")
print("Number of records in each group")
print("-----")
print(grp.size())
print("-----")

```

| pid | pname | city |
|-----|--------|-------|
| 100 | Ram | Hyd |
| 200 | Rajesh | Hyd |
| 300 | Raj | Delhi |
| 400 | Rakesh | AP |
| 500 | Jani | AP |
| 600 | Sai | AP |
| 700 | Kvr | Hyd |
| 800 | Uday | Delhi |
| 900 | Shekar | Delhi |
| 750 | KHUSBU | AP |

===== 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. Here return statement of function can return More Number of Values if required
=>If the function contains yield keyword then it is called generator. Here yield statement returns the value only on demand and reduces the memory space.
=>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.

```
#Program demonstrating Generators
#GenEx1.py
def kvrrange(startv,stopv):
    while(startv<stopv):
        yield startv
        startv=startv+1

#main program
r=kvrrange(10,16)
print("type of r=",type(r)) # <class 'generator'>
print("content of r=",r)
print(next(r))
print(next(r))
print(next(r))
print(next(r))
print(next(r))
print(next(r))
#print(next(r))---generates StopIteration
```

```
#Program demonstrating Generators
#GenEx2.py
def kvrrange(startv,stopv):
    while(startv<=stopv):
        yield startv
        startv=startv+1

#main program
r=kvrrange(10,16)
print("type of r=",type(r)) # <class 'generator'>
print("-----")
while(True):
    try:
        print(next(r))
    except StopIteration:
        print("-----")
        break
```

```
#Program demonstrating Generators
#GenEx3.py
def kvrrange(startv,stopv,stepv):
    while(startv<stopv):
        yield startv
        startv=startv+stepv

#main program
r=kvrrange(10,101,10)
print("type of r=",type(r)) # <class 'generator'>
print("-----")
while(True):
    try:
```

```

        print(next(r))
    except StopIteration:
        print("-----")
        break
-----
#Program demonstrating Generators
#GenEx4.py
def kvrrange(startv,stopv,stepv):
    while(startv>=stopv):
        yield startv
        startv=startv+stepv

#main program
r=kvrrange(100,10,-10)
print("type of r=",type(r)) # <class 'generator'>
print("-----")
while(True):
    try:
        print(next(r))
    except StopIteration:
        print("-----")
        break
-----
#Program demonstrating Generators
#GenEx5.py
def getcourses():
    yield "PYTHON"
    yield "Dsc"
    yield "JAVA"
    yield "C"
    yield "DS"

#main program
gen=getcourses()
print("Type of gen=",type(gen))
print("-----")
print(next(gen))
print(next(gen))
print("Using for Loop")
for val in gen:
    print(val)
-----
```

===== 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, str, 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 value 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))
```

```
#Program for demonstrating Iterators
#IterEx1.py
lst=[10,"Rossum","Python",34.56,True,2+3j]
lstitr=iter(lst) # Here iter() converts Iterable object into list_Iterator
print("type of lstitr=",type(lstitr))
print("Content of lstitr=",lstitr)
print("-----")
print(next(lstitr))
print(next(lstitr))
print(next(lstitr))
print(next(lstitr))
print(next(lstitr))
```

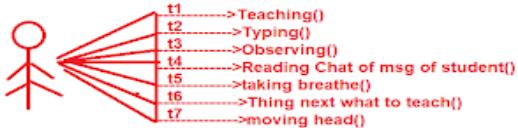
```

print(next(lstitr))
#print(next(lstitr))---generates StopIteration
-----
#Program for demonstrating Iterators
#IterEx2.py
lst=[10,"Rossum","Python",34.56,True,2+3j]
lstitr=iter(lst) # Here iter() converts Iterable object into list_Iterator
print("type of lstiter=",type(lstitr))
print("Content of lstiter=",lstitr)
print("-----")
while(True):
    try:
        print(next(lstitr))
    except StopIteration:
        print("-----")
        break

#OR--By using for Loop
tpl=(10,"Rossum","Python",34.56,True,2+3j)
tplitr=iter(tpl) # Here iter() converts Iterable object into tuple_Iterator
print("type of tpliter=",type(tplitr))
for val in tplitr:
    print(val)
#OR--By using for Loop
print("-----")
s={10,"Rossum","Python",34.56,True,2+3j}
stitr=iter(s) # Here iter() converts Iterable object into set_Iterator
print("type of stitr=",type(stitr))
for val in stitr:
    print(val)
-----
#Program for demonstrating Iterators
#IterEx3.py
d={10:"Python",20:"Java",30:"C++",40:"C"}
dictitr=iter(d) # Here iter() converts Iterable object into dict_keyiterator
print("type of dictiter=",type(dictitr))
print("Content of dictiter=",dictitr)
print("-----")
print("-----")
while(True):
    try:
        k=next(dictitr)
        print(k,d.get(k)) # OR print(k,d[k])
    except StopIteration:
        print("-----")
        break
s="PYTHON"
print("type of s=",type(s))
sitr=iter(s)
print("type of sitr=",type(sitr))

```

```
print("-----")
for val in sitr:
    print(val)
```



===== Multi Threading in Python--3 days =====

Index

=>Purpose of Multi Therading

=>Types of Applications in the Context of Threading

- a) Process Based Applications
- b) Thread Based Applications

=>Definition of Thread

=>Practical Example of Both Process and Thread Based Applications

=>Module Name Required for Developing Thread Based Applications

=>Number of approaches to develop thread based Applications

=>Programming Examples

=>Synchronization Technique in Threading (OR) Dead Locks in Threading

=>Implementation of Synchronization Technique in Threading (OR) Elimination of Dead Locks in Threading

=>Programming Examples

===== Multi Threading =====

=>The Purpose of Multi Threading is that "To Provide Concurrent Execution / Simultaneous Execution Or Parallel Processing(executing all at once)."
=>In Industry , we have two types of Applications / languages. They are

1. Process Based Applications
 2. Thread Based Applications.
-

1. Process Based Applications

=>Process Based Applications contains Single Thread
=>Process Based Applications Provides Sequential Execution
=>Process Based Applications Takes More Execution Time
=>Process Based Applications are treated as Heavy Weight Components.
Examples: C,CPP.

-2. Thread Based Applications

=>Thread Based Applications contains by default Single Thread and Programtically we can create Multiple Threads.
=>Therad Based Applications Provides Both Sequential Execution and Concurrent Execution.
=>Thread Based Applications Takes Less Execution Time
=>Therad Based Applications are treated as Light Weight Components.
=>Examples: Python, Java, C#.net...etc

Intruduction 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:

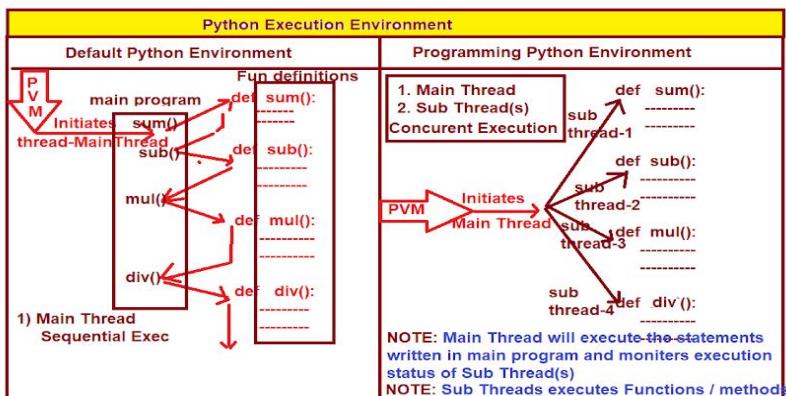
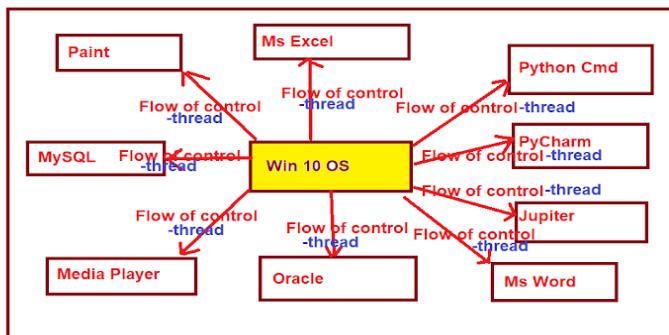
=>A Flow of Control is called thread.
=>The purpose of thread is that "To Perform certain operation whose logic developed in Functions / Methods concurrently."

=>By default Every Python contains Single Thread and whose name is "MainThread" and It provides Sequential Execution.
=>Programtically, 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 Programtically 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 ".



```
#mainthreadex1.py
import threading
print("-----")
print("Default name of thread in python Program=",threading.current_thread().name)
print("By default Numbers of threads in python Program=",threading.active_count())
print("-----")
```

```
#MainThreadwithFunctions.py
import threading
def hello():
    print("I am from hello()--thread name:",threading.current_thread().name)

def hi():
    print("I am from hi()--thread name:",threading.current_thread().name)
```

```

def greet():
    print("Good Morning--thread name:",threading.current_thread().name)
-----
#main program
print("-"*50)
print("Line-13-->Default name of thread in python
Program=",threading.current_thread().name)
print("-"*50)
hello()
print("-"*50)
hi()
print("-"*50)
greet()
print("-"*50)
print("Program execution entered by:",threading.current_thread().name)
-----
#MainThreadwithSubThreadsFunctions.py
import threading
def hello():
    print("I am from hello()--thread name:",threading.current_thread().name)

def hi():
    print("I am from hi()--thread name:",threading.current_thread().name)

def greet():
    print("Good Morning--thread name:",threading.current_thread().name)
-----
#main program
print("-"*50)
print("Line-13-->Default name of thread in python
Program=",threading.current_thread().name)
print("-"*50)
t1=threading.Thread(target=hello) # Creating sub thread by targetting hello()--Thread-1
t2=threading.Thread(target=hi) # Creating sub thread by targetting hello()--Thread-2
t3=threading.Thread(target=greet) # Creating sub thread by targetting hello()--Thread-3

#dispatch sub threads
t1.start()
t2.start()
t3.start()
print("-"*50)
-----
#No-ThreadEx1.py
import time,threading

def square(lst):
    for val in lst:

```

```

        print("Thread Name:{}--"
Square({})={}
        .format(threading.current_thread().name,val,val**2))
        time.sleep(1)

def cube(lst):
    for val in lst:
        print("Thread Name:{}--"
cube({})={}
        .format(threading.current_thread().name,val,val**3))
        time.sleep(1)
-----
#main program
bt=time.time()
print("-"*50)
print("Line-13-->Default name of thread in python
Program=",threading.current_thread().name)
print("-"*50)
lst=[2,7,12,14,19,25,3,4,18]
square(lst)
print("-"*50)
cube(lst)
print("-"*50)
et=time.time()
print("Total Execution of No-Threads Programs={}".format(et-bt))
-----
#WithSubThreadEx1.py
import time,threading

def square(lst):
    for val in lst:
        print("Thread Name:{}--"
Square({})={}
        .format(threading.current_thread().name,val,val**2))
        time.sleep(1)

def cube(lst):
    for val in lst:
        print("Thread Name:{}--"
cube({})={}
        .format(threading.current_thread().name,val,val**3))
        time.sleep(1)
-----
#main program
bt=time.time()
print("-"*50)
print("Line-13-->Default name of thread in python
Program=",threading.current_thread().name)
print("Number of Threads in this program=",threading.active_count())
print("-"*50)
lst=[2,7,12,14,19,25,3,4,18]
#create sub thread

```

```

t1=threading.Thread(target=square,args=(lst,))
t2=threading.Thread(target=cube,args=(lst,))
t1.start()
t2.start()
print("Number of Threads in this program=",threading.active_count())
t1.join()
t2.join()
print("-"*50)
et=time.time()
print("Number of Threads in this program=",threading.active_count())
print("Total Execution of Threads Programs={}".format(et-bt))

```

===== Module Name Required for developing Thread Based Applications =====

=>The module name Required for developing Thread Based Applications is "threading".

Module Name: "threading"

Functions:

1) current_thread():

This Function is used for Obtaining Name of the Thread in the context of the program

Syntax: threading.current_thread().name

2) active_count():

This Function is used for Finding Number of active Threads which are under execution.

threading.active_count()

Class1 : Thread

=>Here "threading"Contains a Pre-defined class called "Thread".

=>The purpose of Thread class is that "To create Sub / Child Threads"

=>Thread class Contains the following Constructors and Methods and given bellow

Constructor :

=>Therad(target,args):varname=threading.Thread(target=Function / Method Name, args=(val1,val2...Val-n))

=>Here varname is called Sub therad Name and This Constructor is used for Creating sub threads by specifying target Function where the Function Conatins Logic which is executed Sub Therad.

Functions

1) getName():

It is Used for obtaining the Name of Thread

Syntax: varname=threadobj.getName()

Here varname contains Name of the thread. Default Name of Sub Threads are Thread-1, Thread-2, ..., Thread-n

=> In the present Version getName() is deprecated on the name of "name" attribute

Syntax: varname=threadobj.name

2) setName():

It is Used set the Name to the subThread

Syntax: threadobj.setName(str data)

Here strdata represents Name of the Sub thread

=> In the present Version setName() is deprecated on the name of "name" attribute

Syntax: threadobj.name="strdata"

Here strdata represents Name of the Sub thread

3) start():

It is used for Dispatching the sub threads and sub threads executes the target Functions

4) join():

It is used for Joining the sub threads after Completion of Function Execution.

Syntax: subthreadobj1.join()

subthreadobj2.join()

subthreadobj-n.join()

5) is_alive() :

This Function returns True Provided Threads are under Execution

This Function returns False Provided Threads are not under Execution

Syntax: subthreadobj.is_alive()

Number of approaches for developing Thread Based Applications

=>In Python Programming, we have 2 Approaches for Developing Based Applications. They are

1. By using Functional Programming
 2. By using Object Oriented Programming
-

1. By using Functional Programming

Steps

1. import threading and other Modules if required
 2. Define a Function,which contains Operational Logic , which is executed by Sub Therad
 3. Create Sub thread(s)
 4. Dispatch the Sub threads for executing the target Function
-

2. By using Object Oriented Programming

Steps

1. import threading and other Modules if required
 2. Define a class and Define Method which contains Operational Logic , which is executed by Sub Therad
 3. Create Sub thread(s)
 4. Dispatch the Sub threads for executing the target Method
-

```
#mainthreadex1.py
import threading
print("-----")
print("Default name of thread in python Program=",threading.current_thread().name)
print("By default Numbers of threads in python Program=",threading.active_count())
print("-----")
```

```
#MainThreadwithFunctions.py
import threading
def hello():
    print("I am from hello()--thread name:",threading.current_thread().name)

def hi():
    print("I am from hi()--thread name:",threading.current_thread().name)

def greet():
    print("Good Morning--thread name:",threading.current_thread().name)
```

```
#main program
print("-"*50)
print("Line-13-->Default name of thread in python
Program=",threading.current_thread().name)
print("-"*50)
hello()
```

```

print("-"*50)
hi()
print("-"*50)
greet()
print("-"*50)
print("Program execution entered by:",threading.current_thread().name)
-----
#MainThreadwithSubThreadsFunctions.py
import threading
def hello():
    print("I am from hello()--thread name:",threading.current_thread().name)

def hi():
    print("I am from hi()--thread name:",threading.current_thread().name)

def greet():
    print("Good Morning--thread name:",threading.current_thread().name)
-----
#main program
print("-"*50)
print("Line-13-->Default name of thread in python")
Program=",threading.current_thread().name"
print("-"*50)
t1=threading.Thread(target=hello) # Creating sub thread by targetting hello()--Thread-1
t2=threading.Thread(target=hi) # Creating sub thread by targetting hello()--Thread-2
t3=threading.Thread(target=greet) # Creating sub thread by targetting hello()--Thread-3

#dispatch sub threads
t1.start()
t2.start()
t3.start()
print("-"*50)
-----
#No-ThreadEx1.py
import time,threading

def square(lst):
    for val in lst:
        print("Thread Name:{}--Square({})={}".format(threading.current_thread().name,val,val**2))
        time.sleep(1)

def cube(lst):
    for val in lst:
        print("Thread Name:{}--cube({})={}".format(threading.current_thread().name,val,val**3))
        time.sleep(1)

```

```

#main program
bt=time.time()
print("-"*50)
print("Line-13-->Default name of thread in python
Program=",threading.current_thread().name)
print("-"*50)
lst=[2,7,12,14,19,25,3,4,18]
square(lst)
print("-"*50)
cube(lst)
print("-"*50)
et=time.time()
print("Total Execution of No-Threads Programs={}".format(et-bt))
-----
#Program for generating 1 to n numbers after each and every second
#NumberGenFunEx1.py
import threading,time # Step-1
def generate(n): # Step-2
    if(n<=0):
        print("{} is invalid Input".format(n))
    else:
        for i in range(1,n+1):
            print("\t{}-->value of
i={}".format(threading.current_thread().name,i))
            time.sleep(2)

#main program
print("Default Name of the thread={}".format(threading.current_thread().name))
print("-----")
t1=threading.Thread(target=generate,args=(10,)) # creating sub thread---Step-3
print("Is t1 under execution before start={}".format(t1.is_alive()))
print("Number of active threads before start()={}".format(threading.active_count()))
t1.start() # Step-4
print("Number of active threads after start()={}".format(threading.active_count()))
print("Is t1 under execution after start()={}".format(t1.is_alive()))
t1.join()
print("-----")
print("Is t1 under execution after completion={}".format(t1.is_alive()))
print("Number of active threads after
completion={}".format(threading.active_count()))
-----
#Program for generating 1 to n numbers after each and every second
#NumberGenFunEx2.py
import threading,time # Step-1
def generate(n): # Step-2
    if(n<=0):
        print("{} is invalid Input".format(n))
    else:
        for i in range(1,n+1):

```

```

        print("\t{}-->value of
i={}".format(threading.current_thread().name,i))
        time.sleep(0.5)
-----
#main program
print("Default Name of the thread={}".format(threading.current_thread().name))
print("-----")
n=int(input("Enter How Many Numbers u want to generate:"))
t1=threading.Thread(target=generate,args=(n,)) # creating sub thread---Step-3
t1.start() # Step-4
print("Number of active threads after start()={}".format(threading.active_count()))
print("Is t1 under execution after start()={}".format(t1.is_alive()))
t1.join()
print("-----")
print("Is t1 under execution after completion={}".format(t1.is_alive()))
print("Number of active threads after
completion={}".format(threading.active_count()))
-----
#Program for generating 1 to n numbers after each and every second
#NumberGenOopsEx1.py
import threading,time # Step-1
class Numbers: # Step-2
    def generate(self,n): # Instance Method--Step-3
        if(n<=0):
            print("{} is invalid Input".format(n))
        else:
            for i in range(1,n+1):
                print("\t{}-->value of
i={}".format(threading.current_thread().name,i))
                time.sleep(0.5)
-----
#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
-----
#Program for generating 1 to n numbers after each and every second
#NumberGenOopsEx2.py
import threading,time # Step-1
class Numbers: # Step-2
    def generate(self,n): # Instance Method--Step-3
        if(n<=0):
            print("{} is invalid Input".format(n))
        else:
            for i in range(1,n+1):
                print("\t{}-->value of
i={}".format(threading.current_thread().name,i))
                time.sleep(0.5)

```

```

-----  

#main program  

t1=threading.Thread(target=Numbers().generate,args=(int(input("Enter How many  

Numbers u want to generate:")))) # Step-4  

t1.name="KVR"  

t1.start() # Step-5  

-----  

#Program for generating 1 to n numbers after each and every second  

#NumberGenOopsEx3.py  

import threading,time # Step-1  

class Numbers: # Step-2  

    def __init__(self):  

        self.n=int(input("Enter How Many Numbers u want to generate:"))  

    def generate(self): # Instance Method--Step-3  

        if(self.n<=0):  

            print("{} is invalid Input".format(self.n))  

        else:  

            for i in range(1,self.n+1):  

                print("\t{}--->value of  

i={}".format(threading.current_thread().name,i))  

                time.sleep(0.5)  

-----  

#main program  

no=Numbers() # Object creation--Pvm Calls Default constructor  

t1=threading.Thread(target=no.generate) # Step-4  

t1.name="KVR"  

t1.start() # Step-5  

-----  

#Program for generating 1 to n numbers after each and every second  

#NumberGenOopsEx4.py  

import threading,time # Step-1  

class Numbers: # Step-2  

    def __init__(self,n):  

        self.n=n  

    def generate(self): # Instance Method--Step-3  

        if(self.n<=0):  

            print("{} is invalid Input".format(self.n))  

        else:  

            for i in range(1,self.n+1):  

                print("\t{}--->value of  

i={}".format(threading.current_thread().name,i))  

                time.sleep(0.5)  

-----  

#main program  

t1=threading.Thread(target=Numbers(int(input("Enter How Many Numbers u want to  

generate:"))).generate) # Step-4  

t1.name="KVR"  

t1.start() # Step-5

```

```
#program creating a therad and set and getting the name of the threads
#SetGetThreadsEx1.py
import threading
```

```
#main program
tname=threading.current_thread()
tn=tname.name # OR tname.getName()
print("Default name of of the thread=",tn)
print("-----")
print("Default name of of the thread=",threading.current_thread().name)
```

```
#program creating a therad and set and getting the name of the threads
#SetGetThreadsEx2.py
import threading
```

```
def hello(val):
    print("Hi {}, Good Morning".format(val))
```

```
#main program
print("Default name of of the thread=",threading.current_thread().name)
#create sub threads
t1=threading.Thread(target=hello,args=("Rossum",))
print("Default Name of Sub Thread=",t1.name) # Thread-1
#set the name to the thread
t1.name="GT" # OR t1.setName("GT")
print("Default Name of Sub Thread=",t1.name) # GT
print("Is sub started execution before start():",t1.is_alive())
t1.start()
print("Is sub started execution after start():",t1.is_alive())
```

```
#WithSubThreadEx1.py
import time,threading
```

```
def square(lst):
    for val in lst:
        print("Thread Name:{}--"
Square({})={}" .format(threading.current_thread().name,val,val**2))
        time.sleep(1)
```

```
def cube(lst):
    for val in lst:
        print("Thread Name:{}--"
cube({})={}" .format(threading.current_thread().name,val,val**3))
        time.sleep(1)
```

```
#main program
bt=time.time()
print("-"*50)
```

```

print("Line-13-->Default name of thread in python
Program=",threading.current_thread().name)
print("Number of Threads in this program=",threading.active_count())
print("-"*50)
lst=[2,7,12,14,19,25,3,4,18]

```

```

#create sub thread
t1=threading.Thread(target=square,args=(lst,))
t2=threading.Thread(target=cube,args=(lst,))
t1.start()
t2.start()
print("Number of Threads in this program=",threading.active_count())
t1.join()
t2.join()
print("-"*50)
et=time.time()
print("Number of Threads in this program=",threading.active_count())
print("Total Execution of Threads Programs={}".format(et-bt))

```

```
#Program for accepting a line of text and displ after each and every second
```

```

#WordsDisplayFunEx.py
import threading,time
def displaywordschars(line):
    if(len(line)==0):
        print("No String data and nothing to display")
    else:
        print("-----")
        print("Given Line:{}".format(line))
        print("-----")
        words=line.split()
        for word in words:
            print("\t{}".format(word))
            print("\t-----")
            for ch in word:
                print("\t\t{}".format(ch))
                time.sleep(1)
            print("\t-----")
            time.sleep(1)
        print("-----")
        #lots of code

```

```
#main program
```

```

t1=threading.Thread(target=displaywordschars,args=(input("Enter Line of Text:"),))
t1.start()

```

```
#Program for generating multiple threads for displaying Even and Odd Numbers
respectively
```

```

#MultipleThreadsFunEx1.py
import threading,time

```

```

def even(n):
    if(n<=0):
        print("{} invalid input:".format(n))
    else:
        for i in range(2,n+1,2):
            print("{}-->Even
Number:{}".format(threading.current_thread().name,i))
            time.sleep(1)

def odd(n):
    if(n<=0):
        print("{} invalid input:".format(n))
    else:
        for i in range(1,n+1,2):
            print("{}-->Odd
Number:{}".format(threading.current_thread().name,i))
            time.sleep(1)

#main programs
t1=threading.Thread(target=odd,args=(int(input("Enter How many Odd Numbers u
want to generate:")),))
t2=threading.Thread(target=even,args=(int(input("Enter How many Even Numbers u
want to generate:")),))
t1.start()
t2.start()

-----
#Program for generating multiple threads for displaying Even and Odd Numbers
respectively
#MultipleThreadsOopsEx1.py
import threading,time
class EvenNumbers:
    def even(self,n):
        if(n<=0):
            print("{} invalid input:".format(n))
        else:
            for i in range(2,n+1,2):
                print("{}-->Even
Number:{}".format(threading.current_thread().name,i))
                time.sleep(1)

class OddNumbers:
    def odd(self,n):
        if(n<=0):
            print("{} invalid input:".format(n))
        else:
            for i in range(1,n+1,2):
                print("{}-->Odd
Number:{}".format(threading.current_thread().name,i))
                time.sleep(1)

#main programs

```

```
t1=threading.Thread(target=OddNumbers()).odd,args=(int(input("Enter How many Odd Numbers u want to generate:")),))
t2=threading.Thread(target=EvenNumbers()).even,args=(int(input("Enter How many Even Numbers u want to generate:")),))
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 thread objects completed their execution.

```
#Non-SynFunEx1.py
import threading,time
def mutable(n):
    if(n<0):
        print("{} is Invalid Input".format(n))
    else:
        print("{}-Mul Table for:{}".format(threading.current_thread().name,n))
```

```

for i in range(1,11):
    print("\t{} x {}={}".format(n,i,n*i))
    time.sleep(1)

#main program
t1=threading.Thread(target=multable,args=(15,))
t2=threading.Thread(target=multable,args=(19,))
t3=threading.Thread(target=multable,args=(5,))
t1.name="Rossum"
t2.name="Travis"
t3.name="Kinney"
t1.start()
t2.start()
t3.start()

-----
#Non-SynOopsEx1.py
import threading,time
class MulTable:
    def multable(self,n):
        if(n<0):
            print("{} is Invalid Input".format(n))
        else:
            print("{}-Mul Table"
for:{}".format(threading.current_thread().name,n))
            for i in range(1,11):
                print("\t{} x {}={}".format(n,i,n*i))
                time.sleep(1)

#main program
t1=threading.Thread(target=MulTable().multable,args=(15,))
t2=threading.Thread(target=MulTable().multable,args=(19,))
t3=threading.Thread(target=MulTable().multable,args=(5,))
t1.name="Rossum"
t2.name="Travis"
t3.name="Kinney"
t1.start()
t2.start()
t3.start()

#SynFunEx1.py
import threading,time
def multable(n):
    L.acquire() # Step-2
    if(n<0):
        print("{} is Invalid Input".format(n))
    else:
        print("{}-Mul Table for:{}".format(threading.current_thread().name,n))
        for i in range(1,11):
            print("\t{} x {}={}".format(n,i,n*i))

```

```

        time.sleep(1)
    L.release() # Step-3
#main program
L=threading.Lock() # Step-1
t1=threading.Thread(target=multable,args=(15,))
t2=threading.Thread(target=multable,args=(-19,))
t3=threading.Thread(target=multable,args=(5,))
t1.name="Rossum"
t2.name="Travis"
t3.name="Kinney"
t1.start()
t2.start()
t3.start()
-----
#SynOopsEx1.py
import threading,time
class MulTable:
    def multable(self,n):
        L.acquire() # Step-2
        if(n<0):
            print("{} is Invalid Input".format(n))
        else:
            print("{}-Mul Table".format(threading.current_thread().name))
        for i in range(1,11):
            print("t{} x {}={}".format(n,i,n*i))
            time.sleep(1)
        L.release() # Step-3
#main program
L=threading.Lock() # Step-1
t1=threading.Thread(target=MulTable().multable,args=(15,))
t2=threading.Thread(target=MulTable().multable,args=(-19,))
t3=threading.Thread(target=MulTable().multable,args=(5,))
t1.name="Rossum"
t2.name="Travis"
t3.name="Kinney"
t1.start()
t2.start()
t3.start()
-----
#SynOopsEx2.py
import threading,time
class MulTable:
    L=threading.Lock() # Step-1--here L is called Class Level Variable
    def multable(self,n):
        MulTable.L.acquire() # Step-2
        if(n<0):
            print("{} is Invalid Input".format(n))
        else:

```

```

        print("{}-Mul Table
for:{}".format(threading.current_thread().name,n))
        for i in range(1,11):
            print("\t{} x {}={}".format(n,i,n*i))
            time.sleep(1)
    MulTable.L.release() # Step-3

#main program
t1=threading.Thread(target=MulTable().multable,args=(15,))
t2=threading.Thread(target=MulTable().multable,args=(-19,))
t3=threading.Thread(target=MulTable().multable,args=(5,))
t1.name="Rossum"
t2.name="Travis"
t3.name="Kinney"
t1.start()
t2.start()
t3.start()

-----
#SynOopsEx2.py
import threading,time
class MulTable:
    L=threading.Lock() # Step-1--here L is called Class Level Variable
    def multable(self,n):
        self.L.acquire() # Step-2
        if(n<0):
            print("{} is Invalid Input".format(n))
        else:
            print("{}-Mul Table
for:{}".format(threading.current_thread().name,n))
            for i in range(1,11):
                print("\t{} x {}={}".format(n,i,n*i))
                time.sleep(1)
        self.L.release() # Step-3

#main program
t1=threading.Thread(target=MulTable().multable,args=(15,))
t2=threading.Thread(target=MulTable().multable,args=(-19,))
t3=threading.Thread(target=MulTable().multable,args=(5,))
t1.name="Rossum"
t2.name="Travis"
t3.name="Kinney"
t1.start()
t2.start()
t3.start()

-----
#SynOopsEx4.py
import threading,time
class MulTable:
    @classmethod
    def getLock(cls):

```

```

cls.L=threading.Lock() # Step-1--here L is called Class Level Variable
def multable(self,n):
    self.L.acquire() # Step-2
    if(n<0):
        print("{} is Invalid Input".format(n))
    else:
        print("{}-Mul Table".format(threading.current_thread().name,n))
        for i in range(1,11):
            print("\t{} x {}={}".format(n,i,n*i))
            time.sleep(1)
    self.L.release() # Step-3

#main program
MulTable.getLock()
t1=threading.Thread(target=MulTable().multable,args=(15,))
t2=threading.Thread(target=MulTable().multable,args=(-19,))
t3=threading.Thread(target=MulTable().multable,args=(5,))
t1.name="Rossum"
t2.name="Travis"
t3.name="Kinney"
t1.start()
t2.start()
t3.start()

-----
#TrainReservationFunEx1.py
import threading,time
def Reservation(nosr):
    global nos
    L.acquire()
    if(nosr>nos):
        print("Hi {} , {} Seats Are not Available".format(threading.current_thread().name,nosr))
        time.sleep(4)
        print("Now Available Seats are:{}".format(nos))
    else:
        nos=nos-nosr
        print("Hi {} , {} Seats Are Reserved".format(threading.current_thread().name,nosr))
        time.sleep(4)
        print("Now Available Seats are:{}".format(nos))
    L.release()

#main program
L=threading.Lock()
nos=10
t1=threading.Thread(target=Reservation,args=(5,))
t1.name="RamyaSri"
t2=threading.Thread(target=Reservation,args=(6,))
t2.name="Sri Laxmi"
t3=threading.Thread(target=Reservation,args=(1,))

```

```
t3.name="Pooja"
t4=threading.Thread(target=Reservation,args=(5,))
t4.name="Swathi"
t5=threading.Thread(target=Reservation,args=(2,))
t5.name="RajaSree"
t1.start()
t2.start()
t3.start()
t4.start()
t5.start()
```

```
#TrainReservationOppsEx1.py
import threading,time
class Train:
    L=threading.Lock() # Class Variable
    nos=10 # Class Varaible
    def Reservation(self,nosr):
        Train.L.acquire()
        if(nosr>Train.nos):
            print("Hi {} , {} Seats Are not Available".format(threading.current_thread().name,nosr))
            time.sleep(4)
            print("Now Available Seats are:{}".format(Train.nos))
        else:
            Train.nos=Train.nos-nosr
            print("Hi {} , {} Seats Are Reserved".format(threading.current_thread().name,nosr))
            time.sleep(4)
            print("Now Available Seats are:{}".format(Train.nos))
        Train.L.release()
```

```
#main program
t1=threading.Thread(target=Train().Reservation,args=(5,))
t1.name="RamyaSri"
t2=threading.Thread(target=Train().Reservation,args=(6,))
t2.name="Sri Laxmi"
t3=threading.Thread(target=Train().Reservation,args=(1,))
t3.name="Pooja"
t4=threading.Thread(target=Train().Reservation,args=(5,))
t4.name="Swathi"
t5=threading.Thread(target=Train().Reservation,args=(2,))
t5.name="RajaSree"
t1.start()
t2.start()
t3.start()
t4.start()
t5.start()
```

```
#TrainReservationOppsEx2.py
import threading,time
```

```

class Train(object):
    @classmethod
    def getLock(cls):
        cls.L=threading.Lock() # Class Variable
        cls.nos=10 # Class Variable
    def __init__(self,nosr):
        self.nosr=nosr
    def Reservation(self):
        Train.L.acquire()
        if(self.nosr>Train.nos):
            print("Hi {} , {} Seats Are not
Available".format(threading.current_thread().name,self.nosr))
            time.sleep(4)
            print("Now Available Seats are:{}".format(Train.nos))
        else:
            Train.nos=Train.nos-self.nosr
            print("Hi {} , {} Seats Are
Reserved".format(threading.current_thread().name,self.nosr))
            time.sleep(4)
            print("Now Available Seats are:{}".format(Train.nos))
        Train.L.release()

#main program
Train.getLock()
print("-----")
print("Total Number of Seats in Train:{}".format(Train.nos))
print("-----")
t1=threading.Thread(target=Train(5).Reservation)
t1.name="RamyaSri"
t2=threading.Thread(target=Train(6).Reservation)
t2.name="Sri Laxmi"
t3=threading.Thread(target=Train(2).Reservation)
t3.name="Pooja"
t4=threading.Thread(target=Train(5).Reservation)
t4.name="Swathi"
t5=threading.Thread(target=Train(2).Reservation)
t5.name="RajaSree"
t1.start()
t2.start()
t3.start()
t4.start()
t5.start()
-----
```

Network Programming in Python---2 days

Index

=>Purpose of Network Programming

=>Definition of Network

=>Steps for Developing Client-Server Applications

- a) Server Side Application
- b) Client Side Application

=>Module Name Required for Developing Client-Server Applications (socket)

=>Details Discussion about "socket" module

=>Programming Examples on Client-Server Applications

- a) Two Tier Applications
 - b) Three-Tier Applications
-

Introduction to Network Programming in Python

=>The purpose of Network Programming in Python is that "To share the data among the remote machines which are located across the Universe".

=>To Share the between multiple machines across the universe, we develop Two Types of Programs. They are

1. Server Side Program
2. Client Side Program

=>Developing both Server-Side and Client Side Programs are collectively called "Client-Server Applications".

1.Definition of Server-Side Program

=>A Server-Side Program is one, which will receive the Request from Client Side Process, Proccess and Gives Response back to Client Side Program

2.Definition of Client-Side Program

=>A Client-Side Program is one, which will make a request to Server Side Program and Obtains Response from Server Side Program.

Definition of DNS:

=>DNS is nothing but name of the Physical Machine where Server Side Program resides

=>The Default DNS of Every Computer is "localhost"

Definition of IP Address:

=>IP Address is is nothing but Address of the Physical Machine Server Side Program resides

=>The Default IP ADDress of Every Computer is "127.0.0.1" (loop back address)

Definition of Port Number

=>A Port Numvber of one of the Unique Numerical ID where Server Side Program is Running.

=>If Client Side Program want to get Connection from Server Side Program then Client Side Program must use (DNS/IP Address, portno)

=====

Steps for developing Client and Server Side Applications

=====

Steps for developing Server Side Applications

Step-1: Import socket Module

Step-2: Create an object of Socket and BINDS with certain DNS/IPAddress and Portno (Server Socket)

Step-3: Every Server Side Program Must CONFIGURE in such a way that to how many clients Server Side Program can Communicate.

Step-4: Every Server Side Program Must ACCEPT Request of Client Side Program(obtaing ClientSocket and Ip Address of Client Side Program)

Step-5: Every Server Side Program Must READ Request of Client Program

Step-6: Every Server Side Program Must PROCESS the Request of Client Program and gives RESPONSE to the Client Side Program

Step-7:Every Server Side Program Repeats Step-4, Step-5 and step-6 until all requests of Client Completed.

Steps for developing Clenet Side Applications

Step-1: Import socket Module

Step-2: Every Client Side Program create an object of Socket and obtains CONNECTION from Server Side Program byb passing (DNS/IPAddress, pno) (known as Client Socket object)

Step-3: Every Client Side Program send REQUEST to Server Side Program

Step-4: Every Client Side Program must RECEIVE the RESPONSE from Server Side program

Step-5: Every Client Side Program repeats Step-3 and Step-4 until all requests are Completed.

Module Name required for Developing Networking Applications

=>The Module Name required for Developing Networking Applications is "socket".

Functions in socket module

1. `socket()`: This Function is Used for Creating an object of Socket

=>Syntax: `varname=socket.socket()`

=>Here varname is an object of <class,socket.Socket>

=>An object of Socket acts as Bi-Directional Communication Entity.

=>If we create an obejct of socket at Server Side Program then It is called Sever Socket

=>If we create an obejct of socket at Client Side Program then It is called Client Socket

Example: `s=socket.socket()`

2. `bind()`: This Function is used for Making Server Side Program (Sever Socket) to be available at Certain Machine and Port Number (is called Binding)

Syntax: `serversocketobj.bind("DNS",portno)`

(OR)

Syntax: `serversocketobj.bind("IP Address",portno)`

Examples: `s.bind("localhost",8888)`

(OR)

`s.bind("127.0.0.1",8888)`

Here s is called Server Socket

3. `listen()`: This Function is used for Configuring Server Side Program in such way that To How many

clients the server side program can communicate.

Syntax: `serversockobj.listen(no. of Clients)`

Example: `s.listen(3)`

4. `accept()`: This Function is used for Obtaining Request of Client Side Program.

=>Obtaining Request of Client Side Program is nothing getting Client Socket object and

Address of Client Side Program at Server Side Program.

=>Syntax: `var1,var2=serversockobj.accept()`

here var1 represents Client Socket object

here var2 represents Address of Client Side

Program

=>Examples: `csock,caddr=s.accept()`

5. `recv()` with `decode()`: This Function is used for Receving the Request of Client at Server and It can also be use at Client Side Program for Receving the Response of Server Side Program

Syntax: `varname=clientsockobj.recv(1024 | 2048 | 4096).decode()`

Here varname is an object of type str

Here decode() converting bytes object data into str type.

Examples:- `csdata=csock.recv(1024).decode()`

6. send with encode(): This Function is used for Sending Request of Client Side Program to Server Side Program and It can also be used at Server Side program for Sending Response to Client Side Program.

Syntax: clientsockobj.send(strdata.encode())
Example: csock.send("strdata".encode())

7. connect(): This function is used for obtaining connection from Server Side Program By Client Side

Program by passing (DNS/IPAddress, Portno).

Syntax: s=socket.socket()
s.connect(("DNS/IPAddress",portno))
Here s is called Client Socket

Examples: s.connect(("localhost",8888))
(OR)
s.connect(("127.0.0.1",8888))

#ChatClient.py

```
import socket
while(True):
    s=socket.socket()
    s.connect(("127.0.0.1",8558))
    csd=input("Student-->")
    if(csd.lower() == "bye"):
        break
    else:
        s.send(csd.encode())
        ssd=s.recv(1024).decode()
        print("KVR-->{}".format(ssd))
```

#ChatServer.py

```
import socket
s=socket.socket()
s.bind(("127.0.0.1",8558))
s.listen(1)
print("SSP is Ready is accept any CSP Request")
print("-----")
while(True):
    cs,ca=s.accept()
    csd=cs.recv(1024).decode()
    print("Student-->{}".format(csd))
    ssd=input("KVR-->")
    cs.send(ssd.encode())
```

#EmpDataBaseClient.py

```
import socket
s=socket.socket()
s.connect(("localhost",7878))
empno=input("Enter Employee Number:")
```

```

s.send(empno.encode())
record=s.recv(1024).decode()
print("Result from Server")
print("-----")
print(record)
print("-----")
-----
#EmpDataBaseServer.py
import socket,cx_Oracle
s=socket.socket()
s.bind(("localhost",7878))
s.listen(1)
print("SSP is Ready to accept any CSP Request:")
print("-----")
while(True):
    try:
        cs,ca=s.accept()
        eno=int(cs.recv(1024).decode())
        #PDDB Code
        con=cx_Oracle.connect("system/manager@localhost/xe")
        cur=con.cursor()
        cur.execute("select * from employee where eno=%d" %eno)
        record=cur.fetchone()
        if(record==None):
            cs.send("{} Employee Number Does not Exist-"
Invalid".format(eno).encode())
        else:
            cs.send(str(record).encode())
    except ValueError:
        cs.send("Don't enter strs,alnums and symbols for Employee"
Number".encode())
    except cx_Oracle.DatabaseError as db:
        cs.send(("Problem in DataBase:"+db).encode())
-----
#EmpDataBaseServerDetails.py
import socket,cx_Oracle
s=socket.socket()
s.bind(("localhost",7878))
s.listen(1)
print("SSP is Ready to accept any CSP Request:")
print("-----")
while(True):
    try:
        cs,ca=s.accept()
        eno=int(cs.recv(1024).decode())
        #PDDB Code
        con=cx_Oracle.connect("system/manager@localhost/xe")
        cur=con.cursor()
        cur.execute("select * from employee where eno=%d" %eno)
        record=cur.fetchone()
    
```

```

        if(record==None):
            cs.send("{} Employee Number Does not Exist-
Invalid".format(eno).encode())
        else:
            s1="Employee Number:{}".format(record[0])
            s2="Employee Name:{}".format(record[1])
            s3="Employee Salary:{}".format(record[2])
            s4="Employee Designation:{}".format(record[3])
            res=s1+"\n"+s2+"\n"+s3+"\n"+s4+"\n"
            cs.send(res.encode())
    except ValueError:
        cs.send("Don't enter strs,alnums and symbols for Employee
Number".encode())
    except cx_Oracle.DatabaseError as db:
        cs.send(("Problem in DataBase:"+db).encode())
-----
#program for reading the records from table--fetchmany()
#OracleSelectRecordsEx3.py
import cx_Oracle
def readrecords():
    try:
        con=cx_Oracle.connect("system/manager@localhost/xe")
        cur=con.cursor()
        sq="select * from employee"
        cur.execute(sq)
        print("-"*50)
        records=cur.fetchmany(6)
        for record in records:
            print(record)
        print("-"*50)

    except cx_Oracle.DatabaseError as db:
        print("Problem in Database:",db)

#main program
readrecords()
-----
#Client Program accepts the numerical value from KBD and gets its square as a
Result from Server Side program
#SquareClient.py
import socket # Step-1
#Step-2
s=socket.socket()
s.connect(("localhost",8888)) # Step-3
print("CSP got Connection from SSP")
print("-----")
#Step-3
n=input("Enter a number ofr geting its Square:")
s.send(n.encode())
#Step-4

```

```

res=s.recv(1024).decode()
print("Square({})={}".format(n,res))

#Server Program accepts the numerical value from client side and gives square as a
Result
#SquareServer.py
import socket # Step-1
#Step-2
s=socket.socket()
s.bind(("localhost",8888))
#Step-3
s.listen(2)
print("SSP is Ready to accept any CSP Request:")
while(True):
    try:
        cs,ca=s.accept() # Step-4
        #Step-5
        csdata=cs.recv(1024).decode()
        print("Value of Client at Server={}".format(csdata))
        #Step-6
        val=int(csdata)
        res=val**2
        cs.send(str(res).encode())
    except ValueError:
        cs.send("Don't enter strs,alnums and symbols for numbers:".encode())

```

===== Regular Expressions in Python--3 days =====

Index

- =>Purpose of Regular Expressions
- =>Applications of Regular Expressions
- =>Definition of Regular Expression
- =>Module Name required for Developing Applications in Regular Expressions (re)
- =>Functions in "re" module
 - 1) search()
 - 2) findall()
 - 3) finditer()
 - 4) start()
 - 5) end()
 - 6) group()
- =>Programming Examples
- =>Programmer-Defined Charcater Classes in Regular Expressions
- =>Programming Examples
- =>Pre-Defined Charcater Classes in Regular Expressions
- =>Programming Examples
- =>Quatifiers in Regular Expressions
- =>Programming Examples

=>Combined Programming Examples in Regular Expressions

=====

Regular Expressions in Python

=====

=>Regular Expressions concept is one of the Programming Languages Independent Concept.

=>Regular Expressions Library Developed in Python on the Module name called "re".

=>When we develop Regular Expressions Applications / Program, we must import "re" module.

=>The purpose of Regular Expressions in Python is that "To Build Robust Application by Data Validations".

Applications of Regular Expressions in Python

=>Regular Expressions are used in Language Compilers and Interpreters

=>Regular Expressions are used in Operating System Development

=>Regular Expressions are used in Electronic Circutary Designing

=>Regular Expressions are used in Universal Protocols development

=>Regular Expressions are used in Pattern Matching Applications.

Definition of Regular Expressions

=>A Regular Expressions is one of the Search Pattern which is a combination of Alphabets,Digits and Special Symbols and It used to Search / Match / Find in Given Data for Obtaining Desired Result.

=====

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 findIter() and search()

=>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

#RegExp1.py

import re

gd="python is an oop Lang.Python is also Fun Prog Lang"

sp="Python"

res=re.search(sp, gd)

print(res,type(res))

"""\D:\KVR-PYTHON-9AM\REG EXPR>py RegExp1.py

<re.Match object; span=(22, 28), match='Python'> <class 're.Match'>

"""

```
#RegExp2.py
import re
gd="Python is an oop Lang.Python is also Fun Prog Lang"
sp="KVR"
res=re.search(sp,gd)
print(res,type(res))

"""
D:\KVR-PYTHON-9AM\REG EXPR>py RegExp2.py
None <class 'NoneType'>
"""


```

```
#RegExp3.py
import re
gd="Python is an oop Lang.Python is also Fun Prog Lang"
sp="Python"
res=re.search(sp,gd)
if(res!=None):
    print("Search is Sucessful")
    print("Start Index:{}".format(res.start()))
    print("End Index:{}".format(res.end()))
    print("Matched Value:{}".format(res.group()))
else:
    print("Search is Not Sucessful")

"""
Search is Sucessful
Start Index:0
End Index:6
Matched Value:Python
"""


```

```
#RegExp4.py
import re
gd="Python is an oop Lang.Python is also Fun Prog Lang"
sp="Python"
noa=re.findall(sp,gd)
print(noa,type(noa))

"""
['Python', 'Python'] <class 'list'>
"""


```

```
#RegExp5.py
import re
gd="Python is an oop Lang.Python is also Fun Prog Lang"
sp="Python"
noa=re.findall(sp,gd)
if(len(noa)):
```

```

        for val in noa:
            print("\t{}".format(val))
        print("{} Matched {} Time(s)".format(sp,len(noa)))
else:
    print("{} not Mached".format(sp))

"""

```

```

D:\KVR-PYTHON-9AM\REG EXPR>py RegExp5.py
    Python
    Python
Python Matched 2 Time(s)
"""

-----
```

```

#RegExp6.py
import re
gd="Python is an oop Lang.Python is also Fun Prog Lang"
sp="Python"
matres=re.finditer(sp,gd) # Here matres is an object of <class 'callable_iterator'>
print("-"*50)
noc=0
for mat in matres:
    print("Start Index:{} End Index:{}"
Value:{}".format(mat.start(),mat.end(),mat.group()))
    noc=noc+1
else:
    print("-"*50)
    print("Number of occurences of '{}' ={}".format(sp,noc))
    print("-"*50)

"""

```

```

D:\KVR-PYTHON-9AM\REG EXPR>py RegExp6.py
-----
```

```

Start Index:0 End Index:6 Value:Python
Start Index:22 End Index:28 Value:Python
-----
```

```

Number of occurences of 'Python' =2
-----
```

```

#RegExp7.py
import re
while(True)
gd=input("Enter a Line of Text:")
sp=input("Enter which word u want search:")
matres=re.finditer(sp,gd)
print("-"*50)
noc=0
for mat in matres:
    print("Start Index:{} End Index:{}"
Value:{}".format(mat.start(),mat.end(),mat.group()))
    noc=noc+1
else:
```

```

print("-"*50)
print("{} Found {} Time(s)".format(sp,noc))
print("-"*50)
-----
#RegExp8.py
import re
lst=[]
print("Enter a Line of Text and press @ to stop:")
while(True):
    gd=input()
    if(gd=="@"):
        break
    else:
        lst.append(gd)

#searching process starts
print("-----")
gd=str(lst)
print("-----")
sp=input("Enter which word u want search:")
matres=re.finditer(sp,gd)
print("-"*50)
noc=0
for mat in matres:
    print("Start Index:{} End Index:{}\nValue:{}".format(mat.start(),mat.end(),mat.group()))
    noc=noc+1
else:
    print("-"*50)
    print("{} Found {} Time(s)".format(sp,noc))
    print("-"*50)
-----

```

Programmer-Defined Charcater Classes in Regular Expressions

=>Programmer-Defined Charcater Classes in Regular Expressions are Defined by Programmers for Preparation of Search Pattern and It is used to search / match / find in Given Data and obtains Desired Result.

=>Syntax: " [Programmer-Defined Character Class Name] "

=>In Python Programming, we can define the following Programmer-Defined Character Classes.

1. [abc] ----->Seaches for either 'a' or 'b' or 'c' only
2. [^abc]----->Searches for all except 'a' or 'b' or 'c'.
3. [a-z]----->Seaches 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 Case)
 - 10.[^A-Za-z]----->Searches for all except Alphabets (Lower and upper Case)
 - 11.[A-Za-z0-9]--->Searches for all Alpha-numeric Values (OR) Word Character only
 12. [^A-Za-z0-9]-->Searches for all Special Symbols
 - 13.[A-Z0-9]----->Searches for Upper Case Alphabets and Digits only
 - 14.[^A-Z0-9]----->Searches for all except Upper Case Alphabets and Digits only
 - 15.[a-z0-9]----->Searches for Lower Case Alphabets and Digits only
 - 16.[^a-z0-9]----->Searches for all except Lower Case Alphabets and Digits
-

#Program for Searching all except a or b or c

```
#RegExp10.py
import re
print("-"*50)
matres=re.finditer("[^abc]", "DaR6%KcLpwMG@5bQ8")
for mat in matres:
    print("Start           Index:{}           End           Index:{}"
          .format(mat.start(),mat.end(),mat.group()))
print("-"*50)

"""
```

D:\KVR-PYTHON-9AM\REG EXPR>py RegExp10.py

```
Start Index:0 End Index:1 Value:D
Start Index:2 End Index:3 Value:R
Start Index:3 End Index:4 Value:6
Start Index:4 End Index:5 Value:%
Start Index:5 End Index:6 Value:K
Start Index:7 End Index:8 Value:L
Start Index:8 End Index:9 Value:p
Start Index:9 End Index:10 Value:w
Start Index:10 End Index:11 Value:M
Start Index:11 End Index:12 Value:G
Start Index:12 End Index:13 Value:@
Start Index:13 End Index:14 Value:5
Start Index:15 End Index:16 Value:Q
Start Index:16 End Index:17 Value:8
"""
```

#Program for Searching all lower case alphabets

```
#RegExp11.py
import re
print("-"*50)
matres=re.finditer("[a-z]", "DaR6%KcLpwMG @5bQ8")
for mat in matres:
    print("Start Index:{} End Index:{}"
          .format(mat.start(),mat.end(),mat.group()))
print("-"*50)

"""
```

D:\KVR-PYTHON-9AM\REG EXPR>py RegExp11.py

```
Start Index:1 End Index:2 Value:a
Start Index:6 End Index:7 Value:c
Start Index:8 End Index:9 Value:p
Start Index:9 End Index:10 Value:w
Start Index:14 End Index:15 Value:b
```

```
#Program for Searching all except lower case alphabets
#RegExp12.py
import re
print("-"*50)
matres=re.finditer("[^a-z]", "DaR6%KcLpwMG @5bQ8")
for mat in matres:
    print("Start Index:{} End Index:{}\nValue:{}".format(mat.start(),mat.end(),mat.group()))
print("-"*50)
```

D:\KVR-PYTHON-9AM\REG EXPR>py RegExp12.py

```
Start Index:0 End Index:1 Value:D
Start Index:2 End Index:3 Value:R
Start Index:3 End Index:4 Value:6
Start Index:4 End Index:5 Value:%
Start Index:5 End Index:6 Value:K
Start Index:7 End Index:8 Value:L
Start Index:10 End Index:11 Value:M
Start Index:11 End Index:12 Value:G
Start Index:12 End Index:13 Value:@
Start Index:13 End Index:14 Value:5
Start Index:15 End Index:16 Value:Q
Start Index:16 End Index:17 Value:8
```

```
#Program for Searching all Upper case alphabets
#RegExp13.py
import re
print("-"*50)
matres=re.finditer("[A-Z]", "DaR6%KcLpwMG @5bQ8")
for mat in matres:
    print("Start Index:{} End Index:{}\nValue:{}".format(mat.start(),mat.end(),mat.group()))
print("-"*50)
```

D:\KVR-PYTHON-9AM\REG EXPR>py RegExp13.py

```

-----
Start Index:0 End Index:1 Value:D
Start Index:2 End Index:3 Value:R
Start Index:5 End Index:6 Value:K
Start Index:7 End Index:8 Value:L
Start Index:10 End Index:11 Value:M
Start Index:11 End Index:12 Value:G
Start Index:15 End Index:16 Value:Q
-----
"""

-----
#Program for Searching all except Upper case alphabets
#RegExp14.py
import re
print("-"*50)
matres=re.finditer("[^A-Z]","DaR6%KcLpwMG @5bQ8")
for mat in matres:
    print("Start Index:{} End Index:{}\nValue:{}".format(mat.start(),mat.end(),mat.group()))
print("-"*50)

```

"""

D:\KVR-PYTHON-9AM\REG EXPR>py RegExp14.py

```

-----
Start Index:1 End Index:2 Value:a
Start Index:3 End Index:4 Value:6
Start Index:4 End Index:5 Value:%
Start Index:6 End Index:7 Value:c
Start Index:8 End Index:9 Value:p

```

```

Start Index:9 End Index:10 Value:w
Start Index:12 End Index:13 Value:@
Start Index:13 End Index:14 Value:5
Start Index:14 End Index:15 Value:b
Start Index:16 End Index:17 Value:8

```

```

-----
#Program for Searching all Digits
#RegExp15.py
import re
print("-"*50)
matres=re.finditer("[0-9]","DaR6%KcLpwMG @5bQ8")
for mat in matres:
    print("Start Index:{} End Index:{}\nValue:{}".format(mat.start(),mat.end(),mat.group()))
print("-"*50)

```

"""

D:\KVR-PYTHON-9AM\REG EXPR>py RegExp15.py

```
Start Index:3 End Index:4 Value:6
Start Index:13 End Index:14 Value:5
Start Index:16 End Index:17 Value:8
```

"""

```
#Program for Searching all except Digits
#RegExp16.py
import re
print("-"*50)
matres=re.finditer("[^0-9]","DaR6%KcLpwMG@5bQ8")
for mat in matres:
    print("Start Index:{} End Index:{}"
Value:{}".format(mat.start(),mat.end(),mat.group())))
print("-"*50)
```

"""

D:\KVR-PYTHON-9AM\REG EXPR>py RegExp16.py

```
Start Index:0 End Index:1 Value:D
Start Index:1 End Index:2 Value:a
Start Index:2 End Index:3 Value:R
Start Index:4 End Index:5 Value:%
Start Index:5 End Index:6 Value:K
Start Index:6 End Index:7 Value:c
Start Index:7 End Index:8 Value:L
Start Index:8 End Index:9 Value:p
Start Index:9 End Index:10 Value:w
Start Index:10 End Index:11 Value:M
Start Index:11 End Index:12 Value:G
Start Index:12 End Index:13 Value:@
Start Index:14 End Index:15 Value:b
Start Index:15 End Index:16 Value:Q
```

"""

```
#Program for Searching all Alphabets
#RegExp17.py
import re
print("-"*50)
matres=re.finditer("[A-Za-z]","DaR6%KcLpwMG@5bQ8")
for mat in matres:
    print("Start Index:{} End Index:{}"
Value:{}".format(mat.start(),mat.end(),mat.group())))
print("-"*50)
```

"""

D:\KVR-PYTHON-9AM\REG EXPR>py RegExp17.py

```
-----
Start Index:0 End Index:1 Value:D
Start Index:1 End Index:2 Value:a
Start Index:2 End Index:3 Value:R
Start Index:5 End Index:6 Value:K
Start Index:6 End Index:7 Value:c
Start Index:7 End Index:8 Value:L
Start Index:8 End Index:9 Value:p
Start Index:9 End Index:10 Value:w
Start Index:10 End Index:11 Value:M
Start Index:11 End Index:12 Value:G
Start Index:14 End Index:15 Value:b
Start Index:15 End Index:16 Value:Q
-----
```

"""

```
-----  
#Program for Searching all except Alphabets  
#RegExp18.py  
import re  
print("-"*50)  
matres=re.finditer("[^A-Za-z]","DaR6%KcLpwMG@5bQ8")  
for mat in matres:  
    print("Start Index:{} End Index:{}  
Value:{}".format(mat.start(),mat.end(),mat.group()))  
print("-"*50)
```

"""

```
-----  
D:\KVR-PYTHON-9AM\REG EXPR>py RegExp18.py
```

```
-----  
Start Index:3 End Index:4 Value:6  
Start Index:4 End Index:5 Value:%  
Start Index:12 End Index:13 Value:@  
Start Index:13 End Index:14 Value:5  
Start Index:16 End Index:17 Value:8
```

"""

```
-----  
#Program for Searching all Alpha-numerics  
#RegExp19.py  
import re  
print("-"*50)  
matres=re.finditer("[A-Za-z0-9]","DaR6%KcLpwMG@5bQ8")  
for mat in matres:  
    print("Start Index:{} End Index:{}  
Value:{}".format(mat.start(),mat.end(),mat.group()))  
print("-"*50)
```

"""

```
D:\KVR-PYTHON-9AM\REG EXPR>py RegExp19.py
```

```

Start Index:0 End Index:1 Value:D
Start Index:1 End Index:2 Value:a
Start Index:2 End Index:3 Value:R
Start Index:3 End Index:4 Value:6
Start Index:5 End Index:6 Value:K
Start Index:6 End Index:7 Value:c
Start Index:7 End Index:8 Value:L
Start Index:8 End Index:9 Value:p
Start Index:9 End Index:10 Value:w
Start Index:10 End Index:11 Value:M
Start Index:11 End Index:12 Value:G
Start Index:13 End Index:14 Value:5
Start Index:14 End Index:15 Value:b
Start Index:15 End Index:16 Value:Q
Start Index:16 End Index:17 Value:8

```

```

#Program for Searching all Special Symbols
#RegExp20.py
import re
print("-"*50)
matres=re.finditer("[^A-Za-z0-9]", "DaR6%KcL!pwM&G@5bQ8")
for mat in matres:
    print("Start Index:{} End Index:{}\nValue:{}".format(mat.start(),mat.end(),mat.group()))
print("-"*50)

```

D:\KVR-PYTHON-9AM\REG EXPR>py RegExp20.py

```

Start Index:4 End Index:5 Value:%
Start Index:8 End Index:9 Value:!
Start Index:12 End Index:13 Value:&
Start Index:14 End Index:15 Value:@

```

```

#Program for Searching either a or b or c only
#RegExp9.py
import re
print("-"*50)
matres=re.finditer("[abc]", "DaR6%KcLpwMG @5bQ8")
for mat in matres:
    print("Start Index:{} End Index:{}\nValue:{}".format(mat.start(),mat.end(),mat.group()))
print("-"*50)

```

"""

D:\KVR-PYTHON-9AM\REG EXPR>py RegExp9.py

Start Index:1 End Index:2 Value:a
 Start Index:6 End Index:7 Value:c
 Start Index:14 End Index:15 Value:b

"""

Pre-Defined Charcater Classes in Regular Expressions

=>Pre-Defined Charcater Classes in Regular Expressions are already Defined by Python Lang Developers and Used By Python Programmers for Preparation of Search Pattern and It is used to search / match / find in Given Data and obtains Desired Result.

=>Syntax: "\Pre-Defined Character Class"

=>In Python Programming, we have the following essential Pre-Defined Character Classes.

1. \s----->Searches for Space Character only
 2. \S----->Searches for all except Space Character
 3. \w----->Searches for a word Character OR [A-Za-z0-9]
 4. \W----->Searches for Special Symbols OR [^A-Za-z0-9]
 5. \d----->Searches for Digit OR [0-9]
 6. \D----->Searches for all Except Digits OR [^0-9]
-

#Program for Searching Space Character

```
#RegExp21.py
import re
print("-"*50)
matres=re.finditer("\s","D aR 6%KcL!pwM&G@5bQ8")
for mat in matres:
    print("Start Index:{} End Index:{}"
Value:{}".format(mat.start(),mat.end(),mat.group()))
print("-"*50)
```

"""

D:\KVR-PYTHON-9AM\REG EXPR>py RegExp21.py

Start Index:1 End Index:2 Value:
 Start Index:4 End Index:5 Value:

"""

#Program for Searching all except Space Character

#RegExp22.py

import re

```

print("-"*50)
matres=re.finditer("\S","D aR 6%KcL!pwM&G@5bQ8")
for mat in matres:
    print("Start Index:{} End Index:{}\nValue:{}".format(mat.start(),mat.end(),mat.group()))
print("-"*50)

"""
-----
```

D:\KVR-PYTHON-9AM\REG EXPR>py RegExp22.py

```

Start Index:0 End Index:1 Value:D
Start Index:2 End Index:3 Value:a
Start Index:3 End Index:4 Value:R
Start Index:5 End Index:6 Value:6
Start Index:6 End Index:7 Value:%
Start Index:7 End Index:8 Value:K
Start Index:8 End Index:9 Value:c
Start Index:9 End Index:10 Value:L
Start Index:10 End Index:11 Value:!
Start Index:11 End Index:12 Value:p
Start Index:12 End Index:13 Value:w
Start Index:13 End Index:14 Value:M
Start Index:14 End Index:15 Value:&
Start Index:15 End Index:16 Value:G
Start Index:16 End Index:17 Value:@
Start Index:17 End Index:18 Value:5
Start Index:18 End Index:19 Value:b
Start Index:19 End Index:20 Value:Q
Start Index:20 End Index:21 Value:8
```

"""

```

#Program for Searching word character
#RegExp23.py
import re
print("-"*50)
matres=re.finditer("\w","D aR 6%KcL!pwM&G@5bQ8")
for mat in matres:
    print("Start Index:{} End Index:{}\nValue:{}".format(mat.start(),mat.end(),mat.group()))
print("-"*50)
```

"""

D:\KVR-PYTHON-9AM\REG EXPR>py RegExp23.py

```

Start Index:0 End Index:1 Value:D
Start Index:2 End Index:3 Value:a
Start Index:3 End Index:4 Value:R
```

```

Start Index:5 End Index:6 Value:6
Start Index:7 End Index:8 Value:K
Start Index:8 End Index:9 Value:c
Start Index:9 End Index:10 Value:L
Start Index:11 End Index:12 Value:p
Start Index:12 End Index:13 Value:w
Start Index:13 End Index:14 Value:M
Start Index:15 End Index:16 Value:G
Start Index:17 End Index:18 Value:5
Start Index:18 End Index:19 Value:b
Start Index:19 End Index:20 Value:Q
Start Index:20 End Index:21 Value:8
-----
```

"""

```

#Program for Searching all special symbols only
#RegExp24.py
import re
print("-"*50)
matres=re.finditer("\W","D aR 6%KcL!pwM&G@5bQ8")
for mat in matres:
    print("Start Index:{} End Index:{}\nValue:{}".format(mat.start(),mat.end(),mat.group()))
print("-"*50)
```

"""

D:\KVR-PYTHON-9AM\REG EXPR>py RegExp24.py

```

-----
```

```

Start Index:1 End Index:2 Value:
Start Index:4 End Index:5 Value:
Start Index:6 End Index:7 Value:%
Start Index:10 End Index:11 Value:!
Start Index:14 End Index:15 Value:&
Start Index:16 End Index:17 Value:@
-----
```

"""

```

#Program for Searching all special symbols only
#RegExp25.py
import re
print("-"*50)
matres=re.finditer("\d","D aR 6%KcL!pwM&G@5bQ8")
for mat in matres:
    print("Start Index:{} End Index:{}\nValue:{}".format(mat.start(),mat.end(),mat.group()))
print("-"*50)
```

"""

D:\KVR-PYTHON-9AM\REG EXPR>py RegExp25.py

```
Start Index:5 End Index:6 Value:6
Start Index:17 End Index:18 Value:5
Start Index:20 End Index:21 Value:8
```

"""

```
#Program for Searching all except Digits
#RegExp26.py
import re
print("-"*50)
matres=re.finditer("\D","D aR 6%KcL!pwM&G@5bQ8")
for mat in matres:
    print("Start Index:{} End Index:{}\nValue:{}".format(mat.start(),mat.end(),mat.group()))
print("-"*50)
```

"""

```
D:\KVR-PYTHON-9AM\REG EXPR>py RegExp26.py
```

```
Start Index:0 End Index:1 Value:D
Start Index:1 End Index:2 Value:
Start Index:2 End Index:3 Value:a
Start Index:3 End Index:4 Value:R
Start Index:4 End Index:5 Value:
Start Index:6 End Index:7 Value:%
Start Index:7 End Index:8 Value:K
Start Index:8 End Index:9 Value:c
Start Index:9 End Index:10 Value:L
Start Index:10 End Index:11 Value:!
Start Index:11 End Index:12 Value:p
Start Index:12 End Index:13 Value:w
Start Index:13 End Index:14 Value:M
Start Index:14 End Index:15 Value:&
Start Index:15 End Index:16 Value:G
Start Index:16 End Index:17 Value:@
Start Index:18 End Index:19 Value:b
Start Index:19 End Index:20 Value:Q
```

"""

Quantifiers in Regular Expressions

=>Quantifiers in Regular Expressions makes us understand the number of occurrences to be searched in given data for obtaining desired result.

=>Quantifiers in Regular Expressions to be used in Search Pattern for finding number of occurrence in Given data for obtaining desired result.

=>The essential Quantifiers in Regular Expressions are given below.

1. k ----->Exactly Searching for One k
 2. k+ ----->Searching either One k or More k
 3. k* ----->Searching for either Zero k or One k or More k's
 4. k?-----> Searching for Either Zero k or One k
 5. . ----->Searching for all of Individual occurrences of each value
-

Most Imp Points

- 1) \d+ or [0-9]+ Searches for One Digit or More Number of Digits
 - 2) \dd OR [0-9][0-9]--Searches for Two Digit Numbers Only
 - 3) In General \d{n} OR [0-9]{n}--Searches for n-Digit Number
 - 4) In General \d{n,m} OR [0-9]{n,m}--Searches for Min n-Digit Number and max m-Digit numbers
 - 5) \w+ OR [A-Z-a-z0-9]+---Searches for either One Word Character or More Word Charater.
 - 6) \w{m} OR [A-Z-a-z0-9]{m}--->Searches for a word whose length is m only
 - 7) In general \w{m,n} OR [A-Z-a-z0-9]{m:n}---Searches for Min m-word length and max n-word length.
 - 8) \dd.\ddd searches a floating point value which contains an Integer part whose length is 2 and Number of Decimal Places are 3.
-

#Program for Exactly One K

```
#RegExp27.py
import re
print("-"*50)
matres=re.finditer("K","KVKKVKKKVKV")
for mat in matres:
    print("Start Index:{} End Index:{}\nValue:{}".format(mat.start(),mat.end(),mat.group()))
print("-"*50)
```

#Program for One OR More K's

```
#RegExp28.py
import re
print("-"*50)
matres=re.finditer("K+","KVKKVKKKVKV")
for mat in matres:
```

```

        print("Start Index:{} End Index:{}"
Value:{}".format(mat.start(),mat.end(),mat.group()))
print("-"*50)
-----
```

```

#Program for zero k or One K OR More K's
#RegExp29.py
import re
print("-"*50)
matres=re.finditer("K*","KVKKVKKKV")
for mat in matres:
    print("Start Index:{} End Index:{}"
Value:{}".format(mat.start(),mat.end(),mat.group()))
print("-"*50)
```

"""

D:\KVR-PYTHON-9AM\REG EXPR>py RegExp29.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 zero k or One K
#RegExp30.py
import re
print("-"*50)
matres=re.finditer("K?","KVKKVKKKV")
for mat in matres:
    print("Start Index:{} End Index:{}"
Value:{}".format(mat.start(),mat.end(),mat.group()))
print("-"*50)
```

"""

D:\KVR-PYTHON-9AM\REG EXPR>py RegExp30.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 all
#RegExp31.py
import re
print("-"*50)
matres=re.finditer(".","KVKKVKKKVKV")
for mat in matres:
    print("Start Index:{} End Index:{}\nValue:{}".format(mat.start(),mat.end(),mat.group()))
print("-"*50)
```

"""

D:\KVR-PYTHON-9AM\REG EXPR>py RegExp31.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
```

"""

```

#Program for Extracting Mails of the students from given text
#ExtractMailsEx1.py
import re
text="Rossum is developer of python and mail id is rossum_123@psf.com , Gosling
is the developer of java and mail id james_gosling1@sun.co.in , Travis is the
developer of numpy and whose mail id is travis_n@numpy.org and Kinney is the
developer of pandas and mail id is wes_kinney@pandas.org and Kvr mail id is
kvr1.java@gmail.com"
mailslist=re.findall("\S+@\S+",text)
print("Mails of People")
for mails in mailslist:
    print("\t{}".format(mails))
```

```

#Program for Extracting Mails of the students from given file
#ExtractMailsEx2.py
```

```

import re
with open("D:\\KVR-PYTHON-9AM\\REG EXPR\\NOTES\\mails.data","r") as fp:
    filedata=fp.read()
    mailslist=re.findall("\S+@\S+",filedata)
    print("Mails of People")
    for mails in mailslist:
        print("\t{}".format(mails))

-----
#Program for Extracting Marks of the students from given text
#ExtractMarksEx1.py
import re
text="Rossum is developer of python and whose marks 77 , Gosling is the developer
of java and whose marks 60 , Travis is the developer of numpy and whose marks 58
and Kinney is the developer of pandas and whose marks 47 and Ritche is the
developer of c and and whose marks 59"
Markslist=re.findall("\d{2}" ,text)
print("-----")
print("Marks of Students")
print("-----")
for marks in Markslist:
    print(marks)
print("-----")

-----
#Program for Extracting Names of the students from given text
#ExtractNamesEx1.py
import re
text="Rossum is developer of python , Gosling is the developer of java, Travis is the
developer of numpy and Kinney is the developer of pandas"
nameslist=re.findall("[A-Z][a-z]+",text)
print("-----")
print("Names of Students")
print("-----")
for names in nameslist:
    print(names)
print("-----")

-----
#Program for Extracting Names and Marks of the students from given text
#ExtractNamesMarksEx1.py
import re
text="Rossum is developer of python and whose marks 77 , Gosling is the
developer of java and whose marks 60 , Travis is the developer of numpy and whose
marks 58 and Kinney is the developer of pandas and whose marks 47 and Ritche is
the developer of c and and whose marks 59"
NamesList=re.findall("[A-Z][a-z]+",text)
Markslist=re.findall("\d{2}" ,text)
print("-----")
print("\tNames\tMarks")
print("-----")
for names,marks in zip(NamesList,Markslist):
    print("\t{}\t{}".format(names,marks))
print("-----")

```

```
#Program for Extracting Names and Marks of the students from given file--
student.data
#ExtractNamesMarksEx2.py
import re
try:
    with open("D:\\KVR-PYTHON-9AM\\REG EXPR\\NOTES\\student.data","r") as
fp:
    filedata=fp.read()
    NamesList=re.findall("[A-Z][a-z]+",filedata)
    MarksList=re.findall("\d{2}",filedata)
    print("-----")
    print("\tNames\tMarks")
    print("-----")
    for names,marks in zip(NamesList,MarksList):
        print("\t{}\t{}".format(names,marks))
    print("-----")

except FileNotFoundError:
    print("File Does not Exist")
```

```
#Program for Mobile Number Verification
#MobileNumberValidationEx1.py
import re
while(True):
    mno=input("Enter Ur Mobile Number:")
    if(len(mno)==10):
        res=re.search("\d{10}",mno)
        if(res!=None):
            print("{} is Valid Mobile Number".format(mno))
            break
        else:
            print("\t{} Invalid Mobile Number bcoz it contains other than
digits".format(mno))
    else:
        print("Invalid Mobile Number and Try again")
```

Rossum is developer of python and whose marks 77 , Gosling is the developer of java and whose marks 60 , Travis is the developer of numpy and whose marks 58 and Kinney is the developer of pandas and whose marks 47 and Ritche is the developer of c and whose marks 59

===== 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 Dict)

=>`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 the data from JSON file which contains a JSON Data.
Syntax:

```
json.load(file_object)
```

Examples:

#Program for JSON File Data into Dict Object

#JsonFiletoDict.py---reading the data from JSON File to Dictobj

import json

try:

```
    with open("emp.json","r" ) as fp:
        dictobj=json.load(fp)
        print(dictobj,type(dictobj))
        print("-----")
        for k,v in dictobj.items():
            print("\t{}-->{}".format(k,v))
        print("-----")
```

except FileNotFoundError:

```
    print("Json File does not exist")
```

Python--- write to JSON file

=>`json.dump()` method can be used to write dict object data to a JSON file.

Syntax:

```
json.dump(dict object, file_pointer)
```

Examples:

```
#Program for Dict data into JSON File
#DicttoJsonFile.py---Writing Dict data to JSON File
import json
dictobj={"ENO":100,"ENAME":"TRAVIS","SAL":56,"DSG":"AUTHOR"}
with open("emp.json","w") as fp:
    json.dump(dictobj,fp) # Here dump() is saving dictobj data into the json file
    print("Dict Data Saved in JSON FILE Format--verify")
```

```
#Program for Dict data into JSON File
#DicttoJsonFile.py---Writing Dict data to JSON File
import json
dictobj={"ENO":100,"ENAME":"TRAVIS","SAL":56,"DSG":"AUTHOR"}
with open("emp.json","w") as fp:
    json.dump(dictobj,fp) # Here dump() is saving dictobj data into the json file
    print("Dict Data Saved in JSON FILE Format--verify")
```

```
#Program for JSON File Data into Dict Object
#JsonFiletoDict.py---reading the data from JSON File to Dictobj
import json
try:
    with open("emp.json","r") as fp:
        dictobj=json.load(fp)
        print(dictobj,type(dictobj))
        print("-----")
        for k,v in dictobj.items():
            print("\t{}-->{}".format(k,v))
        print("-----")
except FileNotFoundError:
    print("Json File does not exist")
```

```
#Program for converting JSON String Data into Dict data
#JsontoDict.py
import json
js='{"SNO":100,"SNAME":"ROSSUM","SAL":4.5 }' # JSON String Format
print(js,type(js))
print("-----")
#Convert JSON Str Format into Dict Type
dobj=json.loads(js) # loads() is used for converting JSON String format into dict obj
type
print(dobj,type(dobj))
print("-----")
for k,v in dobj.items():
    print("\t{}\t{}".format(k,v))
print("-----")
```

```
import qrcode
# generating a QR code using the make() function
qr_img = qrcode.make("Welcome to KVR Python Classes.")
```

```
# saving the image file
qr_img.save("kvr-img.jpg")
```

All students are requested to solve the following Questions

1. Write a Python program to sum all the items in a list.
2. Write a Python program to multiply all the items in a list.
3. Write a Python program to get the largest number from a list.
4. Write a Python program to get the smallest number from a list.
5. Write a Python program to count the number of strings where the string length is 2 or more and the first and last character are same from a given list of strings.

Sample List : ['abc', 'xyz', 'aba', '1221']

Expected Result : 2

6. Write a Python program to get a list, sorted in increasing order by the last element in each tuple from a given list of non-empty tuples.

Sample List : [(2, 5), (1, 2), (4, 4), (2, 3), (2, 1)]

Expected Result : [(2, 1), (1, 2), (2, 3), (4, 4), (2, 5)]

7. Write a Python program to remove duplicates from a list.
8. Write a Python program to check a list is empty or not.
9. Write a Python program to clone or copy a list.
10. Write a Python program to find the list of words that are longer than n from a given list of words.
11. Write a Python function that takes two lists and returns True if they have at least one common member.
12. Write a Python program to print a specified list after removing the 0th, 4th and 5th elements.

Sample List : ['Red', 'Green', 'White', 'Black', 'Pink', 'Yellow']

Expected Output : ['Green', 'White', 'Black']

13. Write a Python program to generate a 3*4*6 3D array whose each element is *.
14. Write a Python program to print the numbers of a specified list after removing even numbers from it.
15. Write a Python program to shuffle and print a specified list.
16. Write a Python program to generate and print a list of first and last 5 elements where the values are square of numbers between 1 and 30 (both included).
17. Write a Python program to generate and print a list except for the first 5 elements, where the values are square of numbers between 1 and 30 (both included).
18. Write a Python program to generate all permutations of a list in Python.
19. Write a Python program to get the difference between the two lists.
20. Write a Python program access the index of a list.
21. Write a Python program to convert a list of characters into a string.
22. Write a Python program to find the index of an item in a specified list.
23. Write a Python program to flatten a shallow list.
24. Write a Python program to append a list to the second list.
25. Write a Python program to select an item randomly from a list.
26. Write a python program to check whether two lists are circularly identical.
27. Write a Python program to find the second smallest number in a list.

28. Write a Python program to find the second largest number in a list.
29. Write a Python program to get unique values from a list.
30. Write a Python program to get the frequency of the elements in a list.
31. Write a Python program to count the number of elements in a list within a specified range.
32. Write a Python program to check whether a list contains a sublist.
33. Write a Python program to generate all sublists of a list.
35. Write a Python program to create a list by concatenating a given list which range goes from 1 to n.

Sample list : ['p', 'q']

n =5

Sample Output : ['p1', 'q1', 'p2', 'q2', 'p3', 'q3', 'p4', 'q4', 'p5', 'q5']

36. Write a Python program to get variable unique identification number or string.
37. Write a Python program to find common items from two lists.
38. Write a Python program to change the position of every n-th value with the (n+1)th in a list.

Sample list: [0,1,2,3,4,5]

Expected Output: [1, 0, 3, 2, 5, 4]

39. Write a Python program to convert a list of multiple integers into a single integer.

Sample list: [11, 33, 50]

Expected Output: 113350

40. Write a Python program to split a list based on first character of word.
41. Write a Python program to create multiple lists.
42. Write a Python program to find missing and additional values in two lists.

Sample data : Missing values in second list: b,a,c

Additional values in second list: g,h

43. Write a Python program to split a list into different variables.
44. Write a Python program to generate groups of five consecutive numbers in a list.
45. Write a Python program to convert a pair of values into a sorted unique array.
46. Write a Python program to select the odd items of a list.
47. Write a Python program to insert an element before each element of a list.
48. Write a Python program to print a nested lists (each list on a new line) using the print() function.
49. Write a Python program to convert list to list of dictionaries.

Sample lists: ["Black", "Red", "Maroon", "Yellow"], ["#000000", "#FF0000", "#800000", "#FFFF00"]

Expected Output: [{"color_name": "Black", "color_code": "#000000"}, {"color_name": "Red", "color_code": "#FF0000"}, {"color_name": "Maroon", "color_code": "#800000"}, {"color_name": "Yellow", "color_code": "#FFFF00"}]

50. Write a Python program to sort a list of nested dictionaries.
51. Write a Python program to split a list every Nth element.

Sample list: ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'K', 'l', 'm', 'n']

Expected Output: [['a', 'd', 'g', 'j', 'm'], ['b', 'e', 'h', 'k', 'n'], ['c', 'f', 'i', 'l']]

52. Write a Python program to compute the difference between two lists.

Sample data: ["red", "orange", "green", "blue", "white"], ["black", "yellow", "green", "blue"]

Expected Output:

Color1-Color2: ['white', 'orange', 'red']

Color2-Color1: ['black', 'yellow']

53. Write a Python program to create a list with infinite elements.
 54. Write a Python program to concatenate elements of a list.
 55. Write a Python program to remove key values pairs from a list of dictionaries.
 56. Write a Python program to convert a string to a list.
 57. Write a Python program to check if all items of a given list of strings is equal to a given string.

58. Write a Python program to replace the last element in a list with another list.

Sample data : [1, 3, 5, 7, 9, 10], [2, 4, 6, 8]

Expected Output: [1, 3, 5, 7, 9, 2, 4, 6, 8]

59. Write a Python program to check whether the n-th element exists in a given list.
 60. Write a Python program to find a tuple, the smallest second index value from a list of tuples.

61. Write a Python program to create a list of empty dictionaries.

62. Write a Python program to print a list of space-separated elements.

63. Write a Python program to insert a given string at the beginning of all items in a list.

Sample list : [1,2,3,4], string : emp

Expected output : ['emp1', 'emp2', 'emp3', 'emp4']

64. Write a Python program to iterate over two lists simultaneously.

65. Write a Python program to move all zero digits to end of a given list of numbers.

Expected output:

Original list:

[3, 4, 0, 0, 0, 6, 2, 0, 6, 7, 6, 0, 0, 0, 9, 10, 7, 4, 4, 5, 3, 0, 0, 2, 9, 7, 1]

Move all zero digits to end of the said list of numbers:

[3, 4, 6, 2, 6, 7, 6, 9, 10, 7, 4, 4, 5, 3, 2, 9, 7, 1, 0, 0, 0, 0, 0, 0, 0]

66. Write a Python program to find the list in a list of lists whose sum of elements is the highest.

Sample lists: [1,2,3], [4,5,6], [10,11,12], [7,8,9]

Expected Output: [10, 11, 12]

67. Write a Python program to find all the values in a list are greater than a specified number.

68. Write a Python program to extend a list without append.

Sample data: [10, 20, 30]

[40, 50, 60]

Expected output : [40, 50, 60, 10, 20, 30]

69. Write a Python program to remove duplicates from a list of lists.

Sample list : [[10, 20], [40], [30, 56, 25], [10, 20], [33], [40]]

New List : [[10, 20], [30, 56, 25], [33], [40]]

70. Write a Python program to find the items starts with specific character from a given list.

Expected Output:

Original list:

['abcd', 'abc', 'bcd', 'bkie', 'cder', 'cdsw', 'sdfsds', 'dagfa', 'acjd']

Items start with a from the said list:

['abcd', 'abc', 'acjd']

Items start with d from the said list:

['dagfa']

Items start with w from the said list:

[]

71. Write a Python program to check whether all dictionaries in a list are empty or not.

Sample list : [{}, {}, {}]

Return value : True

Sample list : [{1,2}, {}, {}]

Return value : False

72. Write a Python program to flatten a given nested list structure.

Original list: [0, 10, [20, 30], 40, 50, [60, 70, 80], [90, 100, 110, 120]]

Flatten list:

[0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, 120]

73. Write a Python program to remove consecutive duplicates of a given list.

Original list:

[0, 0, 1, 2, 3, 4, 4, 5, 6, 6, 6, 7, 8, 9, 4, 4]

After removing consecutive duplicates:

[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 4]

74. Write a Python program to pack consecutive duplicates of a given list elements into sublists.

Original list:

[0, 0, 1, 2, 3, 4, 4, 5, 6, 6, 6, 7, 8, 9, 4, 4]

After packing consecutive duplicates of the said list elements into sublists:

[[0, 0], [1], [2], [3], [4, 4], [5], [6, 6, 6], [7], [8], [9], [4, 4]]

75. Write a Python program to create a list reflecting the run-length encoding from a given list of integers or a given list of characters.

Original list:

[1, 1, 2, 3, 4, 4.3, 5, 1]

List reflecting the run-length encoding from the said list:

[[2, 1], [1, 2], [1, 3], [1, 4], [1, 4.3], [1, 5], [1, 1]]

Original String:

automatically

List reflecting the run-length encoding from the said string:

[[1, 'a'], [1, 'u'], [1, 't'], [1, 'o'], [1, 'm'], [1, 'a'], [1, 't'], [1, 'i'], [1, 'c'], [1, 'a'], [2, 't'], [1, 'y']]

[1, 'y']]

76. Write a Python program to create a list reflecting the modified run-length encoding from a given list of integers or a given list of characters.

Original list:

[1, 1, 2, 3, 4, 4, 5, 1]

List reflecting the modified run-length encoding from the said list:

`[[2, 1], 2, 3, [2, 4], 5, 1]`

Original String:

aabcdddadnss

List reflecting the modified run-length encoding from the said string:

`[[2, 'a'], 'b', 'c', [4, 'd'], 'a', 'd', 'n', [2, 's']]`

77. Write a Python program to decode a run-length encoded given list.

Original encoded list:

`[[2, 1], 2, 3, [2, 4], 5, 1]`

Decode a run-length encoded said list:

`[1, 1, 2, 3, 4, 4, 5, 1]`

78. Write a Python program to split a given list into two parts where the length of the first part of the list is given.

Original list:

`[1, 1, 2, 3, 4, 4, 5, 1]`

Length of the first part of the list: 3

Splited the said list into two parts:

`([1, 1, 2], [3, 4, 4, 5, 1])`

79. Write a Python program to remove the K'th element from a given list, print the new list. Original list:

`[1, 1, 2, 3, 4, 4, 5, 1]`

After removing an element at the kth position of the said list:

`[1, 1, 3, 4, 4, 5, 1]`

80. Write a Python program to insert an element at a specified position into a given list.

Original list:

`[1, 1, 2, 3, 4, 4, 5, 1]`

After inserting an element at kth position in the said list:

`[1, 1, 12, 2, 3, 4, 4, 5, 1]`

81. Write a Python program to extract a given number of randomly selected elements from a given list.

Original list:

`[1, 1, 2, 3, 4, 4, 5, 1]`

Selected 3 random numbers of the above list:

`[4, 4, 1]`

82. Write a Python program to generate the combinations of n distinct objects taken from the elements of a given list.

HINT

Original list: `[1, 2, 3, 4, 5, 6, 7, 8, 9]` Combinations of 2 distinct objects: `[1, 2] [1, 3] [1, 4] [1, 5] ... [7, 8] [7, 9] [8, 9]`

83. Write a Python program to round every number of a given list of numbers and print the total sum multiplied by the length of the list.

Original list: `[22.4, 4.0, -16.22, -9.1, 11.0, -12.22, 14.2, -5.2, 17.5]`

Result:

243

84. Write a Python program to round the numbers of a given list, print the minimum and maximum numbers and multiply the numbers by 5. Print the unique numbers in ascending order separated by space.

Original list: [22.4, 4.0, 16.22, 9.1, 11.0, 12.22, 14.2, 5.2, 17.5]

Minimum value: 4

Maximum value: 22

Result:

20 25 45 55 60 70 80 90 110

85. Write a Python program to create a multidimensional list (lists of lists) with zeros.

Multidimensional list: [[0, 0], [0, 0], [0, 0]]

86. Write a Python program to create a 3X3 grid with numbers.

3X3 grid with numbers:

[[1, 2, 3], [1, 2, 3], [1, 2, 3]]

87. Write a Python program to read a matrix from console and print the sum for each column. Accept matrix rows, columns and elements for each column separated with a space(for every row) as input from the user.

Input rows: 2

Input columns: 2

Input number of elements in a row (1, 2, 3):

1 2

3 4

sum for each column:

4 6

88. Write a Python program to read a square matrix from console and print the sum of matrix primary diagonal. Accept the size of the square matrix and elements for each column separated with a space (for every row) as input from the user.

Input the size of the matrix: 3

2 3 4

4 5 6

3 4 7

Sum of matrix primary diagonal:

14

89. Write a Python program to Zip two given lists of lists.

Original lists:

[[1, 3], [5, 7], [9, 11]]

[[2, 4], [6, 8], [10, 12, 14]]

Zipped list:

[[1, 3, 2, 4], [5, 7, 6, 8], [9, 11, 10, 12, 14]]

90. Write a Python program to count number of lists in a given list of lists.

Original list:

[[1, 3], [5, 7], [9, 11], [13, 15, 17]]

Number of lists in said list of lists:

4

Original list:

`[[2, 4], [[6, 8], [4, 5, 8]], [10, 12, 14]]`

Number of lists in said list of lists:

3

91. Write a Python program to find the list with maximum and minimum length.

Original list:

`[[0], [1, 3], [5, 7], [9, 11], [13, 15, 17]]`

List with maximum length of lists:

`(3, [13, 15, 17])`

List with minimum length of lists:

`(1, [0])`

Original list:

`[[0], [1, 3], [5, 7], [9, 11], [3, 5, 7]]`

List with maximum length of lists:

`(3, [3, 5, 7])`

List with minimum length of lists:

`(1, [0])`

Original list:

`[[12], [1, 3], [1, 34, 5, 7], [9, 11], [3, 5, 7]]`

List with maximum length of lists:

`(4, [1, 34, 5, 7])`

List with minimum length of lists:

`(1, [12])`

92. Write a Python program to check if a nested list is a subset of another nested list.

Original list:

`[[1, 3], [5, 7], [9, 11], [13, 15, 17]]`

`[[1, 3], [13, 15, 17]]`

If the one of the said list is a subset of another.:

True

Original list:

`[[[1, 2], [2, 3]], [[3, 4], [5, 6]]]`

`[[[3, 4], [5, 6]]]`

If the one of the said list is a subset of another.:

True

Original list:

`[[[1, 2], [2, 3]], [[3, 4], [5, 7]]]`

`[[[3, 4], [5, 6]]]`

If the one of the said list is a subset of another.:

False

93. Write a Python program to count the number of sublists contain a particular element.

Original list:
 [[1, 3], [5, 7], [1, 11], [1, 15, 7]]

Count 1 in the said list:

3

Count 7 in the said list:

2

Original list:
 [['A', 'B'], ['A', 'C'], ['A', 'D', 'E'], ['B', 'C', 'D']]

Count 'A' in the said list:

3

Count 'E' in the said list:

1

94. Write a Python program to count number of unique sublists within a given list.

Original list:
 [[1, 3], [5, 7], [1, 3], [13, 15, 17], [5, 7], [9, 11]]
 Number of unique lists of the said list:
 {(1, 3): 2, (5, 7): 2, (13, 15, 17): 1, (9, 11): 1}

Original list:
 [['green', 'orange'], ['black'], ['green', 'orange'], ['white']]
 Number of unique lists of the said list:
 {('green', 'orange'): 2, ('black',): 1, ('white',): 1}

95. Write a Python program to sort each sublist of strings in a given list of lists.

Original list:
 [[2], [0], [1, 3], [0, 7], [9, 11], [13, 15, 17]]
 Sort the list of lists by length and value:
 [[0], [2], [0, 7], [1, 3], [9, 11], [13, 15, 17]]

96. Write a Python program to sort a given list of lists by length and value.

Original list:
 [[2], [0], [1, 3], [0, 7], [9, 11], [13, 15, 17]]
 Sort the list of lists by length and value:
 [[0], [2], [0, 7], [1, 3], [9, 11], [13, 15, 17]]

97. Write a Python program to remove sublists from a given list of lists, which contains an element outside a given range.

Original list:
 [[2], [0], [1, 2, 3], [0, 1, 2, 3, 6, 7], [9, 11], [13, 14, 15, 17]]
 After removing sublists from a given list of lists, which contains an element outside the given range:
 [[13, 14, 15, 17]]

98. Write a Python program to scramble the letters of string in a given list.

Original list:
 ['Python', 'list', 'exercises', 'practice', 'solution']
 After scrambling the letters of the strings of the said list:
 ['tnPhyo', 'tlis', 'ecrsseiex', 'ccpitear', 'noiltuos']

99. Write a Python program to find the maximum and minimum values in a given heterogeneous list.

Original list:

`['Python', 3, 2, 4, 5, 'version']`

Maximum and Minimum values in the said list:

`(5, 2)`

100. Write a Python program to extract common index elements from more than one given list.

Original lists:

`[1, 1, 3, 4, 5, 6, 7]`

`[0, 1, 2, 3, 4, 5, 7]`

`[0, 1, 2, 3, 4, 5, 7]`

Common index elements of the said lists:

`[1, 7]`

All students are requested to solve the following Questions

101. Write a Python program to sort a given matrix in ascending order according to the sum of its rows.

Original Matrix:

`[[1, 2, 3], [2, 4, 5], [1, 1, 1]]`

Sort the said matrix in ascending order according to the sum of its rows

`[[1, 1, 1], [1, 2, 3], [2, 4, 5]]`

Original Matrix:

`[[1, 2, 3], [-2, 4, -5], [1, -1, 1]]`

Sort the said matrix in ascending order according to the sum of its rows

`[[-2, 4, -5], [1, -1, 1], [1, 2, 3]]`

102. Write a Python program to extract specified size of strings from a give list of string values.

Original list:

`['Python', 'list', 'exercises', 'practice', 'solution']`

length of the string to extract:

`8`

After extracting strings of specified length from the said list:

`['practice', 'solution']`

103. Write a Python program to extract specified number of elements from a given list, which follows each other continuously.

Original list:

`[1, 1, 3, 4, 4, 5, 6, 7]`

Extract 2 number of elements from the said list which follows each other continuously:

`[1, 4]`

Original lists:

[0, 1, 2, 3, 4, 4, 4, 4, 5, 7]

Extract 4 number of elements from the said list which follows each other continuously:

[4]

104. Write a Python program to find the difference between consecutive numbers in a given list.

Original list:

[1, 1, 3, 4, 4, 5, 6, 7]

Difference between consecutive numbers of the said list:

[0, 2, 1, 0, 1, 1, 1]

Original list:

[4, 5, 8, 9, 6, 10]

Difference between consecutive numbers of the said list:

[1, 3, 1, -3, 4]

105. Write a Python program to compute average of two given lists.

Original list:

[1, 1, 3, 4, 4, 5, 6, 7]

[0, 1, 2, 3, 4, 4, 5, 7, 8]

Average of two lists:

3.823529411764706

106. Write a Python program to count integer in a given mixed list.

Original list:

[1, 'abcd', 3, 1.2, 4, 'xyz', 5, 'pqr', 7, -5, -12.22]

Number of integers in the said mixed list:

6

107. Write a Python program to remove a specified column from a given nested list.

Original Nested list:

[[1, 2, 3], [2, 4, 5], [1, 1, 1]]

After removing 1st column:

[[2, 3], [4, 5], [1, 1]]

Original Nested list:

[[1, 2, 3], [-2, 4, -5], [1, -1, 1]]

After removing 3rd column:

[[1, 2], [-2, 4], [1, -1]]

108. Write a Python program to extract a specified column from a given nested list.

Original Nested list:

[[1, 2, 3], [2, 4, 5], [1, 1, 1]]

Extract 1st column:

[1, 2, 1]

Original Nested list:

`[[1, 2, 3], [-2, 4, -5], [1, -1, 1]]`

Extract 3rd column:

`[3, -5, 1]`

109. Write a Python program to rotate a given list by specified number of items to the right or left direction.

original List:

`[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]`

Rotate the said list in left direction by 4:

`[4, 5, 6, 7, 8, 9, 10, 1, 2, 3, 4]`

Rotate the said list in left direction by 2:

`[3, 4, 5, 6, 7, 8, 9, 10, 1, 2]`

Rotate the said list in Right direction by 4:

`[8, 9, 10, 1, 2, 3, 4, 5, 6]`

Rotate the said list in Right direction by 2:

`[9, 10, 1, 2, 3, 4, 5, 6, 7, 8]`

110. Write a Python program to find the item with maximum occurrences in a given list.

Original list:

`[2, 3, 8, 4, 7, 9, 8, 2, 6, 5, 1, 6, 1, 2, 3, 4, 6, 9, 1, 2]`

Item with maximum occurrences of the said list:

`2`

111. Write a Python program to access multiple elements of specified index from a given list.

Original list:

`[2, 3, 8, 4, 7, 9, 8, 2, 6, 5, 1, 6, 1, 2, 3, 4, 6, 9, 1, 2]`

Index list:

`[0, 3, 5, 7, 10]`

Items with specified index of the said list:

`[2, 4, 9, 2, 1]`

112. Write a Python program to check whether a specified list is sorted or not.

Original list:

`[1, 2, 4, 6, 8, 10, 12, 14, 16, 17]`

Is the said list is sorted!

`True`

Original list:

`[1, 2, 4, 6, 8, 10, 12, 14, 16, 17]`

Is the said list is sorted!

`False`

113. Write a Python program to remove duplicate dictionary from a given list.

Original list with duplicate dictionary:

`[{'Green': '#008000'}, {'Black': '#000000'}, {'Blue': '#0000FF'}, {'Green': '#008000'}]`

After removing duplicate dictionary of the said list:

`[{'Black': '#000000'}, {'Blue': '#0000FF'}, {'Green': '#008000'}]`

114. Write a Python program to extract the nth element from a given list of tuples.

Original list:

`[('Greyson Fulton', 98, 99), ('Brady Kent', 97, 96), ('Wyatt Knott', 91, 94), ('Beau Turnbull', 94, 98)]`

Extract nth element (n = 0) from the said list of tuples:

`['Greyson Fulton', 'Brady Kent', 'Wyatt Knott', 'Beau Turnbull']`

Extract nth element (n = 2) from the said list of tuples:

`[99, 96, 94, 98]`

115. Write a Python program to check if the elements of a given list are unique or not.

Original list:

`[1, 2, 4, 6, 8, 2, 1, 4, 10, 12, 14, 12, 16, 17]`

Is the said list contains all unique elements!

False

Original list:

`[2, 4, 6, 8, 10, 12, 14]`

Is the said list contains all unique elements!

True

116. Write a Python program to sort a list of lists by a given index of the inner list.

Original list:

`[('Greyson Fulton', 98, 99), ('Brady Kent', 97, 96), ('Wyatt Knott', 91, 94), ('Beau Turnbull', 94, 98)]`

Sort the said list of lists by a given index (Index = 0) of the inner list

`[('Beau Turnbull', 94, 98), ('Brady Kent', 97, 96), ('Greyson Fulton', 98, 99), ('Wyatt Knott', 91, 94)]`

Sort the said list of lists by a given index (Index = 2) of the inner list

`[('Wyatt Knott', 91, 94), ('Brady Kent', 97, 96), ('Beau Turnbull', 94, 98), ('Greyson Fulton', 98, 99)]`

117. Write a Python program to remove all elements from a given list present in another list.

Original lists:

list1: `[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]`

list2: `[2, 4, 6, 8]`

Remove all elements from 'list1' present in 'list2':

[1, 3, 5, 7, 9, 10]

118. Write a Python program to find the difference between elements (n+1th - nth) of a given list of numeric values.

Original list:

[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

Difference between elements (n+1th - nth) of the said list :

[1, 1, 1, 1, 1, 1, 1, 1, 1]

Original list:

[2, 4, 6, 8]

Difference between elements (n+1th - nth) of the said list :

[2, 2, 2]

119. Write a Python program to check if a substring presents in a given list of string values.

Original list:

['red', 'black', 'white', 'green', 'orange']

Substring to search:

ack

Check if a substring presents in the said list of string values:

True

Substring to search:

abc

Check if a substring presents in the said list of string values:

False

120. Write a Python program to create a list taking alternate elements from a given list.

Original list:

['red', 'black', 'white', 'green', 'orange']

List with alternate elements from the said list:

['red', 'white', 'orange']

Original list:

[2, 0, 3, 4, 0, 2, 8, 3, 4, 2]

List with alternate elements from the said list:

[2, 3, 0, 8, 4]

121. Write a Python program to find the nested lists elements which are present in another list.

Original lists:

[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14]

[[12, 18, 23, 25, 45], [7, 11, 19, 24, 28], [1, 5, 8, 18, 15, 16]]

Intersection of said nested lists:

[[12], [7, 11], [1, 5, 8]]

122. Write a Python program to find common element(s) in a given nested lists.

Original lists:

`[[12, 18, 23, 25, 45], [7, 12, 18, 24, 28], [1, 5, 8, 12, 15, 16, 18]]`

Common element(s) in nested lists:

`[18, 12]`

123. Write a Python program to reverse strings in a given list of string values.

Original lists:

`['Red', 'Green', 'Blue', 'White', 'Black']`

Reverse strings of the said given list:

`['deR', 'neerG', 'eulB', 'etihW', 'kcalB']`

124. Write a Python program to find the maximum and minimum product from the pairs of tuple within a given list.

The original list, tuple :

`[(2, 7), (2, 6), (1, 8), (4, 9)]`

Maximum and minimum product from the pairs of the said tuple of list:

`(36, 8)`

125. Write a Python program to calculate the product of the unique numbers of a given list.

Original List : `[10, 20, 30, 40, 20, 50, 60, 40]`

Product of the unique numbers of the said list: `720000000`

126. Write a Python program to interleave multiple lists of the same length.

Original list:

`list1: [1, 2, 3, 4, 5, 6, 7]`

`list2: [10, 20, 30, 40, 50, 60, 70]`

`list3: [100, 200, 300, 400, 500, 600, 700]`

Interleave multiple lists:

`[1, 10, 100, 2, 20, 200, 3, 30, 300, 4, 40, 400, 5, 50, 500, 6, 60, 600, 7, 70, 700]`

127. Write a Python program to remove words from a given list of strings containing a character or string.

Original list:

`list1: ['Red color', 'Orange#', 'Green', 'Orange @', 'White']`

Character list:

`['#', 'color', '@']`

New list:

`['Red', '', 'Green', 'Orange', 'White']`

128. Write a Python program to calculate the sum of the numbers in a list between the indices of a specified range.

Original list:

[2, 1, 5, 6, 8, 3, 4, 9, 10, 11, 8, 12]

Range: 8 , 10

Sum of the specified range:

29

129. Write a Python program to reverse each list in a given list of lists.

Original list of lists:

[[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12], [13, 14, 15, 16]]

Reverse each list in the said list of lists:

[[4, 3, 2, 1], [8, 7, 6, 5], [12, 11, 10, 9], [16, 15, 14, 13]]

130. Write a Python program to count the same pair in three given lists.

Original lists:

[1, 2, 3, 4, 5, 6, 7, 8]

[2, 2, 3, 1, 2, 6, 7, 9]

[2, 1, 3, 1, 2, 6, 7, 9]

Number of same pair of the said three given lists:

3

131. Write a Python program to count the frequency of consecutive duplicate elements in a given list of numbers.

Original lists:

[1, 2, 2, 2, 4, 4, 4, 5, 5, 5, 5]

Consecutive duplicate elements and their frequency:

([1, 2, 4, 5], [1, 3, 3, 4])

132. Write a Python program to find all index positions of the maximum and minimum values in a given list of numbers.

Original list:

[12, 33, 23, 10, 67, 89, 45, 667, 23, 12, 11, 10, 54]

Index positions of the maximum value of the said list:

7

Index positions of the minimum value of the said list:

3

133. Write a Python program to check common elements between two given list are in same order or not.

Original lists:

['red', 'green', 'black', 'orange']

['red', 'pink', 'green', 'white', 'black']

`['white', 'orange', 'pink', 'black']`

Test common elements between color1 and color2 are in same order?

True

Test common elements between color1 and color3 are in same order?

False

Test common elements between color2 and color3 are in same order?

False

134. Write a Python program to find the difference between two list including duplicate elements.

Original lists:

`[1, 1, 2, 3, 3, 4, 4, 5, 6, 7]`

`[1, 1, 2, 4, 5, 6]`

Difference between two said list including duplicate elements):

`[3, 3, 4, 7]`

135. Write a Python program to iterate over all pairs of consecutive items in a given list.

Original lists:

`[1, 1, 2, 3, 3, 4, 4, 5]`

Iterate over all pairs of consecutive items of the said list:

`[(1, 1), (1, 2), (2, 3), (3, 3), (3, 4), (4, 4), (4, 5)]`

136. Write a Python program to remove duplicate words from a given list of strings.

Original String:

`['Python', 'Exercises', 'Practice', 'Solution', 'Exercises']`

After removing duplicate words from the said list of strings:

`['Python', 'Exercises', 'Practice', 'Solution']`

137. Write a Python program to find a first even and odd number in a given list of numbers.

Original list:

`[1, 3, 5, 7, 4, 1, 6, 8]`

First even and odd number of the said list of numbers:

`(4, 1)`

138. Write a Python program to sort a given mixed list of integers and strings.

Numbers must be sorted before strings.

Original list:

`[19, 'red', 12, 'green', 'blue', 10, 'white', 'green', 1]`

Sort the said mixed list of integers and strings:

`[1, 10, 12, 19, 'blue', 'green', 'green', 'red', 'white']`

139. Write a Python program to sort a given list of strings(numbers) numerically.

Original list:

[4', '12', '45', '7', '0', '100', '200', '-12', '-500']

Sort the said list of strings(numbers) numerically:

[-500, -12, 0, 4, 7, 12, 45, 100, 200]

140. Write a Python program to remove the specific item from a given list of lists.

Original list of lists:

[['Red', 'Maroon', 'Yellow', 'Olive'], ['#FF0000', '#800000', '#FFFF00', '#808000'], ['rgb(255,0,0)', 'rgb(128,0,0)', 'rgb(255,255,0)', 'rgb(128,128,0)']]

Remove 1st list from the said given list of lists:

[['Maroon', 'Yellow', 'Olive'], ['#800000', '#FFFF00', '#808000'], ['rgb(128,0,0)', 'rgb(255,255,0)', 'rgb(128,128,0)']]

Remove 2nd list from the said given list of lists:

[['Red', 'Yellow', 'Olive'], ['#FF0000', '#FFFF00', '#808000'], ['rgb(255,0,0)', 'rgb(255,255,0)', 'rgb(128,128,0)']]

Remove 4th list from the said given list of lists:

[['Red', 'Maroon', 'Yellow'], ['#FF0000', '#800000', '#FFFF00'], ['rgb(255,0,0)', 'rgb(128,0,0)', 'rgb(255,255,0)']]

141. Write a Python program to remove empty lists from a given list of lists.

Original list:

[[], [], [], 'Red', 'Green', [1, 2], 'Blue', [], []]

After deleting the empty lists from the said lists of lists

['Red', 'Green', [1, 2], 'Blue']

142. Write a Python program to sum a specific column of a list in a given list of lists.

Original list of lists:

[[1, 2, 3, 2], [4, 5, 6, 2], [7, 8, 9, 5]]

Sum: 1st column of the said list of lists:

12

Sum: 2nd column of the said list of lists:

15

Sum: 4th column of the said list of lists:

9

143. Write a Python program to get the frequency of the elements in a given list of lists.

Original list of lists:

[[1, 2, 3, 2], [4, 5, 6, 2], [7, 8, 9, 5]]

Frequency of the elements in the said list of lists:

{1: 1, 2: 3, 3: 1, 4: 1, 5: 2, 6: 1, 7: 1, 8: 1, 9: 1}

144. Write a Python program to extract every first or specified element from a given two-dimensional list.

Original list of lists:

`[[1, 2, 3, 2], [4, 5, 6, 2], [7, 1, 9, 5]]`

Extract every first element from the said given two dimensional list:

`[1, 4, 7]`

Extract every third element from the said given two dimensional list:

`[3, 6, 9]`

145. Write a Python program to generate a number in a specified range except some specific numbers.

Generate a number in a specified range (1, 10) except [2, 9, 10]

`7`

Generate a number in a specified range (-5, 5) except [-5,0,4,3,2]

`-4`

146. Write a Python program to compute the sum of digits of each number of a given list.

Original tuple:

`[10, 2, 56]`

Sum of digits of each number of the said list of integers:

`14`

Original tuple:

`[10, 20, 4, 5, 'b', 70, 'a']`

Sum of digits of each number of the said list of integers:

`19`

Original tuple:

`[10, 20, -4, 5, -70]`

Sum of digits of each number of the said list of integers:

`19`

147. Write a Python program to interleave two given list into another list randomly.

Original lists:

`[1, 2, 7, 8, 3, 7]`

`[4, 3, 8, 9, 4, 3, 8, 9]`

Interleave two given list into another list randomly:

`[4, 1, 2, 3, 8, 9, 4, 3, 7, 8, 9, 8, 3, 7]`

148. Write a Python program to remove specific words from a given list.

Original list:

`['red', 'green', 'blue', 'white', 'black', 'orange']`

Remove words:

`['white', 'orange']`

After removing the specified words from the said list:
 ['red', 'green', 'blue', 'black']

149. Write a Python program to get all possible combinations of the elements of a given list.

Original list:

['orange', 'red', 'green', 'blue']

All possible combinations of the said list's elements:

[], ['orange'], ['red'], ['red', 'orange'], ['green'], ['green', 'orange'], ['green', 'red'], ['green', 'red', 'orange'], ['blue'], ['blue', 'orange'], ['blue', 'red'], ['blue', 'red', 'orange'], ['blue', 'green'], ['blue', 'green', 'orange'], ['blue', 'green', 'red'], ['blue', 'green', 'red', 'orange']]

150. Write a Python program to reverse a given list of lists.

Original list:

[['orange', 'red'], ['green', 'blue'], ['white', 'black', 'pink']]

Reverse said list of lists:

[['white', 'black', 'pink'], ['green', 'blue'], ['orange', 'red']]

Original list:

[[1, 2, 3, 4], [0, 2, 4, 5], [2, 3, 4, 2, 4]]

Reverse said list of lists:

[[2, 3, 4, 2, 4], [0, 2, 4, 5], [1, 2, 3, 4]]

151. Write a Python program to find the maximum and minimum values in a given list within specified index range.

Original list:

[4, 3, 0, 5, 3, 0, 2, 3, 4, 2, 4, 3, 5]

Index range:

3 to 8

Maximum and minimum values of the said given list within index range:

(5, 0)

152. Write a Python program to combine two given sorted lists using heapq module.

Original sorted lists:

[1, 3, 5, 7, 9, 11]

[0, 2, 4, 6, 8, 10]

After merging the said two sorted lists:

[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]

153. Write a Python program to check if a given element occurs at least n times in a list.

Original list:

[0, 1, 3, 5, 0, 3, 4, 5, 0, 8, 0, 3, 6, 0, 3, 1, 1, 0]

Check if 3 occurs at least 4 times in a list:

True

Check if 0 occurs at least 5 times in a list:

True

Check if 8 occurs at least 3 times in a list:

False

154. Write a Python program to join two given list of lists of same length, element wise.

Original lists:

[[10, 20], [30, 40], [50, 60], [30, 20, 80]]

[[61], [12, 14, 15], [12, 13, 19, 20], [12]]]

Join the said two lists element wise:

[[10, 20, 61], [30, 40, 12, 14, 15], [50, 60, 12, 13, 19, 20], [30, 20, 80, 12]]]

Original lists:

[['a', 'b'], ['b', 'c', 'd'], ['e', 'f']]

[['p', 'q'], ['p', 's', 't'], ['u', 'v', 'w']]

Join the said two lists element wise:

[['a', 'b', 'p', 'q'], ['b', 'c', 'd', 'p', 's', 't'], ['e', 'f', 'u', 'v', 'w']]

155. Write a Python program to add two given lists of different lengths, start from left.

Original lists:

[2, 4, 7, 0, 5, 8]

[3, 3, -1, 7]

Add said two lists from left:

[5, 7, 6, 7, 5, 8]

Original lists:

[1, 2, 3, 4, 5, 6]

[2, 4, -3]

Add said two lists from left:

[3, 6, 0, 4, 5, 6]

156. Write a Python program to add two given lists of different lengths, start from right.

Original lists:

[2, 4, 7, 0, 5, 8]

[3, 3, -1, 7]

Add said two lists from left:

[2, 4, 10, 3, 4, 15]

Original lists:

[1, 2, 3, 4, 5, 6]

[2, 4, -3]

Add said two lists from left:

[1, 2, 3, 6, 9, 3]

157. Write a Python program to interleave multiple given lists of different lengths.

Original lists:

[2, 4, 7, 0, 5, 8]

[2, 5, 8]

[0, 1]

[3, 3, -1, 7]

Interleave said lists of different lengths:

[2, 2, 0, 3, 4, 5, 1, 3, 7, 8, -1, 0, 7, 5, 8]

158. Write a Python program to find the maximum and minimum values in a given list of tuples.

Original list with tuples:

[('V', 60), ('VI', 70), ('VII', 75), ('VIII', 72), ('IX', 78), ('X', 70)]

Maximum and minimum values of the said list of tuples:

(78, 60)

159. Write a Python program to append the same value /a list multiple times to a list/list-of-lists.

Add a value(7), 5 times, to a list:

['7', '7', '7', '7', '7']

Add 5, 6 times, to a list:

[1, 2, 3, 4, 5, 5, 5, 5, 5, 5]

Add a list, 4 times, to a list of lists:

[[1, 2, 5], [1, 2, 5], [1, 2, 5], [1, 2, 5]]

Add a list, 3 times, to a list of lists:

[[5, 6, 7], [1, 2, 5], [1, 2, 5], [1, 2, 5]]

160. Write a Python program to remove first specified number of elements from a given list satisfying a condition.

Remove the first 4 number of even numbers from the following list:

[3, 10, 4, 7, 5, 7, 8, 3, 3, 4, 5, 9, 3, 4, 9, 8, 5]

Output:

[3, 7, 5, 7, 3, 3, 5, 9, 3, 4, 9, 8, 5]

Original list:

[3, 10, 4, 7, 5, 7, 8, 3, 3, 4, 5, 9, 3, 4, 9, 8, 5]

Remove first 4 even numbers from the said list:

[3, 7, 5, 7, 3, 3, 5, 9, 3, 4, 9, 8, 5]

161. Write a Python program to check if a given list is strictly increasing or not.

Moreover, If removing only one element from the list results in a strictly increasing list, we still consider the list true.

True

```
True
False
False
False
False
False
```

162. Write a Python program to find the last occurrence of a specified item in a given list.

Original list:

```
[s', 'd', 'f', 's', 'd', 'f', 's', 'f', 'k', 'o', 'p', 'i', 'w', 'e', 'k', 'c']
```

Last occurrence of f in the said list:

7

Last occurrence of c in the said list:

15

Last occurrence of k in the said list:

14

Last occurrence of w in the said list:

12

163. Write a Python program to get the index of the first element which is greater than a specified element.

Original list:

```
[12, 45, 23, 67, 78, 90, 100, 76, 38, 62, 73, 29, 83]
```

Index of the first element which is greater than 73 in the said list:

4

Index of the first element which is greater than 21 in the said list:

1

Index of the first element which is greater than 80 in the said list:

5

Index of the first element which is greater than 55 in the said list:

3

164. Write a Python program to get the items from a given list with specific condition.

Original list:

```
[12, 45, 23, 67, 78, 90, 45, 32, 100, 76, 38, 62, 73, 29, 83]
```

Number of Items of the said list which are even and greater than 45

5

165. Write a Python program to split a given list into specified sized chunks.

Original list:

[12, 45, 23, 67, 78, 90, 45, 32, 100, 76, 38, 62, 73, 29, 83]

Split the said list into equal size 3

[[12, 45, 23], [67, 78, 90], [45, 32, 100], [76, 38, 62], [73, 29, 83]]

Split the said list into equal size 4

[[12, 45, 23, 67], [78, 90, 45, 32], [100, 76, 38, 62], [73, 29, 83]]

Split the said list into equal size 5

[[12, 45, 23, 67, 78], [90, 45, 32, 100, 76], [38, 62, 73, 29, 83]]

166. Write a Python program to remove None value from a given list.

Original list:

[12, 0, None, 23, None, -55, 234, 89, None, 0, 6, -12]

Remove None value from the said list:

[12, 0, 23, -55, 234, 89, 0, 6, -12]

167. Write a Python program to convert a given list of strings into list of lists.

Original list of strings:

['Red', 'Maroon', 'Yellow', 'Olive']

Convert the said list of strings into list of lists:

[['R', 'e', 'd'], ['M', 'a', 'r', 'o', 'n'], ['Y', 'e', 'l', 'l', 'o', 'w'], ['O', 'l', 'i', 'v', 'e']]

168. Write a Python program to display vertically each element of a given list, list of lists.

Original list:

['a', 'b', 'c', 'd', 'e', 'f']

Display each element vertically of the said list:

a

b

c

d

e

f

Original list:

[[1, 2, 5], [4, 5, 8], [7, 3, 6]]

Display each element vertically of the said list of lists:

1 4 7

2 5 3

5 8 6

169. Write a Python program to convert a given list of strings and characters to a single list of characters.

Original list:

['red', 'white', 'a', 'b', 'black', 'f']

Convert the said list of strings and characters to a single list of characters:

['r', 'e', 'd', 'w', 'h', 'i', 't', 'e', 'a', 'b', 'b', 'l', 'a', 'c', 'k', 'f']

170. Write a Python program to insert an element in a given list after every nth position.

Original list:

[1, 2, 3, 4, 5, 6, 7, 8, 9, 0]

Insert a in the said list after 2 nd element:

[1, 2, 'a', 3, 4, 'a', 5, 6, 'a', 7, 8, 'a', 9, 0]

Insert b in the said list after 4 th element:

[1, 2, 3, 4, 'b', 5, 6, 7, 8, 'b', 9, 0]

171. Write a Python program to concatenate element-wise three given lists.

Original lists:

['0', '1', '2', '3', '4']

['red', 'green', 'black', 'blue', 'white']

['100', '200', '300', '400', '500']

Concatenate element-wise three said lists:

['0red100', '1green200', '2black300', '3blue400', '4white500']

172. Write a Python program to remove the last N number of elements from a given list.

Original lists:

[2, 3, 9, 8, 2, 0, 39, 84, 2, 2, 34, 2, 34, 5, 3, 5]

Remove the last 3 elements from the said list:

[2, 3, 9, 8, 2, 0, 39, 84, 2, 2, 34, 2, 34]

Remove the last 5 elements from the said list:

[2, 3, 9, 8, 2, 0, 39, 84, 2, 2, 34]

Remove the last 1 element from the said list:

[2, 3, 9, 8, 2, 0, 39, 84, 2, 2, 34, 2, 34, 5, 3]

173. Write a Python program to merge some list items in given list using index value.

Original lists:

['a', 'b', 'c', 'd', 'e', 'f', 'g']

Merge items from 2 to 4 in the said List:

['a', 'b', 'cd', 'e', 'f', 'g']

Merge items from 3 to 7 in the said List:

['a', 'b', 'c', 'defg']

174. Write a Python program to add a number to each element in a given list of numbers.

Original lists:

[3, 8, 9, 4, 5, 0, 5, 0, 3]

Add 3 to each element in the said list:

[6, 11, 12, 7, 8, 3, 8, 3, 6]

Original lists:

[3.2, 8, 9.9, 4.2, 5, 0.1, 5, 3.11, 0]

Add 0.51 to each element in the said list:

[3.71, 8.51, 10.41, 4.71, 5.51, 0.61, 5.51, 3.62, 0.51]

175. Write a Python program to find the minimum, maximum value for each tuple position in a given list of tuples.

Original list:

[(2, 3), (2, 4), (0, 6), (7, 1)]

Maximum value for each tuple position in the said list of tuples:

[7, 6]

Minimum value for each tuple position in the said list of tuples:

[0, 1]

176. Write a Python program to create a new list dividing two given lists of numbers.

Original list:

[7, 2, 3, 4, 9, 2, 3]

[7, 2, 3, 4, 9, 2, 3]

[0.7777777777777778, 0.25, 1.5, 1.333333333333333, 3.0, 2.0, 1.5]

177. Write a Python program to find common elements in a given list of lists.

Original list:

[[7, 2, 3, 4, 7], [9, 2, 3, 2, 5], [8, 2, 3, 4, 4]]

Common elements of the said list of lists:

[2, 3]

Original list:

[['a', 'b', 'c'], ['b', 'c', 'd'], ['c', 'd', 'e']]

Common elements of the said list of lists:

['c']

178. Write a Python program to insert a specified element in a given list after every nth element.

Original list:

[1, 3, 5, 7, 9, 11, 0, 2, 4, 6, 8, 10, 8, 9, 0, 4, 3, 0]

Insert 20 in said list after every 4 th element:

[1, 3, 5, 7, 20, 9, 11, 0, 2, 20, 4, 6, 8, 10, 20, 8, 9, 0, 4, 20, 3, 0]

Original list:

['s', 'd', 'f', 'j', 's', 'a', 'j', 'd', 'f', 'd']

Insert Z in said list after every 3 th element:

['s', 'd', 'f', 'Z', 'j', 's', 'a', 'Z', 'j', 'd', 'f', 'Z', 'd']

179. Write a Python program to create the largest possible number using the elements of a given list of positive integers.

Original list:

[3, 40, 41, 43, 74, 9]

Largest possible number using the elements of the said list of positive integers:

9744341403

Original list:

[10, 40, 20, 30, 50, 60]

Largest possible number using the elements of the said list of positive integers:

605040302010

Original list:

[8, 4, 2, 9, 5, 6, 1, 0]

Largest possible number using the elements of the said list of positive integers:

98654210

180. Write a Python program to create the smallest possible number using the elements of a given list of positive integers.

Original list:

[3, 40, 41, 43, 74, 9]

Smallest possible number using the elements of the said list of positive integers:

3404143749

Original list:

[10, 40, 20, 30, 50, 60]

Smallest possible number using the elements of the said list of positive integers:

102030405060

Original list:

[8, 4, 2, 9, 5, 6, 1, 0]

Smallest possible number using the elements of the said list of positive integers:

01245689

181. Write a Python program to iterate a given list cyclically on specific index position.

Original list:

['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h']

Iterate the said list cyclically on specific index position 3 :

['d', 'e', 'f', 'g', 'h', 'a', 'b', 'c']

Iterate the said list cyclically on specific index position 5 :

['f', 'g', 'h', 'a', 'b', 'c', 'd', 'e']

182. Write a Python program to calculate the maximum and minimum sum of a sublist in a given list of lists.

Original list:

[[1, 2, 3, 5], [2, 3, 5, 4], [0, 5, 4, 1], [3, 7, 2, 1], [1, 2, 1, 2]]

Maximum sum of sub list of the said list of lists:

[2, 3, 5, 4]

Minimum sum of sub list of the said list of lists:

[1, 2, 1, 2]

183. Write a Python program to get the unique values in a given list of lists.

Original list:

`[[1, 2, 3, 5], [2, 3, 5, 4], [0, 5, 4, 1], [3, 7, 2, 1], [1, 2, 1, 2]]`

Unique values of the said list of lists:

`[0, 1, 2, 3, 4, 5, 7]`

Original list:

`[['h', 'g', 'l', 'k'], ['a', 'b', 'd', 'e', 'c'], ['j', 'i', 'y'], ['n', 'b', 'v', 'c'], ['x', 'z']]`

Unique values of the said list of lists:

`['e', 'd', 'c', 'b', 'x', 'k', 'n', 'h', 'g', 'j', 'i', 'a', 'l', 'y', 'v', 'z']`

184. Write a Python program to form Bigrams of words in a given list of strings.

From Wikipedia:

A bigram or digram is a sequence of two adjacent elements from a string of tokens, which are typically letters, syllables, or words. A bigram is an n-gram for n=2. The frequency distribution of every bigram in a string is commonly used for simple statistical analysis of text in many applications, including in computational linguistics, cryptography, speech recognition, and so on.

Original list:

`['Sum all the items in a list', 'Find the second smallest number in a list']`

Bigram sequence of the said list:

`[('Sum', 'all'), ('all', 'the'), ('the', 'items'), ('items', 'in'), ('in', 'a'), ('a', 'list'), ('Find', 'the'), ('the', 'second'), ('second', 'smallest'), ('smallest', 'number'), ('number', 'in'), ('in', 'a'), ('a', 'list')]`

185. Write a Python program to convert a given decimal number to binary list.

Original Number: 8

Decimal number (8) to binary list:

`[1, 0, 0, 0]`

Original Number: 45

Decimal number (45) to binary list:

`[1, 0, 1, 1, 0, 1]`

Original Number: 100

Decimal number (100) to binary list:

`[1, 1, 0, 0, 1, 0, 0]`

186. Write a Python program to swap two sublists in a given list.

Original list:

`[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]`

Swap two sublists of the said list:

`[0, 6, 7, 8, 9, 3, 4, 5, 1, 2, 10, 11, 12, 13, 14, 15]`

Swap two sublists of the said list:

`[0, 9, 3, 8, 6, 7, 4, 5, 1, 2, 10, 11, 12, 13, 14, 15]`

187. Write a Python program to convert a given list of tuples to a list of strings.

Original list of tuples:

[('red', 'green'), ('black', 'white'), ('orange', 'pink')]

Convert the said list of tuples to a list of strings:

['red green', 'black white', 'orange pink']

Original list of tuples:

[('Laiba', 'Delacruz'), ('Mali', 'Stacey', 'Drummond'), ('Raja', 'Welch'), ('Saarah', 'Stone')]

Convert the said list of tuples to a list of strings:

['Laiba Delacruz', 'Mali Stacey Drummond', 'Raja Welch', 'Saarah Stone']

188. Write a Python program to sort a given list of tuples on specified element.

Original list of tuples:

[('item2', 10, 10.12), ('item3', 15, 25.1), ('item1', 11, 24.5), ('item4', 12, 22.5)]

Sort on 1st element of the tuple of the said list:

[('item1', 11, 24.5), ('item2', 10, 10.12), ('item3', 15, 25.1), ('item4', 12, 22.5)]

Sort on 2nd element of the tuple of the said list:

[('item2', 10, 10.12), ('item1', 11, 24.5), ('item4', 12, 22.5), ('item3', 15, 25.1)]

Sort on 3rd element of the tuple of the said list:

[('item2', 10, 10.12), ('item4', 12, 22.5), ('item1', 11, 24.5), ('item3', 15, 25.1)]

189. Write a Python program to shift last element to first position and first element to last position in a given list.

Original list:

[1, 2, 3, 4, 5, 6, 7]

Shift last element to first position and first element to last position of the said list:

[7, 2, 3, 4, 5, 6, 1]

Original list:

['s', 'd', 'f', 'd', 's', 's', 'd', 'f']

Shift last element to first position and first element to last position of the said list:

['f', 'd', 'f', 'd', 's', 's', 'd', 's']

190. Write a Python program to find the specified number of largest products from two given list, multiplying an element from each list.

Original lists:

[1, 2, 3, 4, 5, 6]

[3, 6, 8, 9, 10, 6]

3 Number of largest products from the said two lists:

[60, 54, 50]

4 Number of largest products from the said two lists:

[60, 54, 50, 48]

191. Write a Python program to find the maximum and minimum value of the three given lists.

Original lists:

[2, 3, 5, 8, 7, 2, 3]

[4, 3, 9, 0, 4, 3, 9]

[2, 1, 5, 6, 5, 5, 4]

Maximum value of the said three lists:

9

Minimum value of the said three lists:

0

192. Write a Python program to remove all strings from a given list of tuples.

Original list:

[(100, 'Math'), (80, 'Math'), (90, 'Math'), (88, 'Science', 89), (90, 'Science', 92)]

Remove all strings from the said list of tuples:

[(100,), (80,), (90,), (88, 89), (90, 92)]

193. Write a Python program to find the dimension of a given matrix.

Original list:

[[1, 2], [2, 4]]

Dimension of the said matrix:

(2, 2)

Original list:

[[0, 1, 2], [2, 4, 5]]

Dimension of the said matrix:

(2, 3)

Original list:

[[0, 1, 2], [2, 4, 5], [2, 3, 4]]

Dimension of the said matrix:

(3, 3)

194. Write a Python program to sum two or more lists, the lengths of the lists may be different.

Original list:

[[1, 2, 4], [2, 4, 4], [1, 2]]

Sum said lists with different lengths:

[4, 8, 8]

Original list:

[[1], [2, 4, 4], [1, 2], [4]]

Sum said lists with different lengths:

[8, 6, 4]

195. Write a Python program to traverse a given list in reverse order, also print the elements with original index.

Original list:

`['red', 'green', 'white', 'black']`

Traverse the said list in reverse order:

black

white

green

red

Traverse the said list in reverse order with original index:

3 black

2 white

1 green

0 red

196. Write a Python program to move a specified element in a given list.

Original list:

`['red', 'green', 'white', 'black', 'orange']`

Move white at the end of the said list:

`['red', 'green', 'black', 'orange', 'white']`

Original list:

`['red', 'green', 'white', 'black', 'orange']`

Move red at the end of the said list:

`['green', 'white', 'black', 'orange', 'red']`

Original list:

`['red', 'green', 'white', 'black', 'orange']`

Move black at the end of the said list:

`['red', 'green', 'white', 'orange', 'black']`

197. Write a Python program to compute the average of nth elements in a given list of lists with different lengths.

Original list:

`[[0, 1, 2], [2, 3, 4], [3, 4, 5, 6], [7, 8, 9, 10, 11], [12, 13, 14]]`

Average of n-th elements in the said list of lists with different lengths:

`[4.8, 5.8, 6.8, 8.0, 11.0]`

198. Write a Python program to compare two given lists and find the indices of the values present in both lists.

Original lists:

`[1, 2, 3, 4, 5, 6]`

`[7, 8, 5, 2, 10, 12]`

Compare said two lists and get the indices of the values present in both lists:

`[1, 4]`

Original lists:

`[1, 2, 3, 4, 5, 6]`

`[7, 8, 5, 7, 10, 12]`

Compare said two lists and get the indices of the values present in both lists:

[4]

Original lists:

[1, 2, 3, 4, 15, 6]

[7, 8, 5, 7, 10, 12]

Compare said two lists and get the indices of the values present in both lists:

[]

199. Write a Python program to convert a given unicode list to a list contains strings.

Original lists:

['S001', 'S002', 'S003', 'S004']

Convert the said unicode list to a list contains strings:

['S001', 'S002', 'S003', 'S004']

200. Write a Python program to pair up the consecutive elements of a given list.

Original lists:

[1, 2, 3, 4, 5, 6]

Pair up the consecutive elements of the said list:

[[1, 2], [2, 3], [3, 4], [4, 5], [5, 6]]

Original lists:

[1, 2, 3, 4, 5]

Pair up the consecutive elements of the said list:

[[1, 2], [2, 3], [3, 4], [4, 5]]

All students are requested to solve the following Questions

202. Write a Python program to find the indexes of all None items in a given list.

Original list:

[1, None, 5, 4, None, 0, None, None]

Indexes of all None items of the list:

[1, 4, 6, 7]

203. Write a Python program to join adjacent members of a given list.

Original list:

['1', '2', '3', '4', '5', '6', '7', '8']

Join adjacent members of a given list:

['12', '34', '56', '78']

Original list:

['1', '2', '3']

Join adjacent members of a given list:

['12']

204. Write a Python program to check if first digit/character of each element in a given list is same or not.

Original list:

[1234, 122, 1984, 19372, 100]

Check if first digit in each element of the said given list is same or not!

True

Original list:

[1234, 922, 1984, 19372, 100]

Check if first digit in each element of the said given list is same or not!

False

Original list:

['aabc', 'abc', 'ab', 'a']

Check if first character in each element of the said given list is same or not!

True

Original list:

['aabc', 'abc', 'ab', 'ha']

Check if first character in each element of the said given list is same or not!

False

205. Write a Python program to find the indices of elements of a given list, greater than a specified value.

Original list:

[1234, 1522, 1984, 19372, 1000, 2342, 7626]

Indices of elements of the said list, greater than 3000

[3, 6]

Original list:

[1234, 1522, 1984, 19372, 1000, 2342, 7626]

Indices of elements of the said list, greater than 20000

[]

206. Write a Python program to remove additional spaces in a given list.

Original list:

['abc', ' ', ' ', 'sdfds', ' ', ' ', 'sdfds', 'huy']

Remove additional spaces from the said list:

['abc', "", "", 'sdfds', "", "", 'sdfds', 'huy']

207. Write a Python program to find the common tuples between two given lists.

Original lists:

[('red', 'green'), ('black', 'white'), ('orange', 'pink')]

[('red', 'green'), ('orange', 'pink')]

Common tuples between two said lists

[('orange', 'pink'), ('red', 'green')]

Original lists:

[('red', 'green'), ('orange', 'pink')]

[('red', 'green'), ('black', 'white'), ('orange', 'pink')]
 Common tuples between two said lists
 [('orange', 'pink'), ('red', 'green')]

208. Sum a list of numbers. Write a Python program to sum the first number with the second and divide it by 2, then sum the second with the third and divide by 2, and so on.

Original list:

[1, 2, 3, 4, 5, 6, 7]

Sum the said list of numbers:

[1.5, 2.5, 3.5, 4.5, 5.5, 6.5]

Original list:

[0, 1, -3, 3, 7, -5, 6, 7, 11]

Sum the said list of numbers:

[0.5, -1.0, 0.0, 5.0, 1.0, 0.5, 6.5, 9.0]

209. Write a Python program to count the number of groups of non-zero numbers separated by zeros of a given list of numbers.

Original list:

[3, 4, 6, 2, 0, 0, 0, 0, 0, 0, 0, 6, 7, 6, 9, 10, 0, 0, 0, 0, 0, 5, 9, 9, 7, 4, 4, 0, 0, 0, 0, 0, 0, 5, 3, 2, 9, 7, 1, 0, 0, 0]

Number of groups of non-zero numbers separated by zeros of the said list: 4

210. Write a Python program to compute the sum of non-zero groups (separated by zeros) of a given list of numbers.

Original list:

[3, 4, 6, 2, 0, 0, 0, 0, 0, 0, 0, 6, 7, 6, 9, 10, 0, 0, 0, 0, 0, 7, 4, 4, 0, 0, 0, 0, 0, 0, 5, 3, 2, 9, 7, 1, 0, 0, 0]

Compute the sum of non-zero groups (separated by zeros) of the said list of numbers: [15, 38, 15, 27]

211. Write a Python program to remove an element from a given list.

Original list:

['Guido Van Rossum', 98, 'Math', 90, 'Science']

After deleting an element:, using index of the element: [98, 'Math', 90, 'Science']

212. Write a Python program to remove all the values except integer values from a given array of mixed values.

Original list: [34.67, 12, -94.89, 'Python', 0, 'C#']

After removing all the values except integer values from the said array of mixed values: [12, 0]

213. Write a Python program to calculate the sum of two lowest negative numbers of a given array of integers.

Original list elements: [-14, 15, -10, -11, -12, -13, 16, 17, 18, 19, 20]
 Sum of two lowest negative numbers of the said array of integers: -27
 Original list elements: [-4, 5, -2, 0, 3, -1, 4, 9]
 Sum of two lowest negative numbers of the said array of integers: -6

214. Write a Python program to sort a given positive number in descending/ascending order.

Descending -> Highest to lowest.

Ascending -> Lowest to highest

Original Number: 134543

Descending order of the said number: 544331

Ascending order of the said number: 133445

Original Number: 43750973

Descending order of the said number: 97754330

Ascending order of the said number: 3345779

215. Write a Python program to merge two or more lists into a list of lists, combining elements from each of the input lists based on their positions.

Sample Output:

After merging lists into a list of lists:

`[['a', 1, True], ['b', 2, False]]`

`[['a', 1, True], [None, 2, False]]`

`[['a', 1, True], ['_', 2, False]]`

216. Write a Python program to group the elements of a list based on the given function and returns the count of elements in each group.

Sample Output:

`{6: 2, 4: 1}`

`{3: 2, 5: 1}`

217. Write a Python program to split values into two groups, based on the result of the given filtering function.

Sample Output:

`[['white'], ['red', 'green', 'black']]`

218. Write a Python program to sort one list based on another list containing the desired indexes.

Sample Output:

`['apples', 'bread', 'eggs', 'jam', 'milk', 'oranges']`

`['oranges', 'milk', 'jam', 'eggs', 'bread', 'apples']`

219. Write a Python program to build a list, using an iterator function and an initial seed value.

Sample Output:

`[-10, -20, -30, -40]`

220. Write a Python program to map the values of a list to a dictionary using a function, where the key-value pairs consist of the original value as the key and the result of the function as the value.

Sample Output:

{1: 1, 2: 4, 3: 9}

221. Write a Python program to randomize the order of the values of an list, returning a new list.

Sample Output:

Original list: [1, 2, 3, 4, 5, 6]

Shuffle the elements of the said list:

[3, 2, 4, 1, 6, 5]

222. Write a Python program to get the difference between two given lists, after applying the provided function to each list element of both.

Sample Output:

[1.2]

[{'x': 2}]

223. Write a Python program to create a list with the non-unique values filtered out.

Sample Output:

[1, 3, 5]

224. Write a Python program to create a list with the unique values filtered out.

Sample Output:

[2, 4]

225. Write a Python program to retrieve the value of the nested key indicated by the given selector list from a dictionary or list.

Sample Output:

Harwood

2

226. Write a Python program to get a list of elements that exist in both lists, after applying the provided function to each list element of both.

Sample Output:

[2.1]

227. Write a Python program to get the symmetric difference between two lists, after applying the provided function to each list element of both.

Sample Output:
[1.2, 3.4]

228. Write a Python program to get every element that exists in any of the two given lists once, after applying the provided function to each element of both.

Sample Output:
[2.2, 4.1]

229. Write a Python program to find the index of the first element in the given list that satisfies the provided testing function.

Sample Output:
0

230. Write a Python program to find the indexes of all elements in the given list that satisfy the provided testing function.

Sample Output:
[0, 2]

231. Write a Python program to split values into two groups, based on the result of the given filter list.

Sample Output:
[['red', 'green', 'pink'], ['blue']]

232. Write a Python program to chunk a given list into smaller lists of a specified size.

Sample Output:
[[1, 2, 3], [4, 5, 6], [7, 8]]

233. Write a Python program to chunk a given list into n smaller lists.

Sample Output:
[[1, 2], [3, 4], [5, 6], [7]]

234. Write a Python program to convert a given number (integer) to a list of digits.

Sample Output:
[1, 2, 3]
[1, 3, 4, 7, 8, 2, 3]

235. Write a Python program to find the index of the last element in the given list that satisfies the provided testing function.

Sample Output:
2

236. Write a Python program to find the items that are parity outliers in a given list.

Sample Output:

[1, 3]

[2, 4, 6]

237. Write a Python program to convert a given list of dictionaries into a list of values corresponding to the specified key.

Sample Output:

[8, 36, 34, 10]

238. Write a Python program to calculate the average of a given list, after mapping each element to a value using the provided function.

Sample Output:

5.0

15.0

239. Write a Python program to find the value of the first element in the given list that satisfies the provided testing function.

Sample Output:

1

2

240. Write a Python program to find the value of the last element in the given list that satisfies the provided testing function.

Sample Output:

3

4

241. Write a Python program to create a dictionary with the unique values of a given list as keys and their frequencies as the values.

Sample Output:

{'a': 4, 'b': 2, 'f': 2, 'c': 1, 'e': 2}

{3: 4, 4: 2, 7: 1, 5: 2, 9: 1, 0: 1, 2: 1}

242. Write a Python program to get the symmetric difference between two iterables, without filtering out duplicate values.

Sample Output:

[30, 40]

243. Write a Python program to check if a given function returns True for every element in a list.

Sample Output:

True

False

False

244. Write a Python program to initialize a list containing the numbers in the specified range where start and end are inclusive and the ratio between two terms is step. Returns an error if step equals 1.

Sample Output:

[1, 2, 4, 8, 16, 32, 64, 128, 256]

[3, 6, 12, 24, 48, 96, 192]

[1, 4, 16, 64, 256]

245. Write a Python program to that takes any number of iterable objects or objects with a length property and returns the longest one.

Sample Output:

Green

[1, 2, 3, 4, 5]

[1, 2, 3, 4]

246. Write a Python program to check if a given function returns True for at least one element in the list.

Sample Output:

True

False

247. Write a Python program to calculate the difference between two iterables, without filtering duplicate values.

Sample Output:

[3]

248. Write a Python program to get the maximum value of a list, after mapping each element to a value using a given function.

Sample Output:

8

249. Write a Python program to get the minimum value of a list, after mapping each element to a value using a given function.

Sample Output:

2

250. Write a Python program to calculate the sum of a list, after mapping each element to a value using the provided function.

Sample Output:

20

251. Write a Python program to initialize and fills a list with the specified value.

Sample Output:

[0, 0, 0, 0, 0, 0, 0]

[3, 3, 3, 3, 3, 3, 3, 3]

[-2, -2, -2, -2, -2]

[3.2, 3.2, 3.2, 3.2, 3.2]

252. Write a Python program to get the n maximum elements from a given list of numbers.

Sample Output:

Original list elements:

[1, 2, 3]

Maximum values of the said list: [3]

Original list elements:

[1, 2, 3]

Two maximum values of the said list: [3, 2]

Original list elements:

[-2, -3, -1, -2, -4, 0, -5]

Threee maximum values of the said list: [0, -1, -2]

Original list elements:

[2.2, 2, 3.2, 4.5, 4.6, 5.2, 2.9]

Two maximum values of the said list: [5.2, 4.6]

253. Write a Python program to get the n minimum elements from a given list of numbers.

Sample Output:

Original list elements:

[1, 2, 3]

Minimum values of the said list: [1]

Original list elements:

[1, 2, 3]

Two minimum values of the said list: [1, 2]

Original list elements:

[-2, -3, -1, -2, -4, 0, -5]

Threee minimum values of the said list: [-5, -4, -3]

Original list elements:

[2.2, 2, 3.2, 4.5, 4.6, 5.2, 2.9]

Two minimum values of the said list: [2, 2.2]

254. Write a Python program to get the weighted average of two or more numbers.

Sample Output:

Original list elements:

[10, 50, 40]

[2, 5, 3]

Weighted average of the said two list of numbers:

39.0

Original list elements:

[82, 90, 76, 83]

[0.2, 0.35, 0.45, 32]

Weighted average of the said two list of numbers:

82.97272727272727

255. Write a Python program to perform a deep flattens a list.

Sample Output:

Original list elements:

[1, [2], [[3], [4], 5], 6]

Deep flatten the said list:

[1, 2, 3, 4, 5, 6]

Original list elements:

[[[1, 2, 3], [4, 5]], 6]

Deep flatten the said list:

[1, 2, 3, 4, 5, 6]

256. Write a Python program to get the powerset of a given iterable.

Sample Output:

Original list elements:

[1, 2]

Powerset of the said list:

[(), (1,), (2,), (1, 2)]

Original list elements:

[1, 2, 3, 4]

Powerset of the said list:

[(), (1,), (2,), (3,), (4,), (1, 2), (1, 3), (1, 4), (2, 3), (2, 4), (3, 4), (1, 2, 3), (1, 2, 4), (1, 3, 4), (2, 3, 4), (1, 2, 3, 4)]

257. Write a Python program to check if two given lists contain the same elements regardless of order.

Sample Output:

Original list elements:

[1, 2, 4]

[2, 4, 1]

Check two said lists contain the same elements regardless of order!

True

Original list elements:

[1, 2, 3]

[1, 2, 3]

Check two said lists contain the same elements regardless of order!

True

Original list elements:

[1, 2, 3]

[1, 2, 4]

Check two said lists contain the same elements regardless of order!

False

258. Write a Python program to create a given flat list of all the keys in a flat dictionary.

Sample Output:

Original directory elements:

{'Laura': 10, 'Spencer': 11, 'Bridget': 9, 'Howard ': 10}

Flat list of all the keys of the said dictionary:

['Laura', 'Spencer', 'Bridget', 'Howard ']

259. Write a Python program to check if a given function returns True for at least one element in the list.

Sample Output:

False

True

False

260. Write a Python program to check if all the elements of a list are included in another given list.

Sample Output:

True

False

261. Write a Python program to get the most frequent element in a given list of numbers.

Sample Output:

2

Original list:

[2, 3, 8, 4, 7, 9, 8, 2, 6, 5, 1, 6, 1, 2, 3, 2, 4, 6, 9, 1, 2]

Item with maximum frequency of the said list:

2

Original list:

[1, 2, 3, 1, 2, 3, 2, 1, 4, 3, 3]

Item with maximum frequency of the said list:

3

262. Write a Python program to move the specified number of elements to the end of the given list.

Sample Output:

[4, 5, 6, 7, 8, 1, 2, 3]

[6, 7, 8, 1, 2, 3, 4, 5]

[1, 2, 3, 4, 5, 6, 7, 8]

[1, 2, 3, 4, 5, 6, 7, 8]

[8, 1, 2, 3, 4, 5, 6, 7]

[2, 3, 4, 5, 6, 7, 8, 1]

263. Write a Python program to move the specified number of elements to the start of the given list.

Sample Output:

[4, 5, 6, 7, 8, 1, 2, 3]

[6, 7, 8, 1, 2, 3, 4, 5]

[1, 2, 3, 4, 5, 6, 7, 8]

[1, 2, 3, 4, 5, 6, 7, 8]

[8, 1, 2, 3, 4, 5, 6, 7]

[2, 3, 4, 5, 6, 7, 8, 1]

264. Write a Python program to create a two-dimensional list from given list of lists.

Sample Output:

```
[(1, 4, 7, 10), (2, 5, 8, 11), (3, 6, 9, 12)]
[(1, 4), (2, 5)]
```

265. Write a Python program to generate a list, containing the Fibonacci sequence, up until the nth term.

Sample Output:

First 7 Fibonacci numbers:

```
[0, 1, 1, 2, 3, 5, 8, 13]
```

First 15 Fibonacci numbers:

```
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610]
```

First 50 Fibonacci numbers:

```
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597, 2584, 4181, 6765, 10946, 17711, 28657, 46368, 75025, 121393, 196418, 317811, 514229, 832040, 1346269, 2178309, 3524578, 5702887, 9227465, 14930352, 24157817, 39088169, 63245986, 102334155, 165580141, 267914296, 433494437, 701408733, 1134903170, 1836311903, 2971215073, 4807526976, 7778742049, 12586269025]
```

266. Write a Python program to cast the provided value as a list if it's not one.

Sample Output:

```
<class 'list'>
[1]
<class 'tuple'>
['Red', 'Green']
<class 'set'>
['Green', 'Red']
<class 'dict'>
[1, 2, 3]
```

267. Write a Python program to get the cumulative sum of the elements of a given list.

Sample Output:

Original list elements:

```
[1, 2, 3, 4]
```

Cumulative sum of the elements of the said list:

```
[1, 3, 6, 10]
```

Original list elements:

```
[-1, -2, -3, 4]
```

Cumulative sum of the elements of the said list:

```
[-1, -3, -6, -2]
```

268. Write a Python program to get a list with n elements removed from the left, right.

Sample Output:

Original list elements:

[1, 2, 3]

Remove 1 element from left of the said list:

[2, 3]

Remove 1 element from right of the said list:

[1, 2]

Original list elements:

[1, 2, 3, 4]

Remove 2 elements from left of the said list:

[3, 4]

Remove 2 elements from right of the said list:

[1, 2]

Original list elements:

[1, 2, 3, 4, 5, 6]

Remove 7 elements from left of the said list:

[2, 3, 4, 5, 6]

Remove 7 elements from right of the said list:

[1, 2, 3, 4, 5]

269. Write a Python program to get the every nth element in a given list.

Sample Output:

[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

[2, 4, 6, 8, 10]

[5, 10]

[6]

270. Write a Python program to check if the elements of the first list are contained in the second one regardless of order.

Sample Output:

True

True

False

True

271. Write a Python program to check if there are duplicate values in a given flat list.

Sample Output:

Original list:

[1, 2, 3, 4, 5, 6, 7]

Check if there are duplicate values in the said given flat list:

False

Original list:

[1, 2, 3, 3, 4, 5, 5, 6, 7]

Check if there are duplicate values in the said given flat list:

True

272. Write a Python program to generate a list of numbers in the arithmetic progression starting with the given positive integer and up to the specified limit.

Sample Output:

[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]

[3, 6, 9, 12, 15, 18, 21, 24, 27, 30, 33, 36]

[5, 10, 15, 20, 25]
